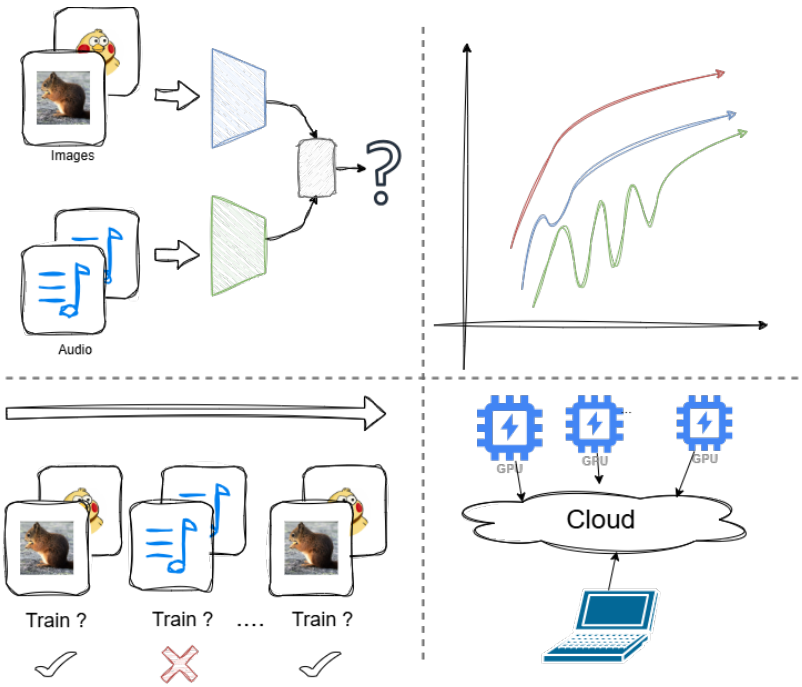
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG




# Optimising Multimodal Learning with a System-Aware Perspective

## Focusing on modality imbalance and training optimisation

Master's thesis in Computer science and engineering

Hugo Manuel Alves Henriques e Silva
Hongguang Chen

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

# Optimising Multimodal Learning with a System-Aware Perspective

Focusing on modality imbalance and training optimisation

Hugo Manuel Alves Henriques e Silva
Hongguang Chen

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Optimising Multimodal Learning with a System-Aware Perspective
Focusing on modality imbalance and training optimisation
Hugo Manuel Alves Henriques e Silva, Hongguang Chen



Supervisor: Selpi, Data Science and AI, Computer Science and Engineering
Examiner: Gerardo Schneider, Data Science and AI, Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Multimodal Model, Training scheduling, Training on Cloud

Typeset in LaTeX
Gothenburg, Sweden 2025

Optimising Multimodal Learning with a System-Aware Perspective
Focusing on modality imbalance and training optimisation
Hugo Manuel Alves Henriques e Silva, Hongguang Chen
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

# Abstract

Multimodal learning in machine learning (ML) aims to improve model performance by integrating multiple modalities, mirroring human perception. However, there are several challenges when training such multimodal models from scratch. These challenges include (1) modality imbalance, where certain modalities dominate the learning process and limit the contribution of others; (2) modality prediction bias, where models disproportionately rely on specific modalities during inference despite the presence of equally informative alternatives; and (3) inefficient resource allocation, resulting from the overtraining of specific modalities and suboptimal utilisation of available hardware. This thesis tackles these issues by proposing a system-aware approach that incorporates algorithmic modality rebalancing techniques, optimisations driven by high-performance computing (HPC), and dynamic adjustments of modality contributions to improve multimodal model performance, training efficiency, and scalability.

By focusing on the previous three key challenges, we introduce novel optimisation strategies to detect and mitigate modality imbalance arising from divergent learning curves. We prevent modality suppression by addressing modality bias during prediction. Finally, we address resource inefficiencies by designing adaptive training procedures, load-aware parallelisation, and dynamic scheduling to optimise GPU utilisation and reduce unnecessary computation.

The proposed methods are evaluated on several multimodal datasets, including CREMA-D, AVE, and IEMOCAP, among others. Experimental results demonstrate the effectiveness of our system-aware optimisations, showing substantial improvements in both model performance and computational efficiency. The research also investigates multi-GPU optimisation strategies to further enhance the scalability of multimodal learning systems in high-performance computing environments.

This work provides a comprehensive approach to advancing multimodal learning by addressing both algorithmic and system-level challenges, contributing to the development of more efficient, scalable, and energy-conscious AI models.

Keywords: Machine Learning, Multimodal Learning, Modality Imbalance, Modality Prediction Bias, High-Performance Computing (HPC), Training Efficiency, Scalability, GPU Utilisation

# Acknowledgements

We would like to express our sincere gratitude to all those who have supported and helped us during this thesis.

We are especially grateful to our supervisor, Selpi, and our examiner, Gerardo Schneider, for their valuable guidance and insightful feedback throughout the research process.

We also thank all the classmates and colleagues who contributed through discussions and suggestions, which greatly helped improve the quality of this work.

Finally, we would like to acknowledge the Alvis platform, provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725, for providing the computational resources used in this research.

Hugo Manuel Alves Henriques e Silva
Hongguang Chen
Gothenburg, 2025-05-21

# Contents

# List of Figures

# List of Tables

# 1

# Introduction

## 1.1 Background and Motivation

Multimodal learning is a rapidly evolving field in Machine Learning that seeks to integrate information from multiple modalities, such as text, audio, images, and video, to improve learning outcomes. Unlike traditional unimodal approaches, which rely on a single data source, multimodal learning leverages the complementary strengths of diverse sensory inputs. This approach is inspired by human perception, where individuals seamlessly process and fuse information from different senses to interpret their environment, make decisions, and respond effectively to complex scenarios.

The importance of multimodal learning extends across numerous AI applications, driving advancements in areas such as autonomous systems, healthcare, human-computer interaction, and content understanding. In speech recognition and emotion detection, for example, integrating both auditory and visual cues leads to more accurate and robust models [1]. In medical diagnostics, multimodal AI combines imaging data, patient history, and genomic information to improve disease prediction and treatment recommendations [2]. Furthermore, in autonomous driving, the combination of data from cameras, LiDar, and radar improves the ability to detect objects and make decisions in dynamic environments [3].

As AI systems become more sophisticated, the demand for efficient and scalable multimodal models continues to grow. However, several challenges persist, particularly in handling modality imbalance, optimising training efficiency, and designing resource-aware computing strategies. Addressing these challenges is crucial for developing more accurate, robust, and computationally efficient multimodal AI models, ultimately advancing the field towards more intelligent, human-like learning systems.

