HLM12ERC, a Multimodal Model for Emotion Recognition in Conversations: **Final Report**

Deep Learning on Public Dataset

CM3070, Final Computer Science Project

TABLE OF CONTENTS

INTRODUCTION

The present document reports on the project that reviewed relevant literature, designed, implemented & evaluated the HLM12ERC model, a solution proposed for emotion classification on multimodal multi-party conversations.

Template Chosen

The template chosen for this project was the **Deep Learning on a public dataset**, and the Dataset selected was the **MELD: A Multimodal Multi-Party Dataset for Emotion Recognition in Conversations**[Poria et al, 2019].

Github Repository

Code, notebooks, tests and instructions to run the model HLM12ERC model can be found at https://github.com/hudsonmendes/uolondon-cm3070-fp.

Motivation, Domain & Users

Assessing emotions during interactions is arguably one of the most essential social skills that humans have. And in order to assess other people's emotions, we utilise not just information from multiple senses such as hearing vision, but also from internal states that we carry and presume others to carry based on previous interactions.

Not surprisingly, enabling machines to be able to recognize emotions in conversations finds a large number of commercial applications like marketing and management, user interaction, finance, politics, health and education[Kratzwald et al, 2018].

To date, many models have attempted to address the problem of emotion recognition in conversations. However, the state of the art performance is still low at around 67% Weighted F1-score[PapersWithCode, 2023].

Public Dataset: MELD

The data set selected was the MELD dataset, a multimodal multi party dataset for emotion recognition in conversations published by Poria et al., in 2019, that extended the previously existing EmotionLines dataset[Poria et al, 2019].

The MELD dataset adds a sentiment polarity label on top of the EmotionLines data set, extends its emotion classes, and also delivers a significant quality improvement with a Fleiss' Kappa 26.5% superior to the original.

Preliminary Critique of Previous Work

Preliminarily, research work directly related to the MELD dataset[Poria et al, 2019], as well as state-of-the-art models that published results against the MELD dataset benchmark[PapersWithCode, 2023], were reviewed and are briefly discussed below.

Previous Work #1: MELD dataset & baselines

The MELD Dataset was introduced by Poria et al in 2019 as the product of a research project recorded into their paper[Poria et al, 2019], which offers invaluable information about the dataset itself, but also about ERC models used as baselines to evaluate the quality of the work produced.

While the EmotionLines dataset presents $14,503$ records, the MELD dataset, which was based on the EmotionLines dataset, has $13,708$ records, as a consequence of efforts in removing corrupt records. The MELD dataset was also entirely relabelled and achieved a superior Fleiss' Kappa Score of 0.43, a 25% increase when compared to the Fleiss' Kappa Score of 0.34 achieved by the EmotionLines dataset[Poria et al, 2019].

The MELD dataset also produced statistics for a number of other baseline models for emotion classification trained and evaluated on its data, namely the text-CNN, the bcLSTM, the DialogRNN[Majumder et al, 2019], from which the latter obtained the highest Weighted F1-score at 60.25% over its 7 classes: anger, disgust, fear, joy, neutral, sadness and surprise[Poria et al, 2019].

Previous Work #2: DialogRNN model

The second work reviewed was the research paper DialogRNN by Majumder et al from 2019[Majumder et al, 2019].

Its proposed model architecture is based on multiple gated recurrent units, or GRUs, ensembled together to model each speaker's context as well as the dialogue's context, and use these combined representations as an input to the emotion GRU responsible for outputting the emotion class.

This model leverages the three modalities of the input text, audio, and video. It also leverages the speaker's context as well as the dialogue's context. However, GRUs are limited in terms of CPU-level data parallelization due to their recursive nature. They also pay more attention to recent interactions instead of long range dependencies, which, as reported, led to classification mistakes.

Previous Work #3: SPCL-CL-ERC model

The third paper reviewed was the SPCL-CL-ERC[Song et al, 2022], that introduced a contrastive loss function and used it to fine tune the SimCSE model[Gao et al, 2021] in an attempt to build an embedding space that increases distance between contrastive sentence representations.

Also, curriculum learning based on a custom distance function was used to eliminate extreme cases that would either require the other modalities for disambiguation or could have a detrimental effect on the representations.

SPCL-CL-ERC uses a queue to ensure that each mini-batch has enough positive and negative examples of each class, regardless of the batch size, preventing an issue known to usually affect contrastive learning.

It also makes use of attention-based text representations, like BERT, and improves these representations with a sophisticated Contrastive Loss function which led it to achieve state-of-the-art results on the MELD benchmark.

However, the paper does mention about the multimodal nature of the data set and the fact that emotions sometimes can only be captured by looking at the different modalities, but leaves this problem unaddressed only looking at the textual modality.

Previous Work #4: M2FNet model

The fourth paper reviewed was the M2FNet model[Chudasama et al, 2022], at present, the third best weight F1-score in the MELD benchmark ranking[PapersWithCode, 2023].

This paper employs some truly bleeding edge techniques such as RoBERTa based text representations, audio representations trained from MEL features and even face recognition for videos. Later, it combines and transforms these into a dialogue embedding. An MLP classifier then transforms these representations into an emotion class.

Contrary to SPCL-CL-ERC, M2FNet also leverages the three modalities of information (text, audio and video). However, it does not model speakers individually as DialogRNN does.

Additionally, the M2FNet model ensemble is large and relies on a fairly heavy feature extraction process that has been seen as an opportunity for simplification.

Project Aims & Objectives

After taking into consideration the strengths and weaknesses of the models presented by the papers reviewed in the critique of previous work, the present project was laid out.

This project aimed to perform emotion recognition conversations or ERC on the MELD dataset having, as its main goal, to design, train and evaluate a model both inspired in the existing literature but also assembled in an novel way.

The baseline architecture used for this project took inspiration from the M2FNet model. However, although the M2FNet has had its results published in the Papers With Code Ranking, its source code was not made available. As a consequence, its architectural design has been recreated from scratch by the present project, reproducing some of its features to the best of the available knowledge.

The project investigated, through a sequence of rigorous experiments, (a) the suitability of employing pre-trained auto-regressive text representations over the entire dialogue, (b) the possible benefits of restricting visual information to the images of faces rather than feeding information about the entire scene as visual features, (c) the superiority of using a state-of-the-art pre-trained audio feature extractor over simpler representations, (d) the differences of fusing features using attention rather than simple concatenation, (e) as well as the potential of complementing the classification loss with a loss based on contrastive learning.

Results of experimentation, assets and code produced by the present project are also provided and conveniently open-sourced, with the purpose of enabling researchers to utilise and further the HLM12ERC model.

LITERATURE REVIEW

Emotion Recognition in Conversations (or "ERC") recurringly appears classified by researchers in the field as a relatively complex task, especially in non-dyadic settings where there are more than two people engaging in a dialogue.

Poria et al (2019) describe that the ERC task "presents several challenges such as conversational context modelling, emotion shift of the interlocutors"[Poria et al, 2019] and that these challenges "make the task more difficult to address"[Poria et al, 2019]. They illustrate a situation stating that "[u]tterances like 'yeah', 'okay', 'no' can express varied emotions depending on the context and discourse of the dialogue (…) most models resort to assigning the majority class"[Poria et al, 2019].

However, despite its challenges, the vast applications of ERC[Kratzwald et al, 2018] including "opinion mining over chat history and social media threads in YouTube, Facebook, Twitter"[Majumder et al, 2019], continues to draw attention to its problem space, given that any improvements in the field can provide the industries that they serve with significant breakthrough.

The present literature review explores papers of solutions proposed for the ERC problem space, specifically the ones trained and evaluated over the MELD Dataset[Poria et al, 2019], breaks down the models proposed into their building blocks, and draws a comparison of components that are present in each one of them, and highlights possible implementation gaps that can be explored for follow-up experimentation.

Scope of Research & Sparsity of Solution Space

Many more models proposed to solve the problem of Emotion Recognition in Conversation than the present literature review could possibly analyse, and a criteria to prioritise the models, and their respective literature, has been devised focusing exclusively on solutions introduced by papers which have reported against the ECR over MELD benchmark[PapersWithCode, 2023].

To date, 45 papers have proposed solutions for the ERC problem and published results against the MELD dataset benchmark. These papers describe models clustering together as being either multi-modal models or textual-only models.

Though they have been devised to solve the same problem, their architecture varies remarkably, and rarely any of the models are mere continuation or few steps improvements of previously available models, most of them opting to recreating model architectures to solve the ERC problem from scratch.

That phenomena creates a vast and sparsely explored solution space, with many possible combinations of building blocks yet to be explored, and require non-trivial judgement to perform this combination properly.

Definition of "Emotions" in the context of ERC

"Emotions are the unseen mental states that are linked to thoughts and feelings"[Mihalceae et al, 2019]. And while there have been some breakthroughs in using brain activity for inference, these could implicate serious ethical concerns and still require significant equipment to be attached to the person[Li et al, 2023], not viable to the applications for which ERC is relevant[Kratzwald et al, 2018].

From that limitation, Chudasama et al. (2022) concludes that "[i]n the absence of physiological indications, [emotions] could only be detected by human actions such as textual utterances, visual gestures, and acoustic signals"[Chudasama et al, 2022], which indicates that to solve ERC effectively, one must take in consideration the multi-modality of ERC data.

Multiple authors seem to agree that exploring multi-modality is important to solve the ERC problem[Chudasama et al, 2022][Majumder et al, 2019][Wu et al, 2019]. However many others utilise a single modality, usually textual[Song et al, 2022], and even without leveraging data from the other modalities, still outperform multimodal models, as for example the SPCL-CL-ERC, the current state-of-the-art model in ERC over the MELD dataset.

Research Paper 1: MELD Dataset

EmotionLines is an emotion corpus of multi-party (non-dyadic) conversations developed by Chen et al. (2018), that "has evolved from the EmotionLines dataset"[Poria et al, 2019] and delivered a number of

improvements. It's based on dialogues extracted from the TV Series "Friends" containing two or more people engaging in dialogue.

MELD, opposed to EmotionLines, "provides multimodal sources"[Poria et al, 2019] and over 13,000 utterances. It was re-annotated using "Ekman's six universal emotions (Joy, Sadness, Fear, Anger, Surprise, and Disgust)"[Poria et al, 2019] and "two additional emotion labels: Neutral and Non-Neutral"[Poria et al, 2019].

MELD introduced improvements to the annotation process. For EmotionLines, annotation was based on Amazon Mechanical Turk (AMT), annotators only looked at the transcriptions, and their domain over the English language could not be asserted. For MELD, "[t]he annotators were graduate students with high proficiency in English speaking and writing"[Poria et al, 2019] and "were briefed about the annotation process with a few examples"[Poria et al, 2019]. These led MELD to a superior Fleiss kappa score of 0.43 against 0.34 achieved by EmotionLines.

Research Paper 2: DialogRNN Model

The first model investigated is the DialogRNN used as one of the baseline models in the MELD Dataset paper[Poria et al, 2019]. It presents a series of GRUs (or "Gated Recurrent Units") that update different states, individually, and all feed into one another to perform the emotion classification task[Majumder et al, 2019].

The entire model can be formalised by the following set of equations:

- Let the t be a given timestep, u_t be the utterance the timestep t
- Let $T(u_t)$, $V(u_t)$, $A(u_t)$ be the functions that performs textual feature extraction, video feature extraction and audio feature extraction respectively on a given utterance;
- X_t be the concatenation of the text, audio and video representations at time t into a single feature vector;
- Let $q_{P,t}$ be the state of any individual speaker P, g_{t-1} be the dialogue context state, and e_t be the emotion label state;
- Let $GRU_P(g_{t-1}, X_t)$, $GRU_G(g_{P,t-1}, X_t)$ and $GRU_E(e_{t-1}, q_{P,t})$ be the Gated Recurrent Units responsible for generating representations for the individual speaker state, to the global dialogue state and to the individual speaker emotion states respectively;

The prediction y_{pred} of the emotion state e_t is given by the following equation:

$$
\begin{aligned}
X_t &= T(u_t) \circ V(u_t) \circ A(u_t) \\
q_{P,t} &= GRU_P(g_{t-1}, X_t) \\
g_{t-1} &= GRU_G(g_{P,t-1}, X_t) \\
y_{pred} &= e_t = GRU_E(e_{t-1}, q_{P,t})
\end{aligned}
$$

As it's possible to observe from the set of equations above, the y_{pred} is mapped from a number of chained dependencies. Namely the g_{t-1}, which is the previous dialogue global state used to compute the $q_{P,t}$, which is the current state of the speaker P, which is then finally input into the GRU_E, alongside the previous emotion states to generate the emotion label $y_{pred} = e_t$.

The authors also propose and comparatively evaluate variations of their base model architecture. The variant $BiDialogRNN + at_{MM}$ is the best performing one, and makes use of an extra Attention layer responsible for learning relationships between individual utterances.

During their ablation studies, Majumder et al demonstrates that "party state stands very important, as without its presence the performance falls by 4.33%"[Majumder et al, 2019], whereas the "Emotion GRU['s] (...) absence causes performance to fall by only 2.51%"[Majumder et al, 2019], which indicates that somehow keeping record of the individual state of individuals might imply more accurate emotion classification.

Research Paper 3: M2FNet Model

M2FNet is a multi-modal fusion network model which "takes advantage of the multi-modal nature of real-world media content [by combining] features from different modalities to generate rich emotion-relevant representations"[Chudasama et al, 2022].

This model largely leverages the power of the attention mechanism in multiple levels of its architecture, including (a) the usage of a Text Encoder which is based on a feature map generated by a modified version of the RoBERTa encoder, the $\phi_{M}RoBERTa$[Chudasama et al, 2022], and (b) the introduction of a "multi-head attention-based fusion layer, which aids the proposed system to combine latent representations of the different inputs"[Chudasama et al, 2022].

While DialogRNN does not utilise pre-trained text encoders, M2FNet leverages the pretrained features provided by RoBERTa to perform textual feature extraction. For the audio and video modalities, M2FNet introduces two non-trivial feature extraction modules and leverages attention to embeddings for each, and trains its model with a newly introduced "adaptive margin-based triplet loss function"[Chudasama et al, 2022].

The feature-maps from all modalities are then fed into the fusion-network that produces a representation which is then used for the emotion classification.