## 1.2 Challenges in the Training of Multimodal Models

Despite recent advances in multimodal learning, training multimodal models still presents significant challenges. These challenges often prevent the model from meeting expectations and, in some cases, even result in inferior performance compared to

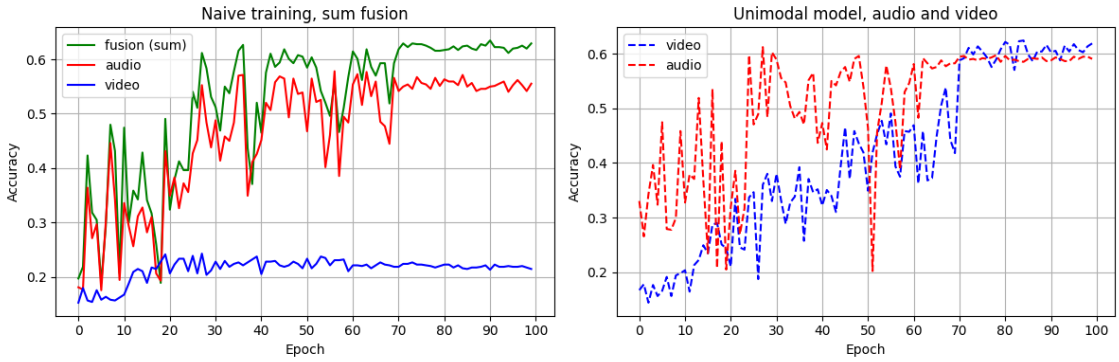



Figure 1.1: These two figures illustrate significant challenges in training a multimodal model using the conventional joint training method. **(Left)** Accuracy on the validation set with conventional joint training. The green curve represents the fusion result, the red curve corresponds to the audio modality's performance within the multimodal model, and the blue curve represents the visual modality. **(Right)** Accuracy of each modality when trained independently in a single-modality model. The red dashed line represents the audio modality, while the blue dashed line represents the visual modality. **Dataset**: CREMA-D [4]. **Training setup**: Both joint and single-modality models are trained using the same hyperparameters.

the best unimodal model when using conventional training methods. Figure 1.1 displays two plots that illustrate one of the most significant challenges that arise when training a multimodal model using a conventional joint training method. In these training methods, the modalities are fused and optimised jointly using a single loss function. When closely examining the plots, we see that the visual modality performs noticeably worse in the multimodal setting, represented by the blue curve in the left plot, compared to its performance in the unimodal setting, shown as the blue dashed line in the right plot. This clearly illustrates how the visual modality is inhibited in a multimodal context.

### 1.2.1 Modality Imbalance

The first challenge in Multimodal Learning is the integration of multiple modalities without causing interference between them, which can result in the loss of modality-specific information. It can be manifested in mainly three key ways:

- Divergent learning curves – Different modalities learn at varying rates, making it difficult to optimise training uniformly.
- Modality suppression – Dominant modalities overshadow weaker ones, reducing their contribution to the overall model.
- Inefficient modality integration – Without tailored optimisation strategies, the fusion of multiple modalities can lead to suboptimal performance.

### 1.2.2 Modality Prediction Contribution

The second challenge involves balancing the contribution of each modality towards the final prediction. In different late fusion strategies, it is possible to directly obtain the contribution of each modality towards the final prediction. During the training of a multimodal model, there is a tendency to form a bias towards certain modalities over others. This leads to biased modalities becoming overconfident and overshadowing the contributions of other modalities. When an inherent bias is introduced in favour of an undesired modality, overall model performance may be compromised. It is therefore essential to establish a balanced multimodal learning setup in which each modality can fully realise its learning potential, while carefully assessing its contribution to the final prediction.

### 1.2.3 Computational Bottlenecks in Training

Additionally, from a high-performance computing (HPC) perspective, current multimodal training paradigms suffer from **resource inefficiencies**, including:

- Unnecessary computation due to fixed training parameters (e.g., identical epochs for all modalities), leading to wasted GPU cycles.
- Underutilised GPU resources, even when optimised with PyTorch framework auto-optimisation, like multi-number data loader.
- Inefficient parallelisation and workload distribution, failing to leverage multi-GPU and heterogeneous computing environments effectively.

## 1.3 Research Goal and Contributions

The goal of this research is to enhance the performance of multimodal models while improving training efficiency, scalability, and energy consumption. The main topics covered are:

- Understanding the mathematical foundations of the widely acknowledged Modality Imbalance problem, where certain modalities dominate the learning process and suppress the contribution of others;
- Identifying and defining a new problem: Modality Prediction Bias, where models disproportionately rely on specific modalities during inference, even when others are equally informative;
- Formally evaluating Modality Imbalance and Modality Prediction Bias, and introducing targeted metrics to quantify and analyse their impact during training and inference;
- Introducing novel methods that build on the strengths of previous state-of-the-art approaches, while extending them to effectively address both Modality

Imbalance and Modality Prediction Bias through various late fusion strategies, with particular attention to training efficiency and scalability.

- Applying system-aware optimisations that overcome the efficiency problem of the current multi-GPU training method, which ignores the structure of the model and the mismatch between high-power GPU devices and the data loading pipeline.

To achieve these goals, we explore a system-aware approach that integrates algorithmic modality rebalancing techniques and HPC-driven optimisations, aiming to optimise multimodal learning from both algorithmic and system-level perspectives.

## 1.4 Related Work

### 1.4.1 Modality Imbalance

Several studies have explored the challenges of modality imbalance in multimodal learning.

In [5], researchers identified **overfitting** as a key factor contributing to modality imbalance, preventing models from achieving optimal performance. To address this, they introduced the **Overfitting Generalisation Rate** metric and proposed **Gradient Blending**, a method designed to suppress overfitting during training. However, despite this approach, slower-learning modalities still underperformed compared to their single-modality counterparts.

Building on this idea, **UMT** [6] introduced a **unimodal teacher model** to guide the multimodal learning process using knowledge distillation techniques. While this approach improved modality alignment, it introduced significant computational overhead, making it impractical for many real-world applications. In contrast, **OGM-GE** [7] aimed to mitigate learning conflicts without adding extra model structures. The authors argued that discrepancies in loss gradients were the primary cause of learning imbalance and proposed a gradient modulation strategy that slowed down faster-learning modalities based on contribution discrepancy. However, this method was limited to two modalities due to the method of computing contribution discrepancies.

To overcome this limitation, **AGM** [8] extended the approach to support more than two modalities and complex model structures by introducing the concept of **"Shapley value"**, which quantitatively evaluates each modality's contribution to the overall multimodal model. This enhancement made the method applicable to a broader range of fusion architectures beyond late fusion.

Further advancing this direction, **PMR** [9] built upon OGM-GE and argued that dominant modalities not only suppress the learning rates of weaker modalities but also interfere with their gradient update directions. To counteract this, they introduced **"prototypical cross-entropy loss"** to enhance weaker modalities and incorporated

**"prototypical entropy regularisation (PER)"** to penalise dominant modalities, thereby preventing premature convergence and alleviating suppression effects.

Meanwhile, other works have focused on **modulating learning rates** to address modality imbalance. **ATF** [10] decoupled the training processes of unimodal and multimodal networks, dynamically balancing learning rates across modalities. The method applied in [11] updated a modality branch only when its **conditional learning speed** exceeded a predefined threshold. **MSLR** [12] adopted a more straightforward approach by applying different learning rates to each modality within a late-fusion framework.

More recent studies have explored other possible solutions. The **suppress and balance** method [13] dynamically adjusted the backward-propagated gradients according to their degree of conflict and convergence speed. In [14], researchers considered global modality discrepancies at the dataset level, demonstrating improvements in commonly curated datasets. In addition, they applied adaptive resampling techniques to enhance the discriminative ability of low-contributing modalities, indirectly improving their impact on the final multimodal prediction.

The current state-of-the-art, **MLA** [15], attributes the inefficiency of multimodal learning to **"modality laziness"**, where certain modalities fail to fully integrate rich multimodal knowledge. The authors argued that previous methods, which updated all encoders simultaneously, led to suboptimal adaptation for weaker modalities. To address this, MLA introduced an approach that decomposes conventional multimodal joint optimisation into an alternating unimodal learning process combined with a test-time dynamic modality fusion mechanism, ensuring better integration of multimodal information.

### 1.4.2 Training on a Cluster Environment

As deep learning models become increasingly large and computationally demanding, optimising resource utilisation has become essential. Training models on GPU clusters has become the mainstream approach, driven by both financial considerations and environmental sustainability. Given the increasing computational requirements of modern models, improving training efficiency remains a crucial research topic.

Many studies have explored different aspects of training optimisation, spanning from low-level system optimisations to high-level training strategies. For instance, in [16], [17], researchers analyse the performance of different data loader implementations in large-scale training scenarios, highlighting the impact of data loading efficiency on overall training performance.

Beyond optimising data loaders, another key challenge is parallelising training. Various parallelisation strategies have been proposed, such as **Data Parallelism** [18] and **Model Parallelism** [19], each with its own limitations.

To address some of these challenges, [20] introduces a pipelined model parallelism

technique that achieves high GPU utilisation while maintaining robust training accuracy through a novel weight prediction mechanism. Similarly, [21] proposes a hybrid approach that combines data parallelism and model parallelism to mitigate efficiency losses when scaling up training workloads.

However, multimodal models have received relatively little attention in this context. Due to their inherently multi-branch structure, they differ significantly from conventional models and cannot be fully optimised using existing parallelisation techniques. This gap underscores the need for specialised training strategies tailored to multimodal architectures.

## 1.5 Overview of Datasets

To support our theoretical analysis and experiments, we briefly introduce the datasets we used for this thesis.

Table 1.1: Modalities of Different Datasets

| **Dataset** | **Audio** | **Visual** | **Text** |
|---|---|---|---|
| CREMA-D | ✓ | ✓ | - |
| AVE | ✓ | ✓ | - |
| MVSA | - | ✓ | ✓ |
| IEMOCAP | ✓ | ✓ | ✓ |
| UR-FUNNY | ✓ | ✓ | ✓ |

- The Crowd-Sourced Emotional Multimodal Actors Dataset (CREMA-D) consists of 7,442 original clips obtained from 48 male and 43 female actors between the ages of 20 and 74 from a variety of races and ethnicities. Actors spoke from a selection of 12 sentences, which were presented using one of six different emotions (Anger, Disgust, Fear, Happy, Neutral, and Sad).

- The Audio-Visual Event (AVE) [22] dataset consists of 4,143 10-second video clips covering 28 different event categories, with both audio and visual modalities.

- The Multimodal Sentiment Analysis (MVSA) dataset [23] contains 5,129 image-text pairs collected from Twitter, labelled into three sentiment classes: positive, neutral, and negative. The dataset includes both visual and textual modalities.

- The Interactive Emotional Dyadic Motion Capture (IEMOCAP) dataset [24] contains approximately 12 hours of audiovisual data, including video, speech, facial motion capture, and text transcriptions, labelled with emotions such as angry, excited, frustrated, neutral, and sad. The dataset includes audio, visual, and textual modalities.

- The UR-FUNNY dataset [25] consists of 16,514 video segments, labelled as humorous or non-humorous. It includes visual, audio and textual modalities.

# 2

# Theory

This chapter presents the theoretical foundation of the research, covering the following topics:

- An overview of fusion methods employed in this study.
- A discussion of the mathematical reasons behind the Modality Imbalance problem.
- An examination of traditional optimisation techniques for single-modality models.
- A review of recent solutions proposed in the literature.
- An introduction to the newly proposed "Modality Prediction Bias" problem.
- A discussion on system-aware approaches to accelerate training with high power GPUs.

## 2.1 Overview of Multimodal Fusion Methods

In multimodal learning, various fusion strategies are applied to combine information from different modalities. We first provide an overview and description of the overall model architecture adopted in this work. Next, we briefly describe the different modality fusion stages. Lastly, we describe the most commonly used late fusion techniques, which are also used in this study: Summation Fusion [7], [26], Concatenation [27], FiLM [28], Gated [29]. Notably, we observed that there are two distinct definitions of Summation fusion. The first one [26] consists of a general definition, shown in Figure 2.3, where modality embeddings are summed before being fed into the classification layer. In the second Summation Fusion approach [7], illustrated in Figure 2.4, the logits from each modality are computed independently and then summed before applying the softmax function for prediction. The summation fusion mechanism used in OGM-GE, AGM, PMR and MLA is the second one. To avoid ambiguities, in this study, we named the first sum fusion mechanism as **Early-Sum** and the second as **Late-Sum**.

### 2.1.1 Full Model Architecture

Figure 2.1 displays the full model architecture adopted for this study for two modalities, audio and visual. In the first stage, each modality's raw data can be preprocessed as desired before being input into its respective encoder. After preprocessing, each modality is passed through its encoder to generate a latent representation. Once all modalities have been processed, their representations are fused into a single multimodal latent representation, which is then passed to the fusion head for prediction. The multimodal latent representation and the fusion head's structure will vary depending on the late-fusion mechanism employed. Importantly, this architecture generalises to any number of modalities, with each modality having its own dedicated encoder. Additionally, we include helper classifiers for each modality to evaluate their individual performance, which is particularly useful when working with complex fusion structures.