Differently to the DialogRNN model, the M2FNet does not model the individual speaker context separately to the dialogue context. Instead it suggests a (a) "utterance level feature extraction"[Chudasama et al, 2022] stage, where the multi-modal features for each one the utterances is extracted separately, followed by a (b) "dialogue level feature extraction"[Chudasama et al, 2022], where "[e]ach modality embeddings (...) are passed through their corresponding network with a variable stack of transformer encoders (...) to learn the inter utterance context"[Chudasama et al, 2022]. Both the "utterance level" (through a residual connection[Chollet, 2017]) and "dialogue level" features are then input into the fusion mechanism.

M2FNet is a large and complex model and, due to its complexity, it is not possible to capture enough of its features in a simplified set of equations that describe it elegantly without missing on important details of its architecture. Given its vast use of transformer models in its ensemble, M2FNet may also be considerably computationally expensive, but its computational cost is neither provided in the paper nor by related work.

Chudasama et al claims that results demonstrate that their "fusion mechanism helps to enhance the accuracy (...) for IEMOCAP and MELD datasets" and "$F1_{score}$ for the IEMOCAP dataset"[Chudasama et al, 2022], and the ablation tests indicate that 5 "Attention Fusion Layers" slightly outperformed 6 "Attention Fusion Layers".

Additionally, the paper proposes that features extracted from both the scene and faces can be combined using a "weighted Face Model", which outperformed representations created by either source on its own.

Research Paper 4: SPCL-CL-ERC Model

In opposition to DialogRNN and M2FNet previously described, the novelty introduced by the SPCL-CL-ERC model is not a substantially different model architecture, but instead a loss function, the "Supervised Prototypical Contrastive Learning (SPCL) loss"[Song et al, 2022] and a sampling strategy which "ensures that each sample has at least one positive sample of the same category and negative samples of all other categories within a mini-batch"[Song et al, 2022] and combined "curriculum learning (...) with contrastive learning"[Song et al, 2022].

Curriculum Learning is a training approach which states that hypothetically "a well chosen curriculum strategy can act as a continuation method (...), i.e., can help to find better local minima of a non-convex training criterion"[Bengio et al, 2009]. Song et al achieve that by "sorting the training data via [a difficulty] function, we can schedule the training data in an easy-to- hard fashion"[Song et al, 2022].

Similarly to M2FNet, and differently to DialogRNN, SPCL-CL-ERC uses the pre-trained SimCSE model[Gao et al, 2021] that leverages the Transformer architecture, but that also trains sentence representation using a Triplet Contrastive Learning approach to improve the separation between embeddings of contradictory utterances.

And even though SPCL-CL-ERC does not leverage the multi-modality of the MELD dataset, it ranks first in the With Code Ranking, with an Weighted F1-Score 0.81% better than M2FNet[PapersWithCode, 2023] that does leverage the multi-modality of the dataset.

Worth of mention, SPCL-CL-ERC also makes use of a technique recently popularised, novel in the context of ERC, called prompt engineering[Liu et al, 2023] which is a text generation technique based on the "pre-train, prompt, and predict"[Liu et al, 2023] approach and has been employed to introduce individual speaker context to the dialogue context using a "prompt-based context encoder"[Song et al, 2022].

The algorithm proposed can be summarised in the following way:

- Let ε be the label set of emotions;
- Let $P_t = for\ u_t, s_t\ fells < mask >$ be the prompt composed, u_t be the textual utterance, all at time t[8];
- Let $C_t = [(s_{t-k}, u_{t-k}), (s_{t-k+1}, u_{t-k+1}), \ldots, (s_t, u_t)]$ be the dialogue context at time t composed by k tuples (s_t, u_t)[8];
- Let $H_t^k = SimCSE(C_t \oplus P_t)$ be the model that takes in a sequence of prompts and returns hidden states for each token at time t, the H_t^k[8];
- Let $G(z_i, z_j)$ be the cosine similarity and τ be a scalar temperature parameter[8]
- Let $I = [z_1, z_2, \cdots, z_N]$ be the batch of size N with all the representations produced by the context encoder, extracted from H_t^k, $A(i) = \{z_x \in I \mid x \neq i\}$ the set of all encoded representations different to z_i, and $P(i) = \{z_x \in I \mid y_x = y_i\}$ the set of all positive examples[8], $E(i) = |\{\varepsilon_i \in \varepsilon \mid \varepsilon_i = y_i\}|$ the set of all emotion labels equal to y_i;
- Let $Q_i = [z_1^i, z_2^i, \ldots, z_M^i]$ be the representation queue containing at most the M most recent representations generated by the $SimCSE(C_t \oplus P_t)$ encoder;

The total loss L_{spcl} is computed with modified versions of the the contrastive loss function components $N_{sup}(i)$ and $P_{sup}(i)$, named the $N_{spcl}(i)$ and $P_{spcl}(i)$ respectively:

$$F(z_i, z_j) = exp(G(z_i, z_j)/\tau)$$

$$N_{sup}(i) = \sum_{z_j \in A(i)} F(z_i, z_j)$$

$$\rho_{sup}(i) = \sum_{z_j \in P(i)} F(z_i, z_j)$$

$$S_k = RandomSelect(Q_i, k)$$

$$T_k = \frac{1}{k} \sum_{z_j^i \in S_k, j \in [1..k]}^{i} z_j^i$$

$$Ty_i = \frac{1}{k} \sum_{z_i \in P(i)}^{i} z_i$$ [see footnote #1]

$$N_{spcl}(i) = N_{sup}(i) + \sum_{k \in E(i)}^{i} F(z_i, T_k)$$

$$P_{spcl}(i) = P_{sup}(i) + F(z_i, Ty_i)$$

$$L_i^{spcl} = -log(\frac{1}{P(i)} \frac{P_{spcl}(i)}{N_{spcl}(i)})$$

$$L_{spcl} = \sum_{i=0}^{N} L_i^{spcl}$$

To predict the emotion label SPCL authors "train an additional linear layer to predict the labels using cross-entropy loss"[Song et al, 2022] that utilise the fine-tuned SimCSE representations for classification.

Given that the SPCL-CL-ERC model does not operate on all modalities, it also relies on curriculum learning to eliminate extreme cases (most ambiguous) from training examples of its encoder and, to achieve that, employs a "difficulty measure"[Song et al, 2022] that determines that

[1] The equation for the term Ty_i has not been provided by the author, and it was inferred based on the information provided in the paper[8] as well in the source code provided with the paper, source: https://github.com/caskcsg/SPCL/blob/cd489397df93dc43bbe20e0e57b0814438666ad9/spcl_loss.py

"the closer the sample is to the category centre, the lower the difficulty"[Song et al, 2022]. The authors provide a qualitative analysis where they claim that a sample-based curriculum learning strategy has proven to be more efficient than direct sorting of the data using the difficulty function which led the "model [to] overfit on simple samples in the early stage of training and produce large losses in the later stage"[Song et al, 2022].

Critique

All papers investigated base their ideas on solid literature, all of them providing comparative results between theirs and existing alternative solutions, and most also open sourced their code. They also provide reasonable ablation studies demonstrating how each one of the underlying components brought into their models provide the benefits they were intended for.

RNNs (and similarly GRU and LSTM), present in the architecture of the DialogRNN model, are known to be inefficient in learning longer-range dependencies in due to the fact that "gradient descent becomes increasingly inefficient when the temporal span of the dependencies increases."[Simard et al, 1994]. However, the authors of DialogRNN do attempt to compensate for that deficiency with an additional Attention mechanism responsible for learning long range dependencies[Majumder et al, 2019].

For natural language processing, RNNs have been largely replaced by transformer-based models, as it is noticeable in the two other models, M2FNet and SPCL-CL-ERC, introduced more recently at a time that the benefits of attention mechanism had over learning long-term dependencies in text[Chudasama et al, 2022][Song et al, 2022] had become more widely accepted.

While M2FNet deploys a hierarchy of several transformer models with multiple attention heads[Chudasama et al, 2022], SPCL-CL-ERC demonstrates that state-of-the-art Weighted F1-Score can be reached operating only over the textual modality alone[Song et al, 2022][PapersWithCode, 2023], with a single transformers based model (SimCSE) if optimised appropriately, which could indicate that there is unnecessary complexity in the M2FNet ensemble that could be eliminated to produce a simpler and more efficient model. However, results on different benchmarks suggest that M2FNet performs better than SPCL-CL-ERC for different datasets[PapersWithCode/IEMOCAP, 2023].

Another noteworthy aspect present in the literature is that modelling a dialogue so that it is observed as a sequence of interactions, and that the arising emotions are a function of our internal states produced by that sequence seems to draw from a shared robust intuition. DialogRNN encodes that intuition directly into its architecture by making use of GRUs[Majumder et al, 2019] that always utilise previous states when predicting new states. Sharing conceptual similarity, M2FNet also encodes that intuition in its architecture by passing each modality utterance embedding "through their corresponding network with a variable stack of transformer encoders (...) to learn the inter utterance context"[Chudasama et al, 2022].

Conversely, SPCL-CL-ERC has a fairly unique approach to modelling context of utterances to allow for inference of emotions for unseen data which does not attempt to learn the relationships between the the present and previous utterances, but that does attempt to create a relationship between utterances by relating them to a category centroid, the so called "prototypical representations"[Song et al, 2022].

Whereas it's debatable whether the benefit of leveraging the multi-modality of the data, or whether sequence of interactions, individual relationship of utterances, and emotions as functions of that sequence, are or aren't fundamental aspects of dialogues, different data distributions with different densities for any of these features may show that one model architecture responds to change and is more robust than the others, something that could potentially be backed up by the difference in model ranking on an alternative benchmark: IEMOCAP[PapersWithCode/IEMOCAP, 2023].

On that aspect, a common issue across the papers investigated is that none of them discussed the distributions of data available from their benchmark. Also there was no attempt to discuss the datasets (e.g.: dialogues from the TV Series "Friends") correlated with ERC data captured from real-world scenarios. This could be a fundamental problem, given that different data skews could significantly limit any model's ability to operate on a different dataset.

DESIGN

The present design introduces a model "HLM12ERC" capable of high Weighted F1-Score for the emotion classification task against the Test Split of the MELD Dataset[Poria et al, 2019]. To pursue that aim, the following objectives are set out:

Objective 1: Create a baseline ERC model inspired in the M2FNet model architecture[Chudasama et al, 2022] with "statistical power"[Chollet, 2017] against the MELD test set taking the multimodal input from the MELD dataset[Poria et al, 2019], concatenating embedded representations of the input, predicting an emotion label and employing an objective function appropriate to the target metric;

Objective 2: Evaluate whether the introduction of prompt-engineering to the model, similar to the one used by SPCL-CL-ERC[Song et al, 2022] paper, but using the entire dialogue context as prior to producing utterance representations, using the auto-regressive language model GPT2[Radford, 2019], outperforms the simple GloVe-based text representations;

Objective 3: Evaluate whether using an encoder based on pre-trained general purpose audio embedding model such as Wave2Vec 2.0[Baevski et al, 2020] outperforms the encoder based on projections of raw wavelength information from audio files;

Objective 4: Evaluate whether a model using ResNet-50 representations based exclusively on the faces of the actors, detected using a pre-trained model such as Tina-Face(ResNet-50)[Yanjia et al, 2021] outperforms the baseline representations of the entire scene;

Objective 5: Evaluate whether a similar fusion network to the one introduced by the M2FNet model[Chudasama et al, 2022] outperforms the simple concatenation of features;

Objective 6: Evaluate whether a triplet contrastive loss such as the one used by the SPCL-CL-ERC model[Song et al, 2022], or another appropriate triplet contrastive loss, outperforms the baseline loss function;

Objective 7: Compile & Evaluate final model based on best performing components.

Model Architecture

As previously described, in-person settings of conversations rely on context provided by multiple senses. To deliver to the target users & domains, the model architecture must address the multi-modality of the data and, consequently, regardless of components chosen through experimentation, it will likely result in a model ensemble of significant complexity.

The design here proposed (a) produces embeddings for each one of the modalities, (b) combines these into a single vector representation of fixed size, (c) transforms it hierarchically into higher level representations through a feedforward network so they can be finally used to (d) produces a label through a softmax activation, as illustrated by **Figure 1**.

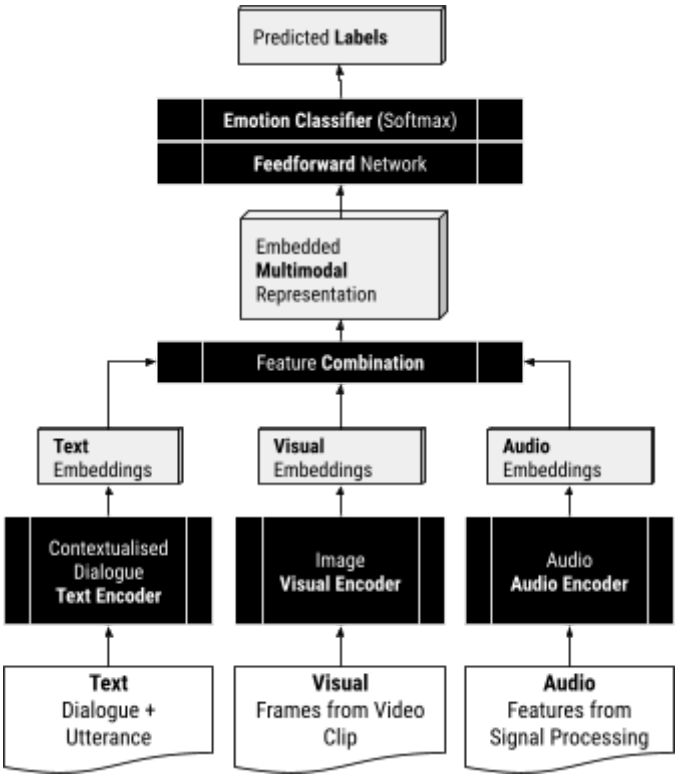

Figure 1: *HLM12ERC High-level model architecture.*

Objective Function & Class Imbalance

The MELD Dataset suffers from significant class-imbalance and the class "neural" is predominant, appearing as ground truth label to approximately 47% of the records (**Table 1**). This situation causes both the Accuracy Metric, as well as the usual Cross Entropy, which weights errors for all classes in the same fashion, to be suboptimal choices.

Class	Frequency	%
Neutral	4710	47%
Joy	1743	17%
Surprise	1205	12%
Anger	1109	11%
Sadness	683	7%
Disgust	271	3%
Fear	268	3%

Table 1: *Emotion Class Frequence in MELD.*
Source: dev/eda.ipynb

Amongst the objective functions usually utilised as alternatives to the Cross Entropy Loss function for multi-class class imbalanced datasets, two are investigated as part **of Objective** 1 to produce a baseline model that learns all the classes rather than just blindly guessing the majority class to obtain a high accuracy. These functions explored are the Dice Loss[Milletari et al, 2016] and the Focal Loss[Lin et al, 2017].

While the F1-Score is non-differentiable and cannot be used to produce an objective function, the Dice Loss is based off the Dice Coefficient and has been proposed as a Surrogate Loss function that can help to optimise the search through the hypothesis space for a model that scores towards higher f1-score[Milletari et al, 2016].

An alternative to the Dice Loss, the Focal Loss[Lin et al, 2017], essentially proposes a weighting mechanism, where weights defined as a hyperparameter α help to increase the error attributed to minority classes.

Algorithms & Equations

Several different algorithms will be investigated by the present project and, if they result in significant gains against the MELD test set[Poria et al, 2019], compose the final model.

Due to their number, complexity and to constraints in the number of pages the present document is limited to, the present section will provide a comprehensive list with a brief description of what these components are, their links to their source research and the reason for which they were chosen.

Baseline Text Representation, GloVe

GloVe token embeddings that are learned through "global word-word co-occurrence"[Pennington et al, 2014]. Using the mean of the token vectors is a common practice in NLP and it is used as the baseline for the textual representation.

Baseline Visual Representation, RestNet-50

The baseline model used for visual features is the ResNet-50[He et al, 2016], which is the foundational model used by TinaFace(ResNet-50) proposed as the Advanced Visual Embedding. By employing the ResNet-50 as the baseline, the results of the advanced approach shall become easier to observe.

Advanced Text Representation, GPT2

Both M2FNet and SPCL-CL-ERC use variants of the RoBERTa model which provides a maximum context-window of 512 tokens. M2FNet uses "a variable stack of transformer encoders (...) to learn the inter utterance context"[Chudasama et al, 2022] and does not model individual speaker's context, while SPCL-CL-ERC uses prototypical vectors rather than learning inter utterance context in any way, but it models the speaker individual context through prompts[Song et al, 2022].

Conversely, the present project attempts to improve the dialogue context modelling by encoding the entire dialogue using a prompt design approach[Liu et al, 2023] with the purpose of benefiting from the Transformers architecture's well known ability to effectively capture long-term dependencies and context information from its fixed input size[Vaswani et al, 2017]. However, a context-window of 512 is not sufficient for that purpose.

The Exploratory Data Analysis revealed that, using the NLTK Word Tokeniser, the longest concatenated dialogue in respect to its number of tokens is 574 tokens, with 75% of the dialogues having 188 tokens or less[Mendes/EDA, 2023].

The GPT2 model supports a context window of 1024 tokens[Radford, 2019], which is sufficient for the textual data distribution found in MELD. Using a representation based on the last hidden layer of GPT2 embeddings as the advanced text representations will be evaluated under Objective 2.

Advanced Visual Representation, TinaFace

While the entire video scene of the utterance might have important disambiguation information to which is the right emotion of the speaker, the present work hypothesise that face features carry better signal-to-noise ratio, and will evaluate whether focusing exclusively on faces with TinaFace(ResNet-50)[Yanjia et al, 2021] can improve the model Weighted F1-Score when compared to simply processing the entire image with the ResNet-50[12] baseline, under Objective 3.

Advanced Audio Representation, Wave2Vec2

The input data for Wave2Vec 2.0 is the raw waveform of the audio[Baevski et al, 2020], the same input data as the baseline model. However, the baseline model attempts to learn all its representations from the training data. Wave2Vec 2.0, on the other hand, carries forward information from its pre-training phase, which may prove valuable and drive higher Weighted F1-Score. Hence, Wave2Vec 2.0 will be evaluated as an alternative to the baseline audio encoder under Objective 4.

Advanced Feature Fusion, Fusion Network

There are multiple techniques that allow feature fusion, with Mean and Concatenation operations being common approaches. However, meaning for instance combines features irrespective of their importance, while concatenation increases the dimensionality of the hypothesis space being explored. An alternative approach, also explored by M2FNet[Chudasama et al, 2022] is allowing the attention mechanism to learn which relationships between features are more relevant than others[Vaswani et al, 2017]. A Fusion Network will be investigated as an alternative to multimodal feature fusion.

Advanced Loss, Contrastive Learning

A triplet loss function based on an anchor, a positive and a negative example can be used to increase distance between contradictory examples and reduce distance between similar examples[Schroff et al, 2015], resulting in better separation of data in the embeddings space. Such an approach is explored by both M2FNet[Chudasama et al, 2022] and SPCL-CL-ERC[Song et al, 2022] models. The present project will also evaluate whether this can drive an improvement in terms of accuracy.

Libraries & Technologies

The software for the present project will be written in python, and make extensive use of popular ML python packages, such as the ones described in this section.

PyTorch & HuggingFace for Model & Training

HLM12ERC will be modelled and packaged as a PyTorch[Paszke et al, 2019] model. However, instead of writing the training loop manually, the HuggingFace Trainer[Wolf et al, 2020] class shall be used.

Sklearn for Metrics

SKLearn[Pedregosa et al, 2011] has a well tested and widely adopted metrics module, which will be used to report on the Weighted F1-Score as well as to produce classification reports.

Pandas, Librosa, Numpy

The MELD dataset is composed of text, videos and audio, and will be represented in the project as Pandas[Pandas, 2020] dataframes. Dataframes, as well as audio read by Librosa[McFee et al, 2023] is stored in memory as a Numpy[Harris et al, 2020] array, which can easily be converted to PyTorch tensors.

Code Structure

Code-wise, the project will be organised as a (a) jupyter notebook **dev/mlops.ipynb**, responsible for the MLOps pipeline and (b) a python package "hlm12erc" with the model, training and serving code, used by the jupyter notebook, in the following structure:

```
./src/hlm12erc/modelling/
./src/hlm12erc/training/
./src/hlm12erc/serving/
./dev/mlops.ipynb
./tests/
./pyproject.toml
./README.md
./.gitignore
```

The $pyproject.toml$ file provides all the relevant metadata that allows the software to be installed from source as a python package, allowing code edits during development at the same time it allows modules to be imported by a package name.

Success Criteria: Weighted F1-Score

Due to the presence of the substantial class-imbalance in the MELD Dataset discussed previously, experiments have been considered successful over another whenever they outperformed the latter in terms of **Weighted F1-Score** over the MELD Validation Split, during the development phase.

Evaluation Protocol: Hold Out Test Approach

The Evaluation of each component in isolation (Objectives 1 to 6), as well as the final model (for Objective 7) have been carried out with a Hold Out Test-set approach[Chollet, 2017].

The choice of the Hold Out approach has been encouraged by the fact that the MELD Dataset[Poria et al, 2019] is provided with three splits: train, dev and test.

All experiments have utilised exclusively the training and validation splits during development. Upon completion, the baseline model, as well as the best performing model, were evaluated against the test split.

Risks

Two main aspects inherent to training multimodal models have been identified and addressed in the early stages of development:

(a) Complexity of the raw data: MELD has 3 different modalities, one of the being a video file with audio-visual features; Preparation of the data is crucial to reduce the complexity of the data, as well as make it more manageable;

(b) Computational expense: raw video/image or audio information are represented by large vectors. Any model processing them requires training millions of parameters, causing the issue of training time to be unavoidable and the usage of hardware acceleration (GPUs) to be essential.

Work Plan

The project phases were inspired by the Double Diamond of Interaction Design[Sharp et al, 2019] and represented by the Gantt Chart from **Figure 2**

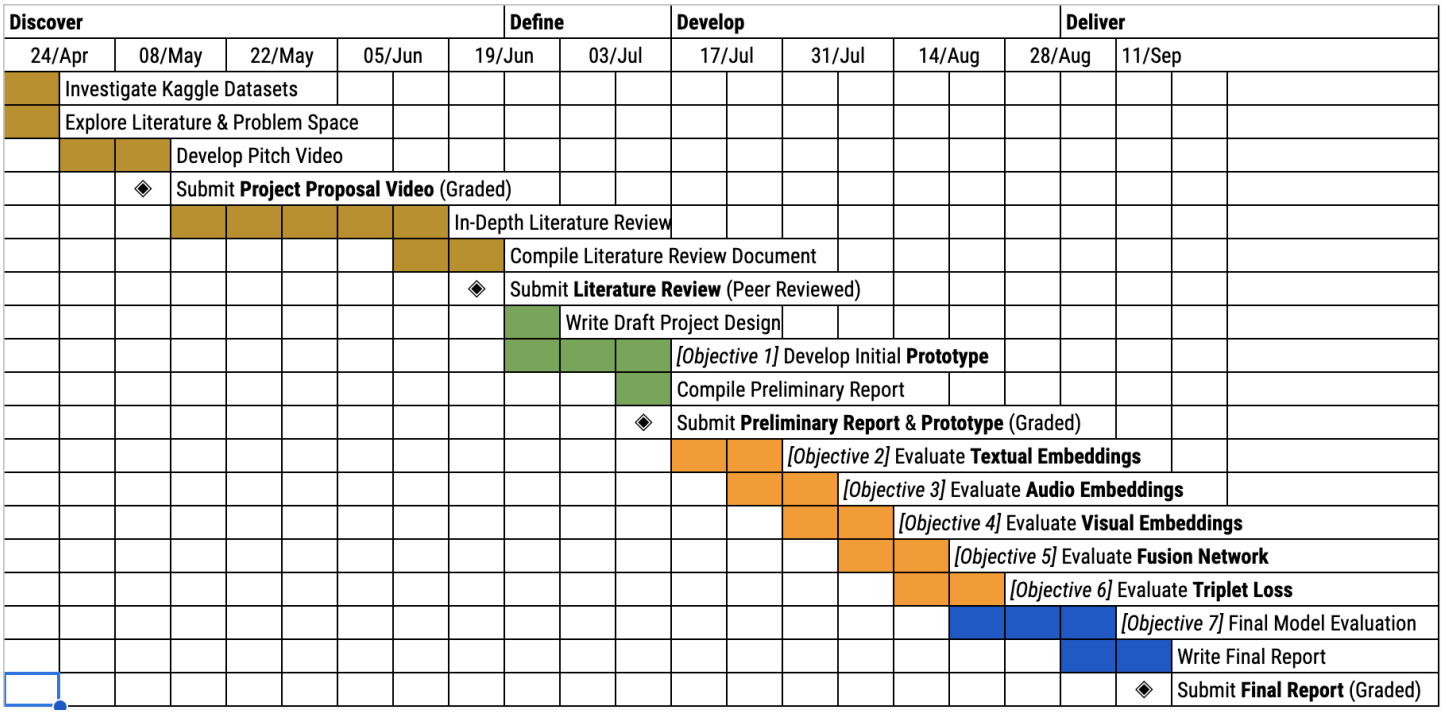

Figure 2: *Gantt Chart representation of the Project Plan, including Objectives, Milestones & Phases.*

IMPLEMENTATION

Classification based on embedded representations using Deep Learning can be boiled down to separation of multi-dimensional data points placed in a hyperplane. Reducing the dimensionality of the representations and plotting examples in 2D or 3D planes can be useful to verify, through data visualisation, the complexity of the separation that must be performed.

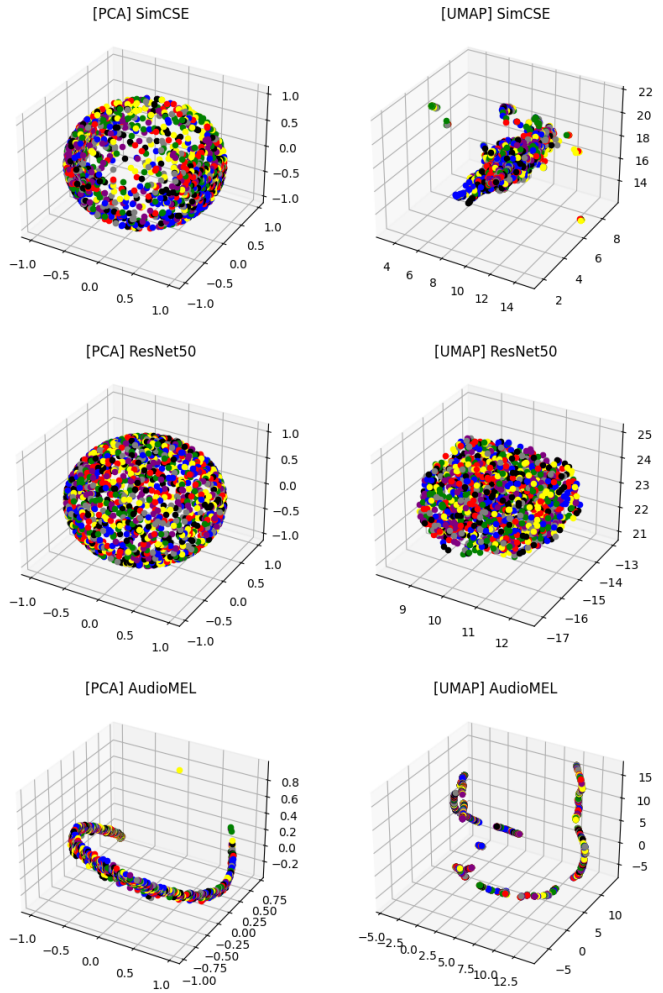

Figure 3: *MELD PCA/UMAP 3D Modality Visualisation. Source: dev/eda.ipynb*

In **Figure 3**, the SimCSE[Gao et al, 2021] was used to represent text, ResNet50[He et al, 2016] was used to represent visual features, representations and MEL features[McFee et al, 2023] were used to represent audio. These representations were then reduced to 3D using Principal Component Analysis (or "PCA") and Uniform Manifold Approximation and Projection (or "UMAP") and plotted having each class with a distinct colour. Such visualisation demonstrates how the data points offer no clear spatial separation, no clear clustering centroids, and evidence that neither linear nor polynomial models are likely to be sufficient to provide good enough separation for classification.