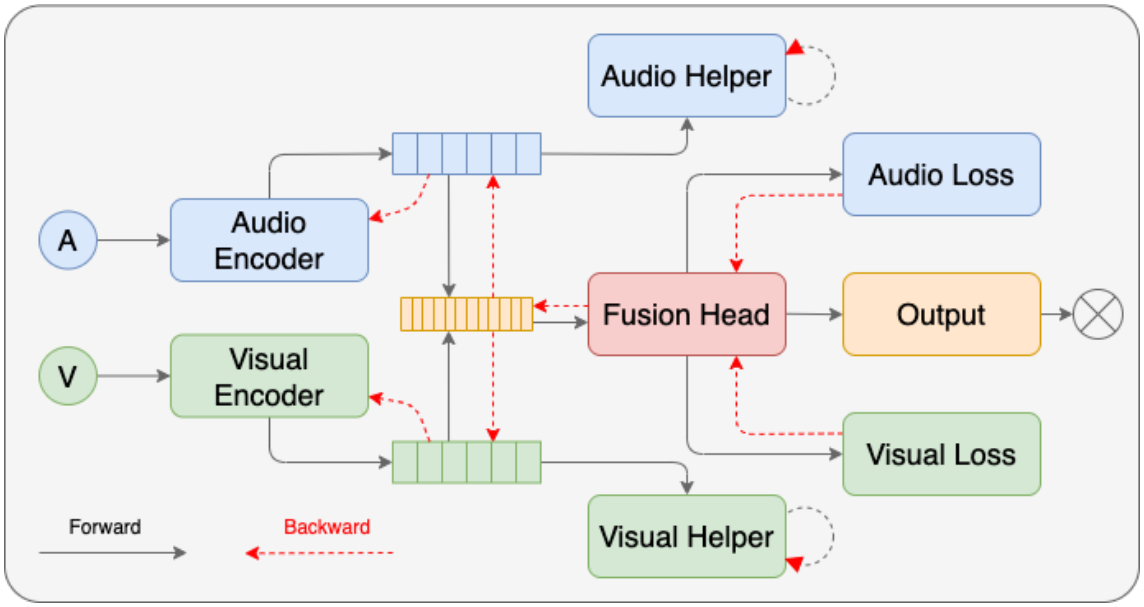



Figure 2.1: Full model architecture for 2 modalities, Audio (A) and Visual (V). One encoder and one helper classifier per modality, and one fusion head.

### 2.1.2 Modality Fusion Stages

Figure 2.2 illustrates the three main stages of modality fusion. In **Early Fusion**, modalities are combined at the data level, prior to core processing such as encoding. **Intermediate Fusion** involves integrating modalities at the encoder level, allowing intercommunication between encoders. The focus of this work is on **Late Fusion**, where modalities are fused after being independently processed by their respective encoders. This can occur either immediately after encoding, by generating a joint latent representation, or at the prediction stage, by combining unimodal outputs. More information on modality fusion stages can be found in [30].

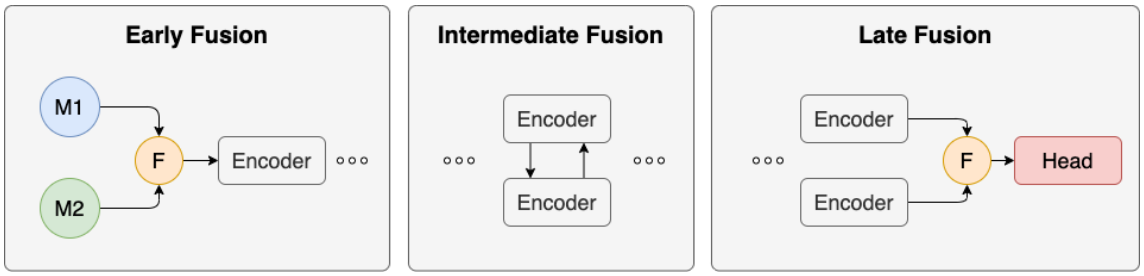



Figure 2.2: Different modality fusion stages in a multimodal model.

### 2.1.3 Late Fusion Strategies

In the following sections, we describe the most frequently used late-fusion mechanisms. For all descriptions, $M$ denotes the total number of modalities, $m$ refers to a single modality ($m \in M$), $e_m$ represents the embedding of modality $m$, $W$ denotes the weight matrix of a linear layer, $b$ is the corresponding bias term, and $z_m$ represents the logits of modality $m$.

#### 2.1.3.1 Early-Sum Fusion

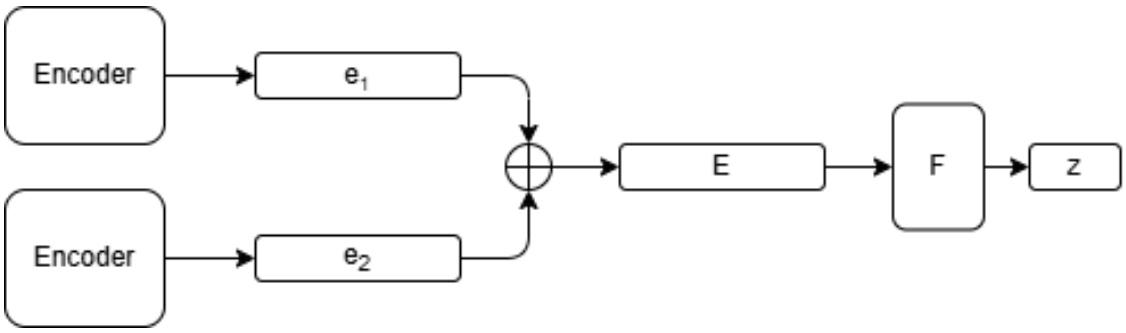



Figure 2.3: Early-Sum Fusion of two modalities.

Early-Sum Fusion consists of summing the embeddings of different modalities before feeding them into the classification layer. Therefore, given M modalities, we form a single fused embedding, $E$, given by the expression:

$$E = \sum e_m, \quad m \in M \tag{2.1}$$

We further obtain the logit outputs, $z$, by feeding the fused embeddings into the classification layer:

$$z = F(E) \tag{2.2}$$

where $F$ is a linear layer. Thus, the full early-sum fusion method is given by:

$$\begin{aligned} z &= W[\sum e_m] + b \\ &= W(e_1 + ... + e_M) + b \\ &= We_1 + ... + We_M + b \end{aligned} \tag{2.3}$$

where $W$ is the weight of $F$, $b$ is the bias term.

When implementing this fusion mechanism, we can obtain unimodal predictions in a multimodal context by feeding each modality's embeddings into the classification layer F. This way, we obtain individual modality logits:

$$z_m = F(e_m) = We_m + b, \quad m \in M \tag{2.4}$$

If we want to obtain the fused final logits given in Eq. (2.3) from the individual modality logits given in Eq. (2.4), we have that:

$$z = \sum(z_m - b) + b = \sum z_m + c, \quad c \in \mathbb{R}, \quad m \in M \tag{2.5}$$

which essentially corresponds to summing all the logits adjusted by a constant value.

#### 2.1.3.2 Late-Sum Fusion

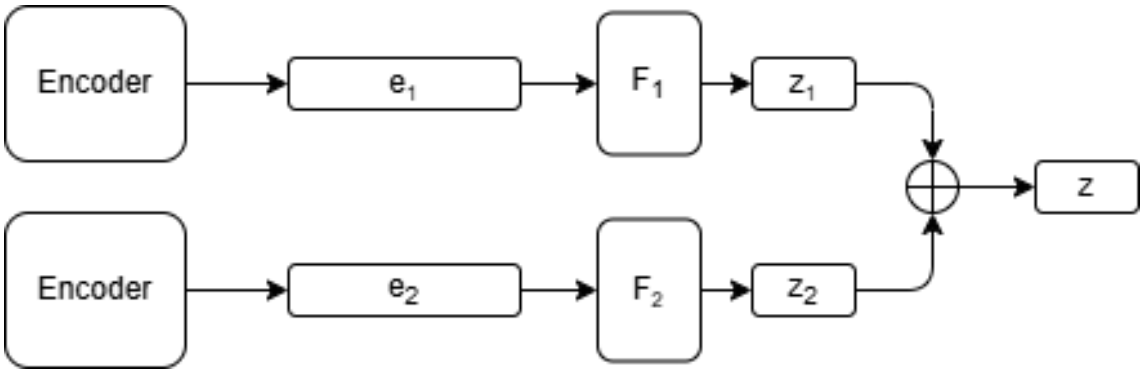



Figure 2.4: Late-Sum Fusion of two modalities.