Prototype: Deep Learning Model Ensemble

To deal with the grand separation complexity observed, a Deep Learning Model Ensemble is introduced, with its prototype being a Baseline Model.

Extraction, Transformation & Loading

Mapped out as a Risk (a), during the Design phase, the MELD dataset in the form provided by Kaggle was hard and expensive to consume, with no separation between the audio and visual modalities.

```
from hlm12erc.etl import ETL, KaggleDataset

ETL(
    dataset=KaggleDataset(
        owner="zaber666",
        name="meld-dataset",
        subdir="MELD-RAW/MELD.Raw",
    )
).into(
    uri_or_folderpath=dir_data,
    force=False,
)
```

```
[7]  ✓ 3.2s

...  INFO:hlm12erc.etl:Kaggle dataset: zaber666/meld-dataset
     INFO:hlm12erc.etl:Workspace set to: /tmp/hlm12erc/etl
```

Figure 4: *Extract, Transform & Load. Source: dev/eda.ipynb*

An Extraction, Transformation & Loading (or "ETL") module (**Figure 4**) was created to transform the data into a simpler and more memory efficient format for consumption during training (**Figure 5**), with a single .csv file per split (train, valid, test) with one row per example, one image with 5 stacked equidistant video frames and one audio file per example.

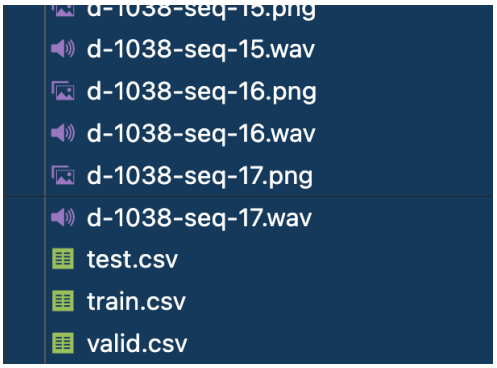

Figure 5: *Data, after ETL*

Objective 1 & "Statistical Power"

Due to class-imbalance discussed previously, a model reaching accuracy of 1/7 or 14.28% does not necessarily demonstrate "statistical power"[Chollet, 2017]. For that reason, the Prototype focused to produce a model:

(a) that could execute end-to-end, performing all the required transformations using the baseline building blocks to infer an emotion class as label, for any given MELD record; and

(b) capable of learning to distinguish through most of the 7 available emotion classes, rather than just infer the majority class (neutral).

```
# processing each modality independently, based on the presence
# of the respective encoder module, which can be set to None in the
# ERCConfig object.
y_text, y_visual, y_audio = None, None, None
if self.text_embeddings is not None:
    y_text = self.text_embeddings(x_text).to(self.device)
if self.visual_embeddings is not None:
    y_visual = self.visual_embeddings(x_visual).to(self.device)
if self.audio_embeddings is not None:
    y_audio = self.audio_embeddings(x_audio).to(self.device)

# fuse the embeddings from the different modalities
y_fusion, y_attn = self.fusion_network(y_text, y_visual, y_audio)

# transform the fused embeddings into the output logits
y_transformed = self.feedforward(y_fusion)
if self.config.feedforward_l2norm:
    y_transformed = l2_norm(y_transformed, p=2, dim=1)

# transform the transformed fused into logits and then into softmax probab
y_logits = self.logits(y_transformed)
y_pred = self.softmax(y_logits)

# if the true labels are provided, calculate the loss
loss = self.loss(y_pred, y_true) if y_true is not None else None

# return the output as a dictionary or tuple
output = ERCOutput(loss=loss, labels=y_pred, logits=y_logits, hidden_state
return output if return_dict else output.to_tuple()
```

Figure 6: *Forward pass of the $ERCModel$. Source: src/hlm12erc/modelling/erc_model.py*

Implemented using PyTorch[Paszke et al, 2019], the HLM12ERC model is expressed as a class named $ERCModel$ demonstrated through its forward pass in **Figure 6**, which is a direct reflection of the model architecture previously illustrated in the Design section of the present document and that delivered both of the specific goals set out for **Objective 1**.

Data Loading, Preprocessing & Training

The training loop was implemented with an extension of the Huggingface $Trainer$[2] class, which encapsulates the often complex logic required to orchestrate the forward pass and the backpropagation.

The flow illustrated in **Figure 7** demonstrates how the different components interact in producing the data required for training.

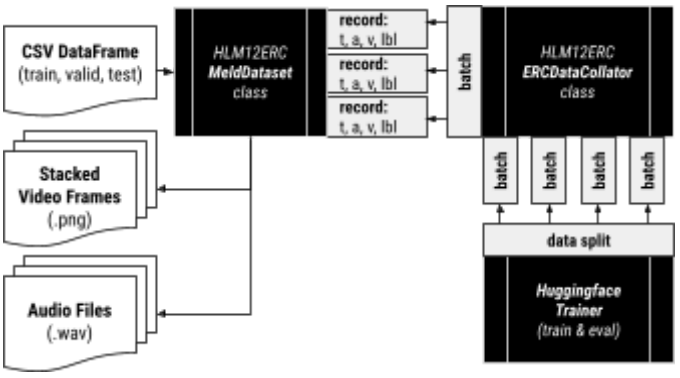

Figure 7: *Data Loading, Preparation & Training.*

Prototype Baseline Building Blocks

The prototype version of the $ERCModel$ was created and was composed of a simple set of building blocks, such as the encoders $ERCGloveTextEmbeddings$ (**Figure 8**), $ERCResNet50VisualEmbeddings$ (**Figure 9**) and the $ERCRawAudioEmbeddings$ (**Figure 10**), as well as the $ERCConcatFusion$ (**Figure 11**) and used the PyTorch built-in $CrossEntropyLoss$ as the objective function.

```
class ERCGloveTextEmbeddings(ERCTextEmbeddings):
    """
    ERCGloveTextEmbeddings is a class that implements the text embedding layer using t
    and then computes the mean of the embeddings for each token in the text.

    Example:
        >>> from hlm12erc.modelling import ERCConfig, ERCTextEmbeddingType
        >>> from hlm12erc.modelling.erc_emb_text import ERCGloveTextEmbeddings
        >>> config = ERCConfig()
        >>> ERCGloveTextEmbeddings.resolve_type_from(ERCTextEmbeddingType.GLOVE)(confi
    """

    hidden_size: int
    _device: torch.device | None

    def __init__(self, config: ERCConfig) -> None: ...

    def forward(self, x: List[str]) -> torch.Tensor:
        """
        Performs a forward pass through the text embedding layer.

        :param x: The input tensor of shape (batch_size,).
        :return: The output tensor of shape (batch_size, hidden_size).
        """
        seqs = [self.tokenizer(text) for text in x]
        tidss = [[self.token_to_idx.get(token, -1) for token in seq] for seq in seqs]
        ttnss = [[torch.tensor(tid) for tid in tids if tid >= 0] for tids in tidss]
        device = self.cache_or_get_same_device_as(self.embeddings)
        if device is not None:
            ttnss = [[ttn.to(device) for ttn in ttns] for ttns in ttnss]
        v = [[self.embeddings(ttn) for ttn in ttns] for ttns in ttnss]
        v = [[vii for vii in vi if torch.any(vii != 0)] for vi in v]
        y = pad_sequence([torch.stack(seq).squeeze() for seq in v], batch_first=True)
        y = torch.mean(y, dim=1)
        y = F.normalize(y, p=2, dim=1)
        return y
```

Figure 8: *$ERCGloveTextEmbeddings$ encoder. Source: src/hlm12erc/modelling/erc_emb_text.py#L50-L127*

[2] Huggingface Trainer class, used as training loop, source: https://huggingface.co/docs/transformers/main_classes/trainer

```python
class ERCResNet50VisualEmbeddings(ERCVisualEmbeddings):
    """
    ERCResNet50VisualEmbeddings is a class that implements the visual
    embedding layer using ResNet50 and simply returning the output of
    the final layer.

    References:
        >>> Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016.
        ... Deepresidual learning for image recognition. In Proceedings
        ... of the IEEE conference on computer vision and pattern
        ... recognition, 770-778."

    Example:
        >>> from hlm12erc.modelling import ERCConfig, ERCVisualEmbeddingType
        >>> from hlm12erc.modelling.erc_emb_visual import ERCVisualEmbeddings
        >>> config = ERCConfig()
        >>> ERCVisualEmbeddings.resolve_type_from(ERCVisualEmbeddingType.RESNET5
    """

    def __init__(self, config: ERCConfig):…

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """
        Performs a forward pass through the ResNet50 visual embedding layer.

        :param x: stacked vectors representing images
        :return: matrix of tensors (batch_size, out_features)
        """
        y = self.resnet50(x)
        y = l2_norm(y, p=2, dim=1)
        return y

    @property
    def out_features(self) -> int:…
```

Figure 9: *$ERCResNet50VisualEmbeddings$, Source: src/hlm12erc/modelling/erc_emb_visual.py#L45-L100*

```python
class ERCRawAudioEmbeddings(ERCAudioEmbeddings):
    """
    ERCRawAudioEmbeddings is a class that implements the
    Audio Feature Extraction model based on raw audio.

    Example:
        >>> from hlm12erc.modelling import ERCConfig, ERCAudioEmbeddingType
        >>> from hlm12erc.modelling.erc_emb_audio import ERCAudioEmbeddings
        >>> config = ERCConfig()
        >>> ERCAudioEmbeddings.resolve_type_from(ERCAudioEmbeddingType.WAVEFORM)
    """

    in_features: int

    def __init__(self, config: ERCConfig) -> None:
        """…
        super(ERCRawAudioEmbeddings, self).__init__(config=config)
        assert config is not None
        assert config.audio_in_features is not None
        assert config.audio_out_features is not None
        self.config = config
        self.in_features = config.audio_in_features
        self.ff = ERCFeedForward(
            in_features=config.audio_in_features,
            layers=[
                ERCConfigFeedForwardLayer(
                    out_features=config.audio_out_features * 3,
                    dropout=0.1,
                ),
                ERCConfigFeedForwardLayer(
                    out_features=config.audio_out_features * 2,
                    dropout=0.1,
                ),
                ERCConfigFeedForwardLayer(
                    out_features=config.audio_out_features,
                    dropout=0.1,
                ),
            ],
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """
        Create a representation based on the fixed size linear projection of the
        input tensor that represents the raw audio.

        :param x: stacked vectors representing audio waveforms
        :return: matrix of tensors (batch_size, out_features)
        """
        y = self.ff(x)
        y = l2_norm(y, p=2, dim=1)
        return y
```

Figure 10: *$ERCRawAudioEmbeddings$, Source: src/hlm12erc/modelling/erc_emb_audio.py#L60-L142*

```python
class ERCConcatFusion(ERCFusion):
    """
    Simple implementation of feature fusion based on concatenation of vectors.
    """

    hidden_size: int

    def __init__(…

    def forward(
        self,
        x_text: torch.Tensor | None,
        x_visual: torch.Tensor | None,
        x_audio: torch.Tensor | None,
    ) -> Tuple[torch.Tensor, torch.Tensor | None]:
        """
        Concatenates the input tensors along the feature dimension.

        :param x: List of tensors to be fused.
        :return: Tuple with fused tensor and attention weights.
        """
        x = [x_modal
             for x_modal in (x_text, x_visual, x_audio)
             if x_modal is not None]
        y = torch.cat(x, dim=1)
        y = l2_norm(y, p=2, dim=1)
        return y, None

    @property
    def out_features(self) -> int:…
```

Figure 11: *$ERCConcatFusion$ layer, Source: src/hlm12erc/modelling/erc_fusion.py#L72-L121*

Prototype Loss (Dice Coefficient)

It was initially conjectured that due to the severe class-imbalance of the MELD Dataset, the Cross Entropy Loss would result in no improvement to the target metric, the **F1 Weighted Score**, an expectation that was confirmed by the first experiment.

The initial prototype model (**Figure 12**) failed to learn other classes and kept on inferring only the majority class, still scoring 42%accuracy. Therefore, in pursuit of the objectives set out for the prototype, alternative objective functions had to be evaluated.

	precision	recall	f1-score	support
anger	0.00	0.00	0.00	153
disgust	0.00	0.00	0.00	22
fear	0.00	0.00	0.00	40
joy	0.00	0.00	0.00	163
neutral	0.42	1.00	0.60	470
sadness	0.00	0.00	0.00	111
surprise	0.00	0.00	0.00	150
accuracy			0.42	1109
macro avg	0.06	0.14	0.09	1109
weighted avg	0.18	0.42	0.25	1109

Figure 12: *42% accuracy, however 6 classes at 0% recall. Source: dev/mlops.ipynb*

The $DiceCoefficientLoss$ (**Figure 13**) was created as an attempt to ensure that the model was optimising for the correct metric (Weighted F1-Score), and it has indeed demonstrated greater ability to (a) learn classes other

than the majority class and (b) converge more stably during training than the alternatives (**Figure 14**).

```python
class DiceCoefficientLoss(ERCLoss):
    """
    Dice Coefficient Loss function for ERC models.

    Reference:
    >>> Peiqing Lv, Jinke Wang, Xiangyang Zhang, Chunlei Ji,
    ... Lubiao Zhou, and Haiying Wang. 2022. An improved residual
    ... U-Net with morphological-based loss function for automatic
    ... liver segmentation in computed tomography. Math. Biosci.
    ... Eng. 19, 2 (January 2022), 1426–1447.
    """

    def __init__(self, config: ERCConfig):...

    def forward(
        self,
        y_pred: torch.Tensor,
        y_true: Optional[torch.Tensor] = None,
    ) -> torch.Tensor:
        """
        Calculate and return the loss given the predicted and true labels
        using the Dice Loss function, which is defined as mean of the Dice
        coefficient across all classes, and derived from the equation:
        >>> 1 - (2 * TP) / (2 * TP + FP + FN)

        :param y_pred: Predicted labels, already converted to a batch of softmax
        :param y_true: True labels
        :return: Loss value
        """
        # Compute TP, FP, and FN for each class
        assert y_true is not None
        TP = (y_pred * y_true).sum(dim=0)
        FP = (y_pred * (1 - y_true)).sum(dim=0)
        FN = ((1 - y_pred) * y_true).sum(dim=0)

        # Compute Dice coefficient for each class
        dice_class = (2 * TP) / (2 * TP + FP + FN + self.epsilon)

        # Average Dice coefficient across all classes and compute the loss
        return 1 - dice_class.mean()
```

Figure 13: *`DiceCoefficientLoss` algorithm. Source: [src/hlm12erc/modelling/erc_loss.py#L80-L121]*

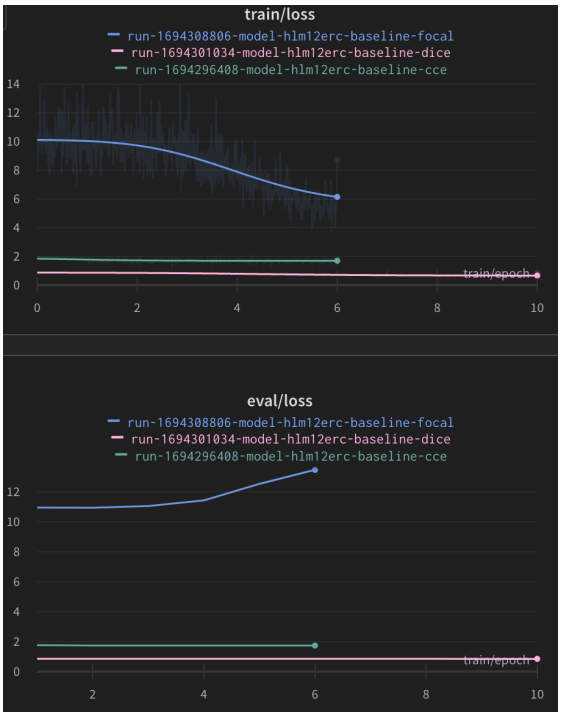

Figure 14: *CrossEntropy (CCE), Dice and Focal Loss Curves, Source:* ***Weights & Biases (hlm12erc_v3)****.*

Beyond the Prototype: Iterative Experimentation

Past the prototype, the implementation focused on executing on each one of the objectives subsequently, but prioritised addressing the risks and challenges first.

The Final Model Architecture (**Figure 16**), that delivered 57% Weighed F1-Score on the Test Split, against 27% from the baseline ("prototype"), was the result of large scale experimentation.

Mapped as Risk (b), the Computational Expense of the Model and the time consumed by the training process was considerable, was addressed first and required substantial efforts, discussed in further detail below.

The amount of experiments proposed and run by the present project was also significant. Managing experiment settings for each one of them required unforeseen sophistication which was addressed with an elegant modular solution, more closely looked into in this section.

Hardware Acceleration (TPU/GPU)

Hardware acceleration has been an essential ally of Deep Learning, without which most of the modern advancements made in the field would have arguably been impossible.

```python
# import torch_xla.core.xla_model as xm
# device = xm.xla_device()

# The following code sets the `device` to one of the GPUs
import torch
device = torch.device("cpu")
if torch.cuda.is_available():
  device = torch.device("cuda:0")
device
```

```
device(type='cuda', index=0)
```

Figure 15: *Loading CUDA GPUs for Training on Google Colab. Source: [dev/mlops.ipynb]*

Model	Hardware	Epochs	Training Time
baseline-cce	CPU	5	~27h00 min
baseline-cce	GPU/A100	10	~01h15 min

Table 2: Training "baseline-cce" on GPU/A100. *Source: [dev/mlops.ipynb]*

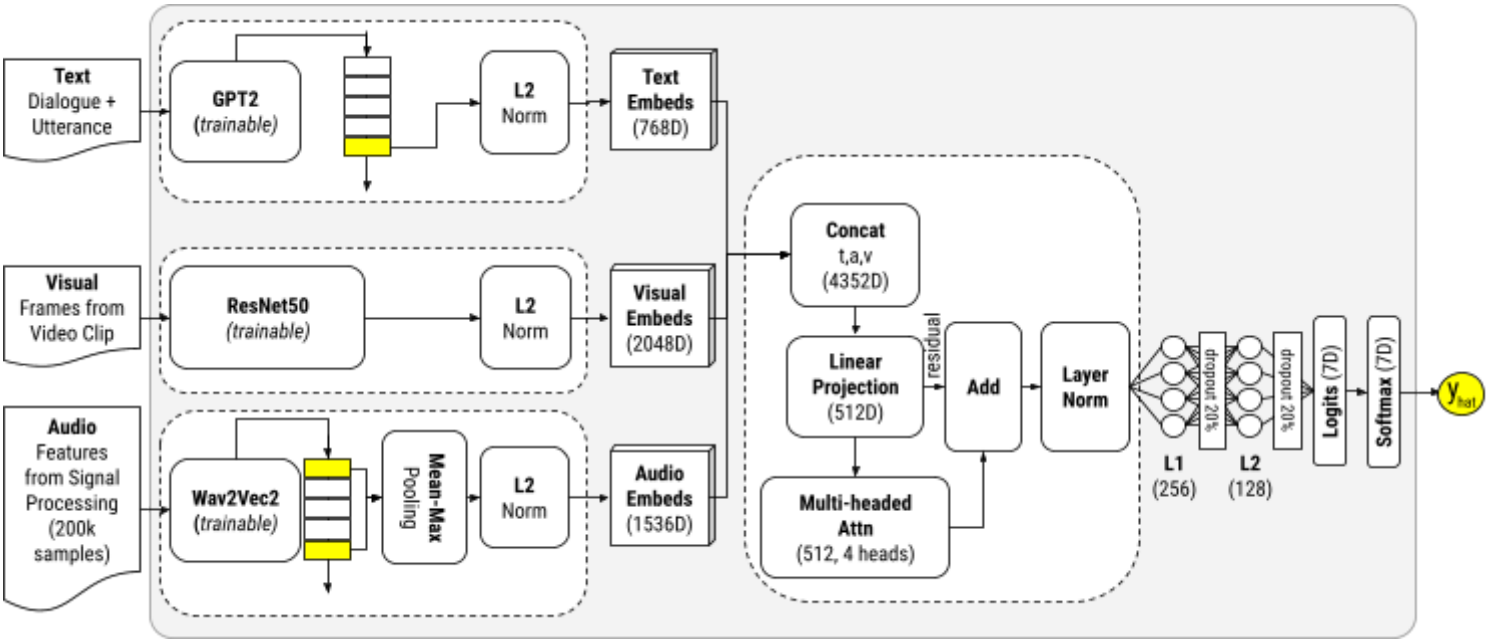

Figure 16: *Final Architecture for the HLM12ERC model*

Tensor Processing Units (or "TPU") designed by Google to run accelerated model training on Google Colab, and they were attempted first. Unfortunately, TPUs could not be used due to an issue found in the XLA library for which a fix[Mendes/XLA, 2023] was out of the scope of the present project.

GPUs, accessible through PyTorch and the CUDA Toolkit[3], was the alternative hardware acceleration option in Google Colab which provides, upon their own availability, NVIDIA A100 GPUs that can be used for model training. **Figure 15** demonstrates the CUDA device being loaded into the Google Colab Environment, and **Table 2** shows the approximate gains obtained in training time by engineering hardware acceleration for training.

Model Architecture from Configuration

$ERCModel$ was designed and implemented to have its internal modules fully chosen based on a .yml configuration files loaded as the data class $ERCConfig$, including the choice of modules to be loaded as Text Encoder, Visual Encoder and Audio Encoder, as well as the Fusion Layer, as demonstrated by **Figure 17**.

Having the model architecture fully loaded from configuration files allowed experiments to be easily version controlled and contributed for a much more stable iterative development. All settings tested experimentally can be found in the `./configs` folder, and can be used for training through the simple combination of two classes: $ERCConfigLoader$ and $ERCTrainer$, as demonstrated by **Figure 18**.

```yaml
classifier_name: t-um-gpt2
classifier_loss_fn: dice
classifier_learning_rate: 0.00005
classifier_weight_decay: 0.1
classifier_warmup_steps: 1000 # @ ~15% => 9989
classifier_metric_for_best_model: f1_weighted
classifier_early_stopping_patience: 5
classifier_classes:
  - anger
  - disgust
  - fear
  - joy
  - neutral
  - sadness
  - surprise

modules_text_encoder: gpt2
modules_visual_encoder: none
modules_audio_encoder: none
modules_fusion: concat

text_in_features: -1 # defined by tokenizer & p
```

Figure 17: *Model configuration ("t-um-gpt2.yml"). Source: configs/t-um-gpt2.yml*

```python
def train_model(
    config_name: str,
    datasets: Tuple[MeldDataset, MeldDataset],
    n_epochs: int,
    batch_size: int,
) -> None:

  # reset training state
  reset_training_state()

  # loading configuration and model architecture
  model_config = ERCConfigLoader(dir_configs / f"{config_name}.yml").load()
  model_trainer = ERCTrainer(model_config)
```

Figure 18: *Loading a model from a config file. Source: dev/mlops.ipynb*

[3] CUDA Toolkit: https://developer.nvidia.com/cuda-toolkit

Artefact & Experiment Tracking

Running training locally generated assets (in particular, trained models in the `.pt` format) that can be easily retrieved. The same is not true to models trained in Google Colab which are deleted after the Runtime expires.

Weights & Biases (Free version) was chosen for its seamless integration with the Huggingface Trainer, that required no more than (a) a dependency and (b) running a login step to start allowing both monitoring and artefact management.

The visualisation for experiments provided by Weights and Biases (**Figure 19**) were instrumental to help distinguish better performing models from the others, monitor overfitting, and recover models after training that have been lost due to Colab Runtime expiration.

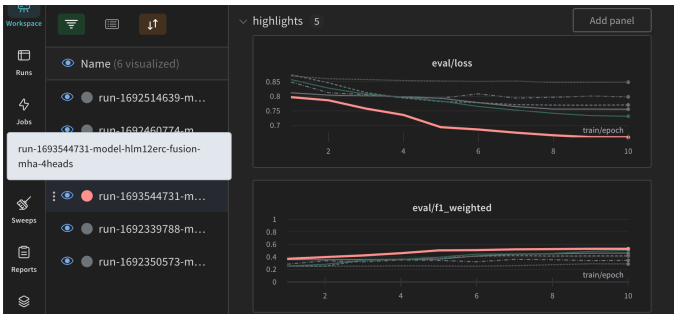

Figure 19: *Model Training Monitoring. Source:* ***Weights & Biases (hlm12erc_v3)****.*

Unit & Integration Testing

Each one of the building blocks used to compose the model architecture, as well the components used by the ETL process, have received ample test coverage.

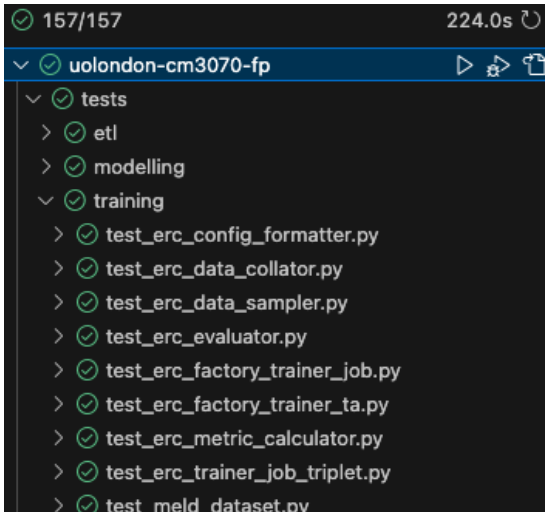

Figure 20*: 157 Automated Tests. Source: tests*

Test-driven Development (TDD) approach was not applied during the development of the project. However both approaches to automated testing known as Unit Testing and Integration Testing were employed during the software development lifecycle.

157 automated tests (**Figure 20**) were instrumental in preventing, on several occasions, errors which would have otherwise compromised the results of experiments.

Objective 2 & GPT2 Embeddings

The GPT2 model is pre-trained in an auto-regressive fashion, learning relationships between tokens in a corpus in order to predict the next token[Radford, 2019]. Its representation of the last token of a sequence carries the information about the entire preceding context. Accordingly, the $ERCGpt2TextEmbeddings$ (**Figure 21**) runs the dialog text through GPT2 and collects the last token representation.

```python
class ERCGpt2TextEmbeddings(ERCTextEmbeddings):
    """
    ERCGpt2TextEmbeddings is a class that implements the text embedding layer
    using the last hidden state produced by the GPT-2 model prior to the
    softmax layer, as a form of embedding.
    """

    gpt2_tokenizer: transformers.GPT2Tokenizer
    gpt2_model: transformers.GPT2Model
    tokenizer_opts: dict
    maxlength: int

    def __init__(self, config: ERCConfig) -> None:…

    def forward(self, x: List[str]) -> torch.Tensor:
        """
        Performs a forward pass through the text embedding layer.

        :param x: The input tensor of shape (batch_size,).
        :return: The output tensor of shape (batch_size, hidden_size).
        """
        # pick the device
        device = self.cache_or_get_same_device_as(self)
        # encode the input
        y = self.gpt2_tokenizer.batch_encode_plus(x, **self.tokenizer_opts)
        y["input_ids"] = y["input_ids"].to(device)
        attention_mask = y["attention_mask"] = y["attention_mask"].to(device)

        # fix shape, if greater than max_length
        if y["input_ids"].size(dim=1) > self.maxlength:
            start = self.maxlength
            y["input_ids"] = y["input_ids"][:, -start:]
            attention_mask = y["attention_mask"] = y["attention_mask"][:, -start:]

        # extract the representation from the last token
        y = self.gpt2_model(**y).last_hidden_state
        last_token_pos = attention_mask.sum(dim=1) - 1
        y = y[torch.arange(y.size(0), device=device), last_token_pos]

        # normalize the representation
        y = l2_norm(y, p=2, dim=1)
        return y

    @property
    def out_features(self) -> int:…
```

Figure 21: *$ERCGpt2TextEmbeddings$ code. Source: src/hlm12erc/modelling/erc_emb_text.py#L137-L205*

The GPT2 text encoder resulted in significant improvement on the target metric (**Table 4**). However, it

also resulted in increased RAM consumption, requiring batch sizes to be adjusted, reduced from 64 to 16, and taking longer to train the model by approximately 1 hour, on NVidia A100 GPUs, that provides 40GB of RAM.

Objective 3 & Wav2Vec2 Embeddings

Similar to GPT2, the Wav2Vec2 model[Baevski et al, 2020] produces a representation of each audio segment.