The Late-Sum Fusion method combines two modality feature vectors by summing them after they are processed by separate linear layers, as demonstrated in Figure 2.4. This simple yet effective method helps to merge information from different modalities. The individual modality logits are obtained in the following manner:

$$z_m = F_m(e_m) = W_m e_m + b_m \tag{2.6}$$

The full forward pass can be described as follows:

$$z = \sum F_m(e_m) = \sum W_m e_m + b_m = \sum z_m \tag{2.7}$$

where, for the modality m, $F_m$ is its fully connected layer that transforms the input features $e_m$ into an output dimension, $W_m$ the layer's weight parameters and $b_m$ the bias term. This method does not require the modalities to have the same feature space, as they are processed independently before fusion. The final logits are obtained by summing the individual modality logits.

#### 2.1.3.3 Concatenation (Concat) Fusion

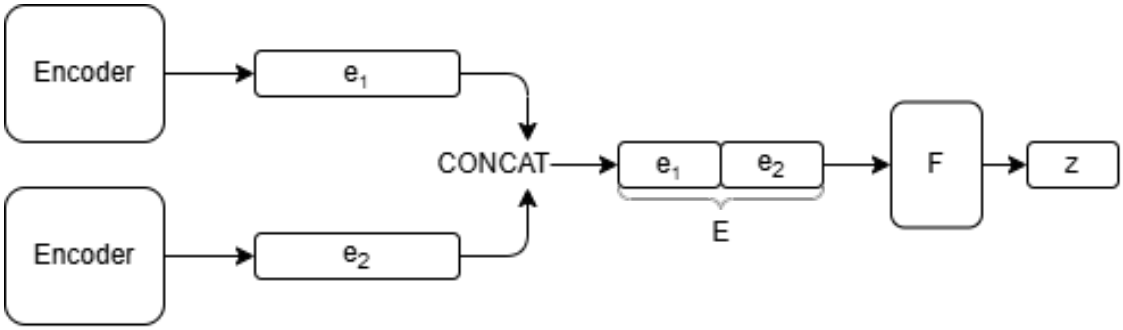



Figure 2.5: Concatenation Fusion of two modalities.

Concatenation Fusion involves combining the feature vectors from multiple modalities by concatenating them along the feature dimension, as Figure 2.5 shows. This method produces a larger feature vector that is then passed through a final fully connected layer. Mathematically, individual modality logits are computed as:

$$z_m = W_m e_m + b, \quad m \in M \tag{2.8}$$

and the fusion output **z** is computed as:

$$\begin{aligned} z &= F([e_0, e_1, ..., e_M]) \\ &= W[e_0, e_1, ..., e_M] + b \\ &= [W_0, W_1, ..., W_M]^T [e_0, e_1, ..., e_M] + b \\ &= \sum (z_m - b) + b \\ &= \sum z_m + c, \quad c \in \mathbb{R}, \quad m \in M, \end{aligned} \tag{2.9}$$

where $F$ is a linear layer with weight matrix $W$ and bias term $b$, and $W_m$ denotes the subset of weights in $W$ corresponding to modality $m$ within that layer. $[,]$ represents the concatenation operator.

Just as both sum fusion mechanisms, the final fused logits in concatenation fusion can be obtained by summing the individual modality logits, in a multimodal scenario, adjusted by a constant value.

#### 2.1.3.4 Gated Fusion

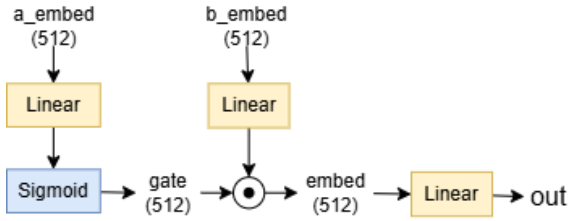



Figure 2.6: Gate Fusion - Only applicable to two modalities.

Gated Fusion is a late fusion mechanism only applicable to two modalities that introduces a gating mechanism to control how much each modality should contribute to the final fused representation, as Figure 2.6 demonstrates. The gate is computed as a sigmoid transformation applied to one modality's features, and it is used to modulate the other modality's features. For two modalities, $a$ and $b$, The final output $\mathbf{z_c}$ is given by:

$$z_c = \begin{cases} \mathrm{FC}_{\mathrm{out}}(\sigma(\mathrm{FC}_a(e_a)) \odot \mathrm{FC}_b(e_b)) & \text{if gate with } e_a \\ \mathrm{FC}_{\mathrm{out}}(\sigma(\mathrm{FC}_b(e_b)) \odot \mathrm{FC}_a(e_a)) & \text{if gate with } e_b \end{cases} \tag{2.10}$$

where $\sigma$ denotes the sigmoid activation function, $\mathrm{FC}_{[a,b]}$ is a fully connected layer for modality features $\mathbf{e_a}$, and $\mathbf{e_b}$, , and $\odot$ represents element- wise multiplication.

#### 2.1.3.5 FiLM Fusion

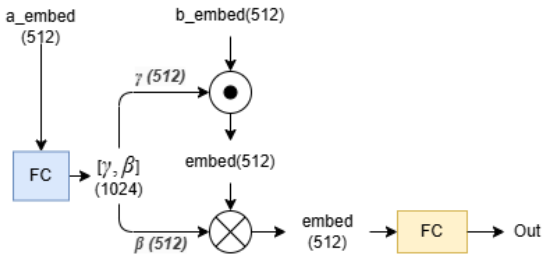



Figure 2.7: Film Fusion - Only applicable to two modalities.

Similarly to Gated Fusion, FiLM (Feature-wise Linear Modulation) Fusion is only applicable to two modalities. It modulates one modality's features by applying a learned linear transformation based on the features from another modality, as shown in Figure 2.7. Specifically, for modalities $a$ and $b$ and respective given input features $\mathbf{e_a}$ and $\mathbf{e_b}$, the fusion output $\mathbf{z_c}$ is computed as:

$$\mathbf{z_c} = \gamma(\mathbf{e_a}) \odot \mathbf{e_b} + \beta(\mathbf{e_a}), \tag{2.11}$$

where $\gamma(\mathbf{e_a})$ and $\beta(\mathbf{e_a})$ denote the linear projection to $\mathbf{e_a}$, and $\odot$ represents element-wise multiplication.