To produce a single fixed-size vector representation of the entire audio, the $ERCWave2Vec2TextEmbeddings$ (**Figure 22)**, takes mean and max pooling of the segments encoded by the Wave2Vec2 model and combines them into a single representation that has twice the dimensionality of each segment representation (2×768), following an approach known as Mean-Max Attention Autoencoder[Zhang et al, 2018].

```python
class ERCWave2Vec2Embeddings(ERCAudioEmbeddings):
    """
    ERCWave2Vec2Embeddings is a class that implements the
    Audio Feature Extraction model using the pretrained Wave2Vec2 model,
    by concatenating the mean and max pooling of the hidden states.

    References:
        >>> Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli.
        ... 2020. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech
        ... Representations. In Advances in Neural Information Processing Systems,
        ... Curran Associates, Inc., 12449–12460. Retrieved
        ... from https://proceedings.neurips.cc/paper_files/paper/2020/file/92d1e1eb1cd6

        >>> Minghua Zhang, Yunfang Wu, Weikang Li, and Wei Li. 2018. Learning
        ... Universal Sentence Representations with Mean-Max Attention Autoencoder.
        ... In Proceedings of the 2018 Conference on Empirical Methods in Natural
        ... Language Processing, 4514–4523.

    Example:
        >>> from hlm12erc.modelling import ERCConfig, ERCAudioEmbeddingType
        >>> from hlm12erc.modelling.erc_emb_audio import ERCAudioEmbeddings
        >>> config = ERCConfig()
        >>> ERCAudioEmbeddings.resolve_type_from(ERCAudioEmbeddingType.WAV2VEC2)(config)
    """

    in_features: int
    hidden_size: int
    wav2vec2: transformers.Wav2Vec2Model
    fc: torch.nn.Linear | None

    def __init__(self, config: ERCConfig) -> None: ⋯

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """
        Create a representation based on the last hidden state of the pretrained
        Wave2Vec2 model concatenating the mean and max pooling of the hidden states.

        :param x: stacked vectors representing audio waveforms
        :return: matrix of tensors (batch_size, out_features)
        """

        # concatenate the mean and max pooling of the hidden states
        # to generate a fixed size vector that can be used as input
        # to the classifier
        h = self.wav2vec2(x).last_hidden_state
        h_mean = torch.mean(h, dim=1)
        h_max = torch.max(h, dim=1)[0]
        y = torch.cat((h_mean, h_max), dim=1)

        # only projects if the output size is different from the hidden size
        if self.fc is not None and self.hidden_size != y.size(dim=1):
            y = self.fc(y)

        # normalize the output vector to have unit norm
        return l2_norm(y, p=2, dim=1)

    @property
    def out_features(self) -> int: ⋯
```

Figure 22: *$ERCWave2Vec2Embeddings$ code. Source: src/hlm12erc/modelling/erc_emb_audio.py#L145-L235*

Wav2Vec2 representations *underperformed* when compared to the baseline encoder, a 3-layer fully connected network, both for audio-only modality and when features fused through concatenation (**Table 4**).

However, the richer representations have been leveraged more efficiently by the Attention-based Feature Fusion approach (**Table 4**), with resource consumption similar to the GPT2-based Text Encoder.

Objective 4 & RetinaFace Pre-processing

The hypothesis tested by Objective 4 was that faces carried better signal-to-noise ratio than entire scenes, with respect to the emotions of the speakers.

```python
class MeldVisualPreprocessorFilepathToFaceOnlyImage(MeldVisualPreprocessor):
    """
    Preprocessor calss for the visual (image stack) files, turning them into
    tensors, for better compatibility with TPU training without affecting
    negatively CPU-based training.
    """

    def __init__(
        self,
        filepath_retinaface_resnet50: pathlib.Path,
        device: torch.device | None = None,
    ) -> None: ⋯

    def __call__(self, image: Image | pathlib.Path) -> Image | torch.Tensor:
        """
        Preprocesses the image by applying a series of transformations to it.

        :param x: The image to be preprocessed
        :return: The preprocessed image
        """
        # ensure that we have a filepath to start with
        if isinstance(image, torch.Tensor): ⋯
        elif isinstance(image, Image): ⋯

        # detect faces in image
        faces = self.face_detector(image)

        # create the image we will manipulate
        with open_image(image) as image_instance:
            # eliminate faces that larger than 1/5 of the image
            # because it would be a face spanning over 2 frames
            # which is impossible
            faces = [face
                     for face in faces
                     if (face["y2"] - face["y1"]) < image_instance.height / 5]

            # black out pixels not within faces
            image_blackedout = self._black_out_non_face_pixels(
                image=image_instance,
                faces=faces,
            )

            # returns the image instance, that can be used by the next preprocessor
            return image_blackedout

    def _black_out_non_face_pixels(self, image: Image, faces: list) -> Image:
        """
        Uses numpy to efficiently black out pixels that are not within the bounding

        :param image: The image to be manipulated
        :param faces: The list of faces detected in the image
        :return: The image with the non-face pixels blacked out
        """
        img_array = np.array(image)
        mask = np.zeros((image.height, image.width), dtype=bool)
        for face in faces:
            x1, y1, x2, y2 = face["x1"], face["y1"], face["x2"], face["y2"]
            mask[y1:y2, x1:x2] = True

        img_array[~mask] = [0, 0, 0]
        return image_from_array(img_array)
```

Figure 23: *Preprocessing class used by $MeldDataset$, $MeldVisualPreprocessorFilepathToFaceOnlyImage$ (...)training/meld_record_preprocessor_visual.py#L85-161*

The lack of a pretrained model that could be as easily incorporated into the ensemble as the gpt2 or wav2vec2 models led to the implementation of RetinaFace[Yanjia et al, 2021] as a preprocessing step (**Figure 23**), which also required sophistication of the $MeldDataset$ class to produce visual data limited only to the faces detected in the original image.

Contrary to the hypothesis, results show that feeding only the faces detected to the encoder does in fact perform worse than feeding the entire scene, in both unimodal and multimodal settings. Furthermore, this approach is significantly more computationally expensive than feeding the entire scene, since it runs the RetinaFace detector prior to the model's forward pass, hence requiring a smaller batch (8) and longer training time (over 6 hours).

Objective 5 & Attention-based Feature Fusion

Having E be a collection of embeddings output by the encoders of each modality (e.g.: E_{text} being the embedding of the text encoder), the baseline feature fusion simply concatenates all E into a single long feature vector E_{combo} with $D = |E_{combo}| = \sum_{i=1}^{|E|} |E_i|$, which is then fed into a feed forward network responsible for transforming the vector representation from E_{combo} into the $logits$ that are then normalised into a $Softmax$ distribution of probabilities.

Conversely, the $ERCMultiheadedAttentionFusion$ module (**Figure 24**) makes use of the PyTorch built-in $MultiheadAttention$ layer that trains D^2 parameters (significantly larger) to learn relationships between the features in the feature vector.

The largest model architecture, based on the most advanced encoders for each modality, described, at the feature fusion step, the following dimensionality:

$$\begin{aligned} & D \\ &= |E_{text}| + |E_{audio}| + |E_{visual}| \\ &= 768 + 1536 + 2048 \\ &= 4352 \end{aligned}$$

Unfortunately, if the inputs Q, K and V of the $MultiheadAttention$ have $D = 4352$, then the $MultiheadAttention$ will have $4,352^2 = 18,939,904$ parameters, which are too large to be trained, even with a small batch size, over the most powerful GPU (A100).

```python
class ERCMultiheadedAttentionFusion(ERCFusion):
    """…

    def __init__(…

    def forward(
        self,
        x_text: torch.Tensor | None,
        x_visual: torch.Tensor | None,
        x_audio: torch.Tensor | None,
    ) -> (torch.Tensor, torch.Tensor | None):
        """
        Performs Multiheaded Attention on the input tensors,
        through concatenating the input, transforming the inputs
        using a multi-headed attention layer, and then performing
        the element-wise addition of the input and the attention
        output (residual connection).

        Each input X (for each modality) is projected over a linear
        representation of output dimensionality proportional to the
        input dimensionality.

        These projected representations are then concatenated and fed
        into the attention mechanism, which outputs a tensor of the
        same dimensionality of the input.

        Finally, the output of the attention mechanism is added to
        the input (residual connection), and layer normalisation is
        applied.

        :param x_text: Tensor with the text embeddings, with dimensions (ba
        :param x_visual: Tensor with the visual embeddings, with dimensions
        :param x_audio: Tensor with the audio embeddings, with dimensions (
        :return: Tuple with fused tensor and attention weights.
        """
        # dimensionality reduction through mapping using a linear layer
        x_all = []
        if self.fc_text:
            x_all.append(self.fc_text(x_text))
        if self.fc_visual:
            x_all.append(self.fc_visual(x_visual))
        if self.fc_audio:
            x_all.append(self.fc_audio(x_audio))
        # concatenate the mapped embeddings
        x = torch.cat(x_all, dim=1)
        # perform the attention mechanism
        attn, _ = self.attn.forward(query=x, key=x, value=x)
        # residual connection
        y = x + attn
        # layer normalisation
        y = self.layer_norm(y)
        # output
        return y, attn

    @property
    def out_features(self) -> int:…
```

Figure 24: *$ERCMultiheadedAttentionFusion$. Source: src/hlm12erc/modelling/erc_fusion.py#L124-L262*

To solve this issue, a $Linear$ (layer) was introduced for each modality, with dimensionality proportional to the input embeddings, with their joint dimensionality adding up to a hyperparameter set in the $ERCConfig$ class called $fusion\_out\_features$. After the embeddings are linear projected into a lower dimension, they are then fed into the $MultiheadAttention$ with much less parameters to train ($512^2 = 262,144$, during experimentation) and becomes viable.

Feature fusion implemented with $MultiheadAttention$, in conjunction with other advanced Feature Extraction modules was responsible for the greatest Weighted F1-Score achieved during experimentation at 53%, and ability to distinguish 5 out of the 7 emotion classes from the multimodal data reasonably well (**Table 4**).

Objective 6 & Triplet Contrastive Learning

Both the SPCL-CL-ERC model[Song et al, 2022] (as well as its underlying text encoder, SimCSE[Gao et al, 2021]) and the M2FNet model[Chudasama et al, 2022] use different forms of contrastive learning, which aims to create a embedding space with better structure and greater separation margin between representations of different classes, thus simplifying downstream classification.

The hypothesis tested by Objective 6 was that if an additional loss, computed based on triplets (anchor, positive & negative) was summed to the classification loss, it would force the final representation produced before logits to be more easily separable, which in turn would lead to an improved classification performance.

```python
def __iter__(self):
    """
    Iterates through the classes and yields 2 indices per class, until the
    balanced size is reached.
    """
    iter_cursor = 0
    class_cursor = {k: 0 for k in self.indices_per_class.keys()}
    while iter_cursor < self.balanced_size:
        # every pass of all classes, randomise the order in which
        # the classes appear together, to allow triplet contrastive
        # learning to happen accross different class pairs, rather
        # than fixed ones.
        indices_of_shuffled_classes = self._randomise_class_indices_pairs()
        for class_, class_indices in indices_of_shuffled_classes:
            # yields as many items as required per class
            # which is optimised based on the batch size and
            # number of classes available in the datset
            for _ in range(self.n_examples_per_class):
                try:
                    yield class_indices[class_cursor[class_] % len(class_indices)]
                finally:
                    class_cursor[class_] += 1
                    iter_cursor += 1
                if iter_cursor >= self.balanced_size:
                    return

def _randomise_class_indices_pairs(self) -> List[Tuple[str, List[int]]]:
    """
    Changes the order in which the classes appear together, to allow triplet
    contrastive learning to happen accross different class pairs, rather than
    fixed ones.

    :return: List of indices per class.
    """
    pairs = list(self.indices_per_class.items())
    random.shuffle(pairs)
    return pairs
```

Figure 25: *$ERCDataSampler$, custom batches. Source: src/hlm12erc/training/erc_data_sampler.py#L16-L103*

Building triplets requires a special batching strategy, implemented in the form of a Sampler (**Figure 25**), attached the the DataLoader created by yet another extended version of the Huggingface Trainer, responsible for the customised loss calculation (**Figure 26**).

```python
def compute_loss(
    self,
    model,
    inputs,
    return_outputs=False,
) -> Union[torch.Tensor, Tuple[torch.Tensor, Any]]:
    """
    Computes and sums the classification loss and the triplet loss

    :param model: Model to compute the loss for.
    :param inputs: Inputs to the model.
    :param return_outputs: Whether to return the outputs of the model.
    :return: Loss of the model.
    """
    # get hold of labels, required to calculate metrics & triplet loss
    labels = inputs.get(ERCDataCollator.LABEL_NAME)

    # compute normal classifier metrics
    outputs = model(**inputs)
    total_loss = classifier_loss = outputs.loss
    classifier_metrics = dict()

    if labels is not None:
        classifier_metrics = self._compute_metrics(
            labels,
            outputs,
            loss=classifier_loss.item(),
        )

        # if we are training using the triplet loss, we must compute the
        if self.is_in_train:
            triplet_loss = self._compute_triplet_loss(
                outputs=outputs,
                labels=labels,
                loss_fn=self.triplet_loss,
            )
            total_loss += triplet_loss
            classifier_metrics.update(
                dict(
                    triplet_loss=triplet_loss.item(),
                    total_loss=total_loss.item(),
                )
            )

    if classifier_metrics:
        self.log(classifier_metrics)

    return (total_loss, outputs) if return_outputs else total_loss
```

Figure 26: *$ERCTrainerTripletJob$ extending the Huggingface Trainer to compute the combo loss based on TripletLoss and ClassifierLoss (Dice). Source: src/hlm12erc/training/erc_trainer_job_triplet.py*

The $ERCTripletLoss$ (**Figure 27**) calculated the loss through an equation inspired by the SimCSE[Gao et al, 2021] paper, expressed as:

$$-log\left(\frac{\frac{\sum_{i=0}^{|p|} sim(a, p_i)}{|p|}}{\frac{\sum_{i=0}^{|p|} sim(a, p_i)}{|p|} + \frac{\sum_{i=0}^{|n|} sim(a, n_i)}{|n|}}\right)$$

While the results of the experimentation with a Triplet Loss function were inconclusive, some interesting conclusions arose from the exercise.

```python
class ERCTripletLoss(torch.nn.Module):
    """…

    def __init__(self, config: ERCConfig, *args, **kwargs):…

    def forward(
        self,
        anchor: torch.Tensor,
        positives: torch.Tensor,
        negatives: torch.Tensor,
    ) -> torch.Tensor:
        """
        Calculate and return the loss given the predicted and true labels using
        a losse equation inspired on the Triplet Loss function devised by the
        SimCSE Paper, that can be described by the following equation.
        >>> loss = -log(
        ...     sum(sim(anchor, positives)) /
        ...     sum(cat(sim(anchor, positives), sim(anchor, negatives))
        ... )

        Reference:
        >>> Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple
        ... Contrastive Learning of Sentence Embeddings. In EMNLP 2021 - 2021
        ... Conference on Empirical Methods in Natural Language Processing,
        ... Proceedings (EMNLP 2021 - 2021 Conference on Empirical Methods in
        ... Natural Language Processing, Proceedings), Association for
        ... Computational Linguistics (ACL), 6894-6910.

        :param anchor: Anchor embeddings, used as reference for the positive and
        :param positive: Positive embeddings, used as a positive example for the
        :param negative: Negative embeddings, used as a negative example for the
        :return: Loss value
        """
        p = self._sim(anchor, positives)
        n = self._sim(anchor, negatives)
        weighted_p = p / positives.shape[0]
        weighted_n = n / negatives.shape[0]
        weighted_all = torch.cat((weighted_p, weighted_n))
        ratio = torch.sum(weighted_p) / (torch.sum(weighted_all) + self.epsilon)
        loss = -torch.log(ratio)
        return loss

    def _sim(self, anchor: torch.Tensor, other: torch.Tensor) -> torch.Tensor:
        """
        Produces a normalised cosine similarity ranging [0, 1] between the
        anchor and the other tensor/matrix.

        :param anchor: Anchor tensor
        :param other: Other tensor
        :return: Normalised cosine similarity
        """
        return (1 + torch.cosine_similarity(anchor, other)) / 2
```

Figure 27: *ERCTripletLoss forward code. Source: src/hlm12erc/modelling/erc_loss.py#L174-L267*

The model's evaluation Weighted F1-Score curve (**Figure 28**) trajectory was significantly low, achieving 26% at its highest. However, the Evaluation Weighted F1-Score has apparently entered on a low yet increasing trajectory before it was stopped by the Early Stopping callback, for having reached its limit patience (set to 5). Increasing the patience or disabling Early Stopping could potentially lead to different results.

Additionally (**Figure 29**), even though the training was interrupted early, the model evaluation metrics looked promising in terms of distinguishing emotion classes.

Further experimentation and longer training in terms of epochs are required before the full potential of triplet contrastive learning can be realised. However, the custom batching strategy required for the Triplet Loss functions to work leads to considerably oversampling, training takes multiple times longer than the alternative models, maxing out the resources of the most powerful GPU almost entirely.

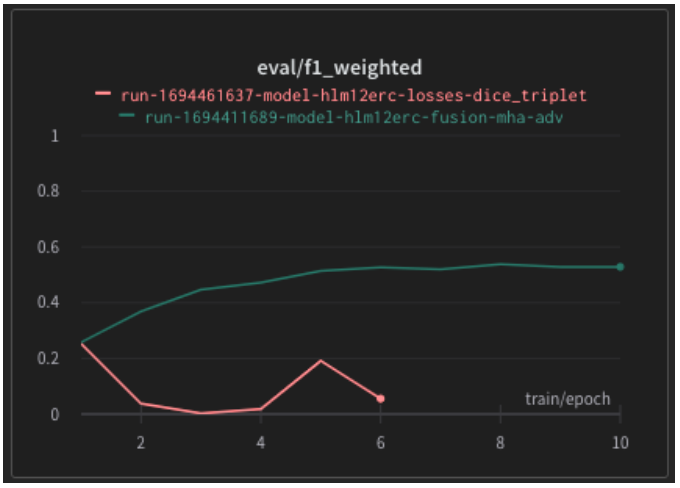

Figure 28: Triplet Loss Early Stopped at epoch 6 due to low dip in Weighted F1-Score on the Validation Split. Source: ***Weights & Biases (hlm12erc_v3)***

The consequential increase in costs related to hardware acceleration rendered further experimentation of this topic as not viable for the scope of this project.

	precision	recall	f1-score	support
anger	0.17	0.20	0.18	153
disgust	0.00	0.00	0.00	22
fear	0.00	0.00	0.00	40
joy	0.13	0.02	0.04	163
neutral	0.42	0.56	0.48	470
sadness	0.06	0.02	0.03	111
surprise	0.12	0.17	0.14	150

Figure 29: *Triplet Loss-based model, Valid Split. Source: dev/mlops.ipynb*

EVALUATION

Reflection on Aims and Goals

Reflecting on the original aim of "performing ERC on the MELD dataset by designing, training and evaluating a model inspired by the existing literature and assembled in a novel way", I conclude that this project can be considered a success and major achievement, but with some important caveats.

Objective 1 set out the creation of a Prototype/Baseline Model inspired by M2FNet[Chudasama et al, 2022], and succeeded in doing so by deploying a model that could effectively learn multimodal representations each with a separate Feature Extraction module for each modality, fusing these features using a Feature Fusion module and then used these feature fusions to train a classifier capable of outputting an emotion label. The Baseline delivered not just a working architecture, but also successfully handled the challenge of learning to distinguish some ambiguous emotions from a severely class-imbalanced dataset by replacing the Cross Entropy loss, traditionally used to train classifiers, by the Dice Coefficient Loss which empirically proven to be better than the former ($+ 2\%$, **Table 4**, Exp 1 and 2) in optimising towards an improved Weighted F1-Score.

Objective 2 had the goal of experimenting and choosing the best performing Textual Feature Extractor by comparing the existing baseline, based on mean-polled GloVe vectors of the dialog text, and an alternative powered by a pre-trained Transformers-based model, succeeding in this objective beyond original expectations. GPT-2 provided features performing remarkably better than the former ($+ 21\%$, **Table 4**, Exp 4, 5). Additionally, the present project introduced a novel approach to modelling the dialogue, not seen in previous work, including state-of-the-art models in the field, utilising prompt-engineering similar to SPCL-CL-ERC[Song et al, 2022], but feeding the entire dialogue into the GPT-2 context window that is limited to 1024 tokens but sufficient for fitting all dialogues in the training dataset.

Objective 3 explored alternative pre-trained models to create Audio Representations and successfully devised an alternative Audio Feature Extractor based on Wav2Vec2[Baevski et al, 2020] that later was found to perform significantly better than the baseline 3-layered fully connected network alternative ($+ 9\%$, **Table 4**, Exp 17, 21, 24), as long as the features were fused using a Multi-headed attention layer (**Table 4**, Exp 9, 15).

Objective 4 was designed to conclude whether feeding only faces rather than the entire scene to the Visual Feature Extractor would be beneficial and concluded that the approach was considerably more computationally expensive, roughly doubling the training (6 hours) when compared to the final model average training time (4 hours), and did not result in better performance.

Objective 5 aimed to investigate whether Multi-headed Attention (or "MHA") could improve feature fusion, which it successfully achieved by showing that MHA performed $+ 3\%$ better when fusing features created by baseline Feature Extractors (**Table 4**, Exp 2, 13) and $+ 17\%$ better when fusing more sophisticated features output by more advanced Feature Extractors, namely GPT2 and Wav2Vec2 (**Table 4**, Exp 14, 15).

Whereas Objective 6 did not achieve its principal goal of providing a clear picture to whether a Triplet Loss function and Curriculum Learning combined with the classification loss could be superior to the baseline Dice Coefficient Loss function, it did deliver a valid implementation of a Triplet Loss function inspired by SimCSE[Gao et al, 2021], delivered a reasonable data sampling strategy to build better batches that shares some of the features of the Curriculum Learning used by SPCL-CL-ERC[Song et al, 2022] and that could be furthered to better support different forms of Contrastive Learning. Further experimentation on the topic had been determined as out-of-scope for the present project, but are discussed later in the document as opportunities for future work.

Finally, the Fine-tuning and Final Evaluation targeted as part of Objective 7 were fruitful, resulting in $+ 2\%$ better performance than the previous best performing model (**Table 4**, Experiment 24) just by adjusting how many samples were used as input to the Audio Extractor, as well as finding the smallest possible model amongst options that performed best, generalising well on the Test Split

(**Table 3**), eliminating computational costs and aligning the HLM12ERC model to the Occam Razor's principle.

Based on that reflection, I believe there is little doubt that the HLM12ERC model, delivered by the present project, fully achieved its original aims, delivering for each one of this project's objectives, with high standards of quality and research rigour.

However, I must not understate how satisfied I am with the high standards with which the present project is presented, and I am confident that this project doesn't just deliver the conclusions it aimed to deliver initially, but also creates a solid baseline from which further work can be developed by fellow researchers.

Self-Reflection

Performing ERC on a multimodal dataset exceeded my expectations of which size of challenge I was undertaking with the present project by a great extent.

Computational costs were a major impediment to conducting further research, especially around Objective 6 (Triplet Contrastive Learning) which required a data sampling strategy that caused considerable oversampling of the dataset, and led to significantly longer training times.

If I were to start this same project today, I would have opted to pick a single aspect of this model and explore in much greater depth than covering such a large surface.

However, exploring such a large solution space such as the one required to build an end-to-end model capable of multimodal classifications was an unparalleled opportunity to deepen my Machine Learning expertise far beyond what it was at the beginning.

Final Evaluation Results

The best performing model, based on the "Final Model Architecture" (**Figure 28**) achieved a 57% Weighted F1-Score on the Test Split. While not at par with the state-of-the-art SPCL-CL-ERC[PapersWithCode, 2023], at 67.7%, it does represent a significant improvement against the baseline architecture ("baseline-dice") outperforming it by 30% on the target metric.

As dictated by the rigour of the praxis [Chollet, 2017], all experiments from 1 to 30 have been evaluated against the Validation Split (**Table 4**). However the Final Evaluation (***Table 3***) has been run against the Test Split, to confirm that the model is generalising appropriately over data unseen during the training and fine-tuning stages.

	support	baseline dice	wav2vec2 200Kx1536
anger	345	13%	43%
disgust	68	0%	0%
fear	50	0%	0%
joy	402	12%	54%
neutral	1256	48%	75%
sadness	208	0%	24%
surprise	281	9%	49%
weighted	2610	27%	57%

Table 3: *Baseline vs Best Weighted F1-Score on Test Split. Source: dev/mlops.ipynb*

The research was organised into two batches of experiments: (1) *Model Design Phase* (experiments 1 to 16), and (2) *Fine-tuning Phase* (experiments 17 to 30).

All experiments were run on Google Colab, using NVidia A100 GPUs, with 40GB of RAM, as training hardware acceleration. Early Stopping was enabled, with patience set to 5 and watching the eval/f1_weighted metric. The Loss and Weighted F1-Score were monitored using Weights & Biases.

The final Learning Rates and Weighted Decay (λ coefficient for L2 Regularisation) were also used for most experiments, generally set to 5×10^{-5} and 0.1 respectively, with a few exceptions for some of the configurations experimented upon.

During the *Model Design Phase*, the "**fusion-mha-adv**" architecture, composed of (a) the GPT2-based text encoder, (b) Wav2Vec2 audio encoder, (c) ResNet50 visual encoder, a (d) 4-headed Attention layer operating on top of a 512D feature vector concatenated from a linear projection of each individual modality, a (e) Single Feedforward layer of 64 units and (f) the DiceLoss as objective function outperformed all others, with a 53% Weighted F1-Score.

GIven the fact that "**fusion-mha-adv**" has resulted in better Validation Weighted F1-Score than its counterparts on the Validation Split during the *Model Design Phase*, the fine-tuning experiments have been conducted towards

optimising its hyperparameters, as well as bringing conclusion to the outstanding research questions.

The *Fine-tuning Phase* focused on a number of different aspects: (a) enhancing the waveform audio encoder with a bigger network and its own multi-headed attention layer, (b) the size of the feature fusion network output, as well as (c) increasing the size of the feedforward network that generated representations prior the logits layer.

Detailed Experimental Results

To reach the "Final Model Architecture", large scale experimentation was required. Further detail about the research process can be found in the *dev/mlops.ipynb* Jupyter notebook.

For convenience, **Table 4** presents the summarised Weighted F1-Score of all experiments run during Development on the Validation Split of the MELD Dataset.

Exp	Model Architecture	Anger F1-score	Disgust F1-score	Fear F1-score	Joy F1-score	Neutral F1-score	Sadness F1-score	Surprise F1-score	Weighted F1-score
	Support	153	22	40	163	470	111	150	**1109**
01	**baseline-cce**	0%	0%	0%	0%	60%	0%	0%	**25%**
02	**baseline-dice**	15%	0%	0%	16%	49%	0%	12%	**27%**
03	**baseline-focal**	0%	8%	2%	3%	54%	0%	11%	**25%**
04	**t-um-glove**	0%	0%	0%	23%	61%	0%	0%	**29%**
05	**t-um-gpt2**	30%	0%	0%	52%	73%	30%	30%	**50%**
06	**t-mm-gpt2**	10%	0%	0%	33%	73%	7%	15%	**40%**
07	**a-um-waveform**	14%	0%	0%	16%	51%	9%	10%	**28%**
08	**a-um-wav2vec2**	0%	0%	0%	0%	60%	0%	0%	**25%**
09	**a-mm-wav2vec2**	0%	0%	0%	0%	59%	0%	0%	**25%**
10	**v-um-resnet50**	0%	0%	0%	0%	57%	0%	14%	**26%**
11	**v-um-retinaface**	0%	0%	0%	0%	60%	0%	0%	**25%**
12	**v-mm-retinaface**	0%	0%	0%	20%	56%	0%	0%	**27%**
13	**fusion-mha-baseline**	13%	0%	0%	17%	54%	4%	13%	**30%**
14	**fusion-concat-adv**	0%	0%	0%	39%	72%	0%	0%	**36%**
15	**fusion-mha-adv**	32%	0%	0%	51%	73%	31%	52%	**53%**
16	**losses-dice_triplet**	18%	0%	0%	4%	48%	3%	14%	**26%**
17	**waveform-150Kx50**	17%	0%	0%	35%	73%	11%	45%	**46%**
18	**waveform-150Kx50_lr5e-4**	15%	0%	0%	34%	66%	9%	17%	**38%**
19	**waveform-150Kx256**	17%	0%	0%	20%	62%	19%	18%	**36%**
20	**waveform-150Kx768**	17%	0%	0%	31%	65%	9%	19%	**38%**
21	**waveform-200Kx256**	25%	0%	0%	32%	73%	14%	37%	**46%**
22	**waveform-200Kx768**	25%	0%	0%	28%	70%	14%	17%	**41%**
23	**waveform-200Kx256mha**	9%	0%	0%	24%	71%	10%	27%	**39%**
24	**wav2vec2-200Kx1536**	38%	0%	0%	48%	75%	32%	55%	**55%**
25	**wav2vec2-250Kx1536**	36%	0%	0%	53%	75%	28%	47%	**54%**
26	**mha1024-ff512x256-wd0_01**	39%	0%	0%	49%	74%	28%	52%	**54%**
27	**mha1024-ff512x512-wd0_01**	37%	0%	0%	52%	74%	30%	49%	**54%**
28	**mha1024-ff1024x512x512_da02**	39%	0%	0%	49%	74%	28%	52%	**54%**
29	**mha1024-ff1024x1024x512x512_da02**	37%	0%	0%	52%	74%	30%	49%	**54%**
30	**l2norm_off**	39%	0%	0%	53%	74%	33%	48%	**54%**

Table 4: *Experiment Results for phase (1) "Model Fine-Tuning". Source: dev/mlops.ipynb*

CONCLUSIONS

Final Considerations

The present project brings forward a solid exploration of the ERC problem space, as well as of a relevant portion of its solution space described by the state-of-the-art models ranking against the test-set provided by the MELD benchmark, such as the M2FNet and the SPCL-CL-ERC models.