## 2.2 Performance Evaluation of Individual Modalities in Multimodal Models

Understanding the individual effectiveness of different modalities is crucial for improving overall model performance.

### Helper Classifiers (Probes)

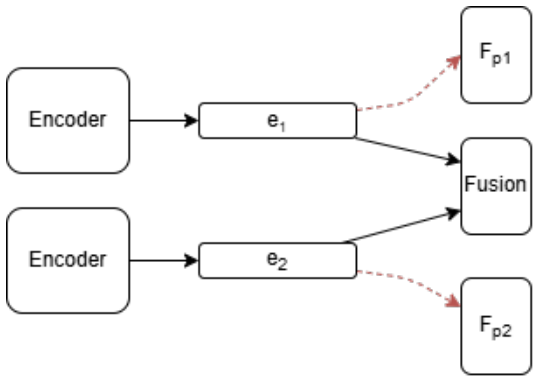



Figure 2.8: The probing mechanism. $F_{p1}$ and $F_{p2}$ are linear classification layers used to assess individual modality performance.

To directly evaluate the contribution and predictive performance of each modality (or encoder), we employ the linear probing technique, as used in [8], [31]. Linear probing involves training a simple additional classifier, specifically, a linear classifier, which we refer to as a probe in this study. The probe is trained using the embeddings produced by individual modalities, allowing us to isolate and measure their effectiveness. Figure 2.8 illustrates the probing mechanism, where $F_{p1}$ and $F_{p2}$ represent individual linear classification layers applied to different modality embeddings.

### Modality-wise Accuracy Measurement

To quantify the contribution of each modality, we adopt different approaches depending on the fusion mechanism used in the multimodal model:

For **Summation fusion** and **Concatenation fusion**, individual modality logits, $z_m$ ($m \in M$), can be directly computed from the model's output, as formulated in Eq. 2.4, Eq. 2.6, and Eq. 2.8. These equations enable direct performance assessment of each modality without requiring additional modifications to the model.

For **Gated fusion** and **FiLM fusion**, individual modality contributions are more entangled. Thus, we use independent *probes* (helper classifiers) to measure per-

modality performance. The results from these probes serve as an estimate of each modality's effectiveness in contributing to the final prediction.

## 2.3 Mathematical Perspective on the Modality Imbalance Problem

To better understand the cause of Modality Imbalance in multimodal learning, we analyse the impact of the cross-entropy loss and its interaction with different modalities. Starting with a simple structure, we focus on the Late-Sum Fusion architecture, a straightforward late fusion approach, as shown in Figure 2.4, and later extend the explanation to other fusion mechanisms. In all examples, for simplicity, we consider a scenario with two modalities, $a$ and $b$.

### 2.3.1 Cross-Entropy Loss and Gradient Behaviour

The cross-entropy loss for a classification problem with softmax output can be defined as:

$$L = -\sum_i y_i \log(\hat{y_i}) \tag{2.12}$$

where $y_i$ is the one-hot encoded ground truth label, and $\hat{y}_i$ is the predicted probability for class $i$. The gradient of the loss with respect to the logit $z_i$ (before Softmax) is given by:

$$\frac{\partial L}{\partial z_i} = \hat{y}_i - y_i \tag{2.13}$$

When $\hat{y}_i$ approaches 1, the gradient approaches 0, leading to vanishing updates.

### 2.3.2 Gradient Propagation in Sum Fusion

In Late-Sum fusion, shown in Equation 2.7 and illustrated in Figure 2.4, the gradient of the loss with respect to the fused output $z_c$ is:

$$\frac{\partial L}{\partial z_c} = \hat{y} - y \tag{2.14}$$

By the chain rule, the gradients of $z_a$ and $z_b$ are:

$$\frac{\partial L}{\partial z_a} = \frac{\partial L}{\partial z_c} \cdot \frac{\partial z_c}{\partial z_a} = \hat{y} - y \tag{2.15}$$

$$\frac{\partial L}{\partial z_b} = \frac{\partial L}{\partial z_c} \cdot \frac{\partial z_c}{\partial z_b} = \hat{y} - y \tag{2.16}$$

This means that both modalities receive the same gradient update.

### 2.3.3 Gradient Conflict and Disappearance

The first issue is **gradient conflict**, a problem previously analysed in PMR [9]. This occurs when $z_a$ and $z_b$ produce divergent high-confidence predictions, leading to conflicting gradient directions. Such misalignment can cause gradient interference, potentially slowing down convergence or even destabilising the learning process. The problem becomes more pronounced when one modality learns at a faster rate than the other, allowing its updates to dominate and further disrupting the optimisation of the slower modality.

The second issue, which is another crucial cause of modality imbalance, is what we refer to as **gradient disappearance**. This phenomenon occurs when all modalities enhance their individual prediction accuracy, leading to a more accurate joint prediction. As $z_c$ converges toward a state where $\hat{y} \to 1$, the loss gradient diminishes, i.e., $\frac{\partial L}{\partial z_c} \to 0$. As a result, all modalities cease learning before achieving their optimal individual representations. This premature stagnation prevents further refinement of their features, ultimately leading to suboptimal solutions. A real example of this phenomenon is illustrated in Figures 2.9, 2.10, 2.11 for a single sample, while the overall training loss of the CREMA-D dataset is shown in Figure 2.12. In these figures, we observe that the fused prediction is dominated by the audio modality, demonstrated by the closely aligned audio and fusion curves in Figure 2.9's left plot and heatmaps of Figure 2.10. From Figure 2.11, we observe that under individual loss optimisation, gradient values are significantly stronger. For the audio modality, gradients diminish at a similar time step in both individual and joint optimisation settings. However, for the visual modality, gradients remain meaningful throughout the entire training process when optimised individually, in contrast to the rapid decline seen during joint optimisation. This behaviour is further illustrated in the right plot of Figure 2.9, which shows the gradient values at a fixed epoch. Consequently, with joint modality optimisation, as seen in Figure 2.12, the visual accuracy and visual loss stagnate prematurely around epoch 20.

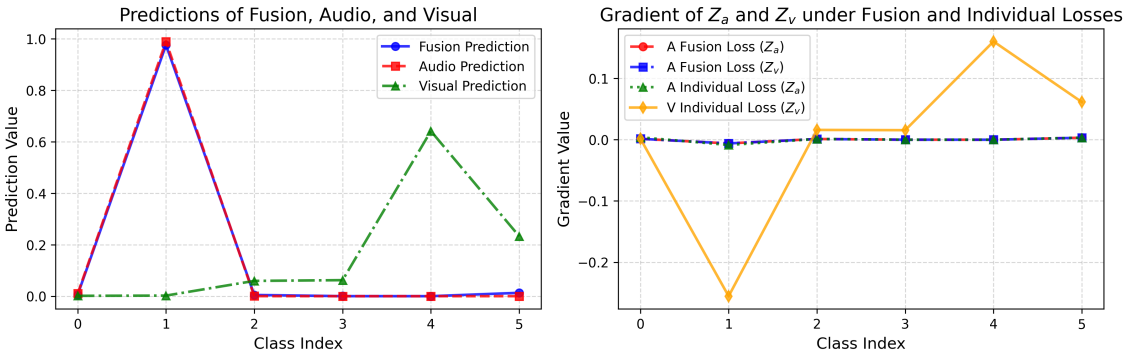



Figure 2.9: Details of a sample from the CREMA-D training set at epoch 24. (Left) Predictions from the fusion, audio, and visual outputs. (Right) Gradients of $Z_a$ and $Z_v$ with respect to the fusion loss (joint optimisation) and the individual modality losses (modality-specific optimisation).

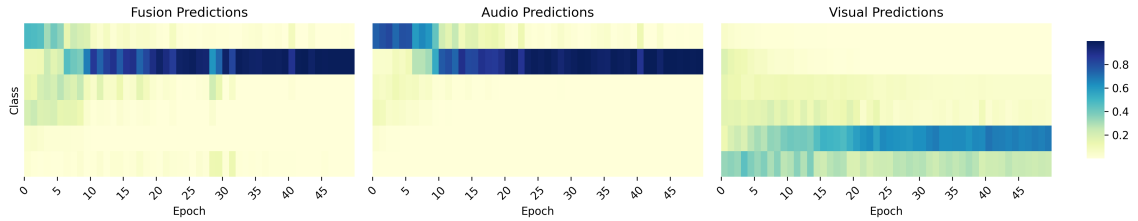



Figure 2.10: Prediction heat maps for the fused and unimodal (audio and visual) cases, showing class confidence over epochs 0 to 50 for the same sample shown in Figure 2.9.

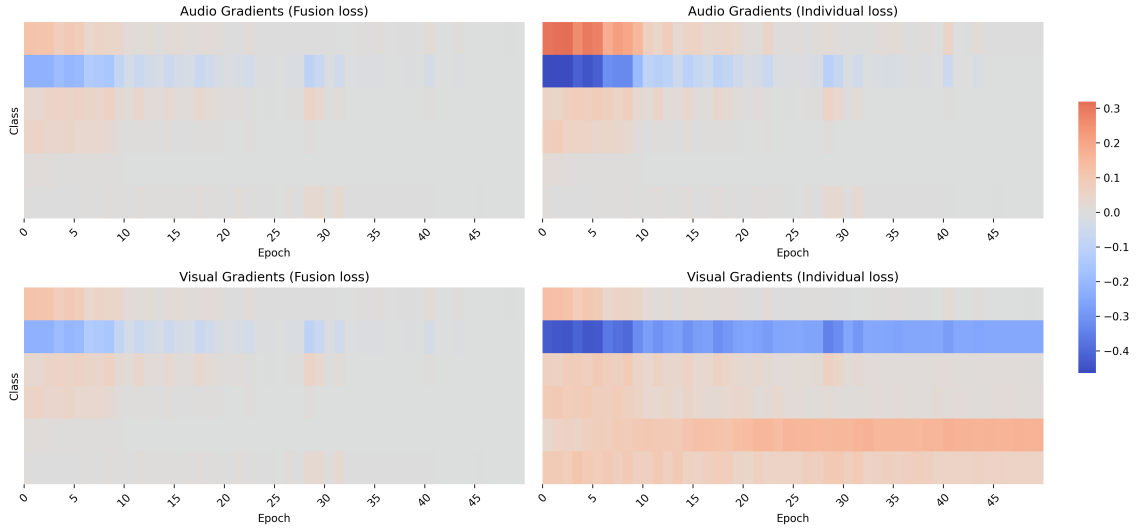



Figure 2.11: Gradient heatmaps of the same sample as in Figure 2.9. With joint loss optimisation (left), gradients diminish before epoch 20 and nearly vanish, as the audio prediction becomes overly confident. Individual modality loss optimisation (right) maintains stronger gradients throughout training.

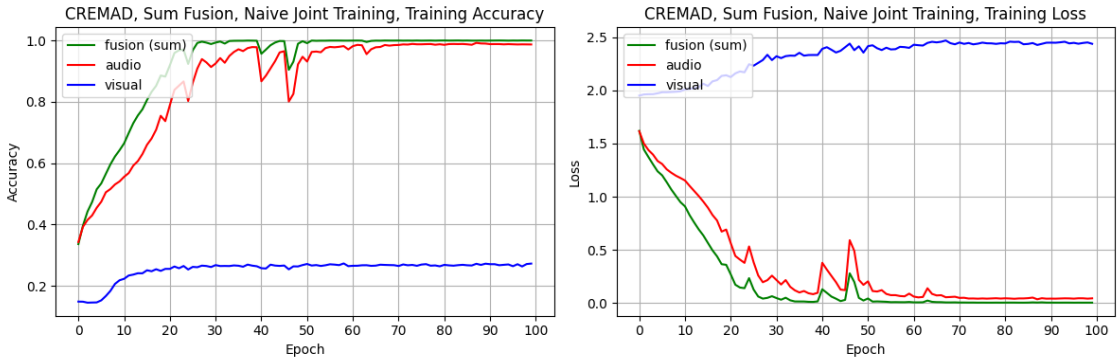



Figure 2.12: Empirical Case: Loss behavior in conventional joint training on the CREMA-D dataset. The fusion loss approaches 0, preventing the visual modality from learning effectively.

### 2.3.4 Extending to other fusion mechanisms

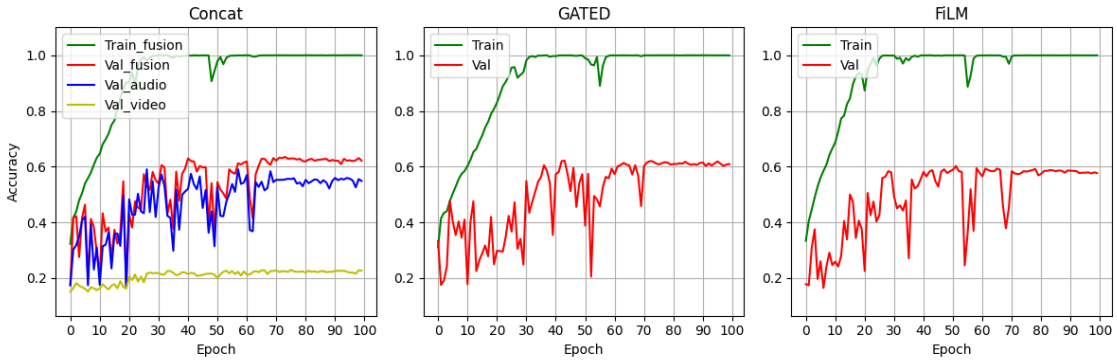



Figure 2.13: Concat, Gated, Film fusion trained with naive joint method (training without any imbalance solving technique), on CREMA-D, accuracy on training set and validation set.