It introduces the HLM12ERC model, a solution capable of performing ERC over the 3 modalities present in the MELD dataset: text, audio and video, achieving 57% Weighted F1-Score performance against the MELD Test split, exploiting some of the most modern techniques to represent text (GPT2), audio (Wav2Vec2) and visual features (ResNet50), as well as Multi-headed Attention to support the fusion and weighting of latent features provided by the modal representations.

From the perspective of objective functions, the HLM12ERC demonstrates that the Dice Coefficient Loss can be beneficial and help scenarios where of significant class-imbalance for which the Cross Entropy loss, traditionally employed by classifiers, may not be the best choice. Furthermore, it produces information about and starting point for using triplet loss functions and the custom sampling strategy that they require to be implemented.

From the code perspective, the present project delivers a codebase with the highest standard quality of code and reasonable test-coverage, conveniently open-sourced under the GNU General Public License v3.0, to enable the research community to use the HLM12ERC model as their baseline and extend it for future work.

Finally, the detailed information on results produced by the lengthy experimentation, conducted while designing and fine-tuning what ultimately has become the final HLM12ERC model architecture, has been fully disclosed by this project, with the hope that it will be used to further the community's collective ability of performing better emotion recognition in conversations, as well as help deepen our understanding on how multimodal models work, and how they can be improved.

Future Work

The present project used the ResNet50 as its most basic Visual Feature Extraction mechanism, and it has done so to facilitate the interpretation of the results comparing using the entire scene to using only the faces detected by the ReTinaFace/ResNet-50[Yanjia et al, 2021]. However, in doing so, it did not explore the possibility of using smaller models that could potentially learn representations just as effectively, as well as learn to focus on particular areas of the image using mechanisms such as Attention[Vaswani et al, 2017] which is known to have a weighting effect on features, similar to what has been recently been popularised by Visual Transformers. Further work could explore this solution space further to possibly create a more efficient model.

Given the exceptional results achieved by M2FNet[Chudasama et al, 2022], SPCL-CL-ERC[Song et al, 2022] and the learnings delivered by the current project on the topic, the potential of Contrastive Learning still offers a great opportunity for further exploration. Different sampling strategies such as the one proposed by $ERCDataSampler$ combined with adaptations of the Triplet Loss function devised by the $ERCTripletLoss$ can be studied in more depth and potentially render significant results. However, sampling strategies that result in oversampling as well as the sheer computation costs of training multimodal models must be taken in consideration. Combined with the computational cost and complexity of the topic to be explored, research working on improving classification using Triplet Loss is advised to focus on this specific topic in isolation and go deeper rather than wider.

Finally, given that the HLM12ERC is a native PyTorch model, it could be engineered to be exported as a PreTrainedModel[4] into the Huggingface repository, facilitating for users to download and embed the model to their systems through the transformers API.

[4] Huggingface PreTrainedModel: https://huggingface.co/(...)/model

REFERENCES

[Paszke et al, 2019] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library.

[Radford, 2019] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. (2019).

[Baevski et al, 2020] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. 2020. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. In Advances in Neural Information Processing Systems, Curran Associates, Inc., 12449–12460. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2020/file/92d1e1eb1cd6f9fba3227870bb6d7f07-Paper.pdf

[Vaswani et al, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In Advances in Neural Information Processing Systems, Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[Kratzwald et al, 2018] Bernhard Kratzwald, Suzana Ilić, Mathias Kraus, Stefan Feuerriegel, and Helmut Prendinger. 2018. Deep learning for affective computing: Text-based emotion recognition in decision support. Decision Support Systems 115, (2018), 24–35. DOI: https://doi.org/10.1016/j.dss.2018.09.002

[McFee et al, 2023] Brian McFee, Matt McVicar, Daniel Faronbi, Iran Roman, Matan Gover, Stefan Balke, Scott Seyfarth, Ayoub Malek, Colin Raffel, Vincent Lostanlen, Benjamin van Niekirk, Dana Lee, Frank Cwitkowitz, Frank Zalkow, Oriol Nieto, Dan Ellis, Jack Mason, Kyungyun Lee, Bea Steers, Emily Halvachs, Carl Thomé, Fabian Robert-Stöter, Rachel Bittner, Ziyao Wei, Adam Weiss, Eric Battenberg, Keunwoo Choi, Ryuichi Yamamoto, CJ Carr, Alex Metsai, Stefan Sullivan, Pius Friesch, Asmitha Krishnakumar, Shunsuke Hidaka, Steve Kowalik, Fabian Keller, Dan Mazur, Alexandre Chabot-Leclerc, Curtis Hawthorne, Chandrashekhar Ramaprasad, Myungchul Keum, Juanita Gomez, Will Monroe, Viktor Andreevitch Morozov, Kian Eliasi, nullmightybofo, Paul Biberstein, N. Dorukhan Sergin, Romain Hennequin, Rimvydas Naktinis, beantowel, Taewoon Kim, Jon Petter Åsen, Joon Lim, Alex Malins, Darío Hereñú, Stef van der Struijk, Lorenz Nickel, Jackie Wu, Zhen Wang, Tim Gates, Matt Vollrath, Andy Sarroff, Xiao-Ming, Alastair Porter, Seth Kranzler, Voodoohop, Mattia Di Gangi, Helmi Jinoz, Connor Guerrero, Abduttayyeb Mazhar, toddrme2178, Zvi Baratz, Anton Kostin, Xinlu Zhuang, Cash TingHin Lo, Pavel Campr, Eric Semeniuc, Monsij Biswal, Shayenne Moura, Paul Brossier, Hojin Lee, and Waldir Pimenta. 2023. librosa/librosa: 0.10.0.post2. Zenodo. DOI:https://doi.org/10.5281/zenodo.7746972

[Harris et al, 2020] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. Nature 585, 7825 (September 2020), 357–362. DOI:https://doi.org/10.1038/s41586-020-2649-2

[Cer et al, 2018] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Brian Strope, and Ray Kurzweil. 2018. Universal Sentence Encoder for English. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Association for Computational Linguistics, Brussels, Belgium, 169–174. DOI:https://doi.org/10.18653/v1/D18-2029

[Wu et al, 2019] Dong Zhang, Liangqing Wu, Changlong Sun, Shoushan Li, Qiaoming Zhu, and Guodong Zhou. 2019. Modeling both Context- and Speaker-Sensitive Dependence for Emotion Detection in Multi-speaker Conversations. In Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence,

IJCAI-19, International Joint Conferences on Artificial Intelligence Organization, 5415–5421. DOI:https://doi.org/10.24963/ijcai.2019/752

[Pedregosa et al, 2011] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12, (2011), 2825–2830.

[Milletari et al, 2016] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. 2016. V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation. In 2016 Fourth International Conference on 3D Vision (3DV), 565–571. DOI:https://doi.org/10.1109/3DV.2016.79

[Schroff et al, 2015] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. FaceNet: A unified embedding for face recognition and clustering. In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 815–823. DOI:https://doi.org/10.1109/CVPR.2015.7298682

[Chollet, 2017] Francois Chollet. 2017. Deep Learning with Python (1st ed.), Manning Publications Co., USA.

[Kimm et al, 2021] H. Kimm, I. Paik and H. Kimm, "Performance Comparision of TPU, GPU, CPU on Google Colaboratory Over Distributed Deep Learning," 2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), Singapore, Singapore, 2021, pp. 312-319, doi: 10.1109/MCSoC51149.2021.00053.

[Sharp et al, 2019] Helen Sharp, Jennifer Preece, and Yvonne Rogers. 2019. Interaction Design : Beyond Human-Computer Interaction. John Wiley & Sons, Incorporated, Newark. Retrieved from http://ebookcentral.proquest.com/lib/londonww/detail.action?docID=5746446

[Mendes/XLA, 2023] Hudson Mendes,. 2023. "Error when attempting to access XLA tensor.shape", Torch Discussion Forums. Retrieved from: https://discuss.pytorch.org/t/error-when-attempting-to-access-xla-tensor-shape/186214

[Mendes/EDA, 2023] Hudson Mendes. 2023. CM3070 Final Project, Exploratory Data Analysis. University of London. Retrieved from https://github.com/hudsonmendes/cm3070-fp/blob/87252d525d84c79f2f1cd35fefaf190a785872fc/dev/eda.ipynb

[Pennington et al, 2014] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In Empirical Methods in Natural Language Processing (EMNLP), 1532–1543. Retrieved from http://www.aclweb.org/anthology/D14-1162

[He et al, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, 770–778.

[Zhang et al, 2018] Minghua Zhang, Yunfang Wu, Weikang Li, and Wei Li. 2018. Learning Universal Sentence Representations with Mean-Max Attention Autoencoder. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 4514–4523.

[Majumder et al, 2019] Navonil Majumder, Soujanya Poria, Devamanyu Hazarika, Rada Mihalcea, Alexander Gelbukh, and Erik Cambria. 2019. DialogueRNN: An Attentive RNN for Emotion Detection in Conversations. Proceedings of the AAAI Conference on Artificial Intelligence 33, 1 (2019), 6818–6825. DOI:https://doi.org/10.1609/aaai.v33i01.33016818

[PapersWithCode/IEMOCAP, 2023] Papers with Code, "Emotion Recognition in Conversation on IEMOCAP". Source: https://paperswithcode.com/sota/emotion-recognition-in-conversation-on

[PapersWithCode, 2023] Papers with Code, "Emotion Recognition in Conversation on MELD". Source: https://paperswithcode.com/sota/emotion-recognition-in-conversation-on-meld

[Liu et al, 2023] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. ACM Comput. Surv. 55, 9, Article 195 (September 2023), 35 pages. https://doi.org/10.1145/3560815

[Poria et al, 2019] Soujanya Poria, Devamanyu Hazarika, Navonil Majumder, Gautam Naik, Erik Cambria, and Rada Mihalcea. 2019. MELD: A Multimodal Multi-Party Dataset for Emotion Recognition in Conversations. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Florence, Italy, 527–536. DOI:https://doi.org/10.18653/v1/P19-1050

[Mihalceae et al, 2019] Soujanya Poria, Navonil Majumderd, Rada Mihalceae, and Eduard Hovy. 2019. Emotion Recognition in Conversation: Research Challenges, Datasets, and Recent Advances. IEEE Access PP, (July 2019), 1–1. DOI:https://doi.org/10.1109/ACCESS.2019.2929050

[Pandas, 2020] The pandas development team. 2020. pandas-dev/pandas: Pandas. Zenodo. DOI:https://doi.org/10.5281/zenodo.3509134

[Wolf et al, 2020] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Association for Computational Linguistics, Online, 38–45. Retrieved from https://www.aclweb.org/anthology/2020.emnlp-demos.6

[Gao et al, 2021] Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple Contrastive Learning of Sentence Embeddings. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Online, 6894–6910. DOI:https://doi.org/10.18653/v1/2021.emnlp-main.552

[Lin et al, 2017] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal Loss for Dense Object Detection. In 2017 IEEE International Conference on Computer Vision (ICCV), 2999–3007. DOI:https://doi.org/10.1109/ICCV.2017.324

[Chudasama et al, 2022] Vishal Chudasama, Purbayan Kar, Ashish Gudmalwar, Nirmesh Shah, Pankaj Wasnik, and Naoyuki Onoe. 2022. M2FNet: Multi-modal Fusion Network for Emotion Recognition in Conversation. In 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 4651–4660. DOI:https://doi.org/10.1109/CVPRW56347.2022.00511

[Li et al, 2023] Wenyi Li, Shengjie Zheng, Yufan Liao, Rongqi Hong, Chenggang He, Weiliang Chen, Chunshan Deng, and Xiaojian Li. 2023. The brain-inspired decoder for natural visual image reconstruction. Frontiers in Neuroscience 17, (2023). DOI:https://doi.org/10.3389/fnins.2023.1130606

[Song et al, 2022] Xiaohui Song, Longtao Huang, Hui Xue, and Songlin Hu. 2022. Supervised Prototypical Contrastive Learning for Emotion Recognition in Conversation. In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 5197–5206. Retrieved from https://aclanthology.org/2022.emnlp-main.347

[Simard et al, 1994] Y. Bengio, P. Simard, and P. Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks 5, 2 (1994), 157–166. DOI:https://doi.org/10.1109/72.279181

[Yanjia et al, 2021] Yanjia Zhu, Hongxiang Cai, Shuhan Zhang, Chenhao Wang, and Yichao Xiong. 2021. TinaFace: Strong but Simple Baseline for Face Detection.

[Bengio et al, 2009] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In Proceedings of the 26th Annual International Conference on Machine Learning (ICML '09). Association for Computing Machinery, New York, NY, USA, 41–48. https://doi.org/10.1145/1553374.1553380