#### Gradient Propagation in Concatenation Fusion

For concatenation fusion, the fused output (Eq. 2.9) is expressed as:

$$z_c = FC_{\text{out}}([e_a; e_b]) = W[e_a; e_b] + b \tag{2.17}$$

where $W$ is the weight matrix and $b$ is the bias term. The weight matrix $W$ can be partitioned into two submatrices, $W_a$ and $W_b$, corresponding to the contributions of $e_a$ and $e_b$, respectively. Thus, just as in Equations 2.5 and 2.9, the fused output can be reformulated as:

$$z_c = W_a e_a + W_b e_b + b = z_a + z_b + c \tag{2.18}$$

Therefore, concatenation fusion effectively reduces to sum fusion, inheriting the same issues associated with gradient conflict and disappearance.

#### Gradient Propagation in Gated Fusion

For gated fusion, when using $e_a$ as the gate input, let $g = \sigma(FC_a(e_a))$, $z_a = FC_a(e_a)$ and $z_b = FC_b(e_b)$. so that the fused output (Eq. 2.10) becomes:

$$z_c = FC_{\text{out}}(g \odot z_b). \tag{2.19}$$

Assuming $\frac{\partial L}{\partial z_c}$ is the gradient of the loss $L$ with respect to $z_c$, the gradient propagates as:

$$\frac{\partial L}{\partial (g \odot z_b)} = W_{\text{out}}^T \frac{\partial L}{\partial z_c}, \tag{2.20}$$

yielding

$$\begin{aligned} \frac{\partial L}{\partial z_b} &= \left( W_{\text{out}}^T \frac{\partial L}{\partial z_c} \right) \odot g, \\ \frac{\partial L}{\partial z_a} &= \left( W_{\text{out}}^T \frac{\partial L}{\partial z_c} \right) \odot \Big( z_b \odot g(1-g) \Big). \end{aligned} \tag{2.21}$$

with $g(1-g)$ being the derivative of the sigmoid function.

However, gradient vanishing may still occur as the loss approaches zero, preventing further updates as displayed by Figure 2.13's leftmost plot. In this plot, the visual modality's performance stagnates just as the training accuracy nears a perfect 100% accuracy score.

**Gradient Propagation in FiLM Fusion**

For FiLM fusion, as defined in Equation 2.11, using the chain rule, the gradients with respect to $e_a$ and $e_b$ are:

$$\frac{\partial L}{\partial e_b} = \frac{\partial L}{\partial z_c} \odot \gamma(e_a), \tag{2.22}$$

$$\frac{\partial L}{\partial e_a} = \frac{\partial L}{\partial z_c} \odot \left(e_b \odot \gamma'(e_a) + \beta'(e_a)\right), \tag{2.23}$$

where $\gamma'(e_a)$ and $\beta'(e_a)$ denote the derivatives of $\gamma(e_a)$ and $\beta(e_a)$ with respect to $e_a$.

Thus, similar to sum fusion, FiLM fusion is also susceptible to gradient vanishing when the loss approaches zero, which hinders further model updates during training.

### 2.3.5 Conclusion

From the analysis and real-world cases presented above, we hypothesise that two key factors are essential for addressing modality imbalance: (1) ensuring that all modalities learn at a similar pace, and (2) preventing overfitting to mitigate gradient vanishing. These insights provide a foundation for developing more effective multimodal learning strategies. Figure 2.14 presents additional examples where the performance of individual modalities in a multimodal setting is consistently lower than in their respective unimodal settings.

## 2.4 Limitations of Unimodal Optimisation Techniques in a Multimodal Context

As observed in real-world cases and discussed in [5], overfitting is a major factor contributing to modality imbalance. While many techniques have been developed to address overfitting in single-modality model training, their effectiveness in multimodal settings has not met expectations. This section examines the limitations of these commonly used techniques.

### 2.4.1 Data augmentation

Data augmentation [32] is a common technique to reduce overfitting by increasing the diversity of training data. It works by applying transformations such as cropping, flipping, noise injection, and time-warping, helping models generalise better. This

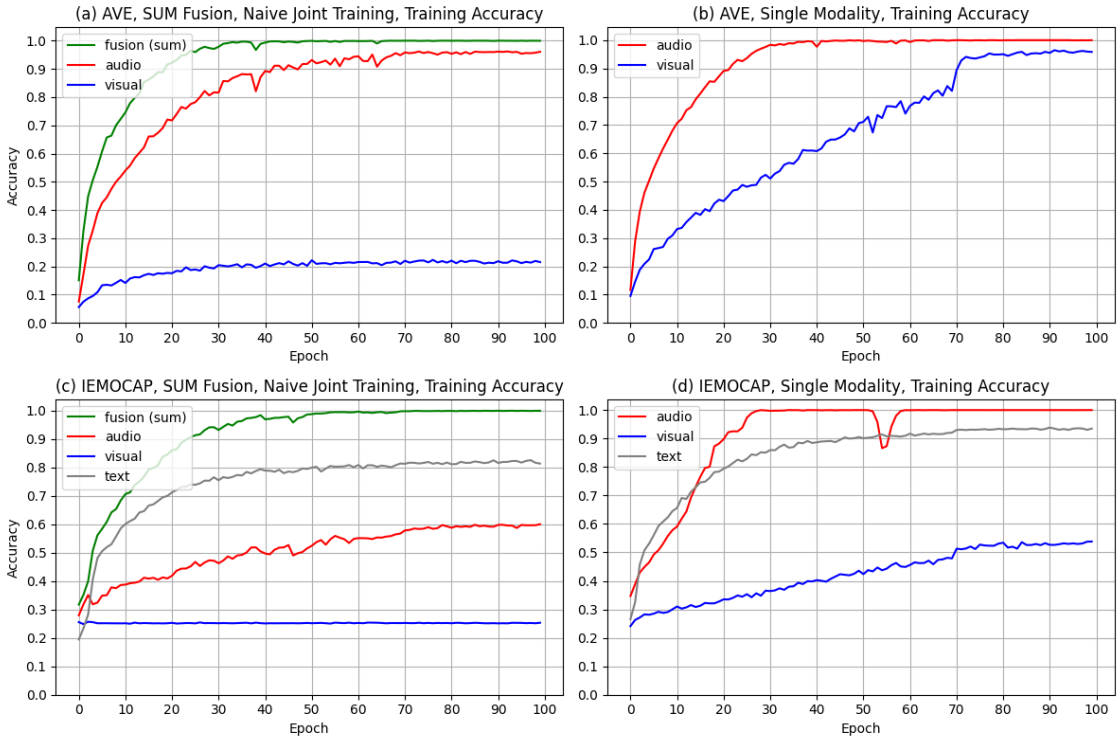



Figure 2.14: Naive joint method and Single modality performance, on AVE IEMOCAP - Training Accuracy.

approach is especially useful for small datasets and has been widely applied in image-related tasks.

In multimodal learning, especially for audio, similar augmentation methods like time stretching, pitch shifting, and background noise injection [33] can be used. However, we find that previous works (e.g., OGM-GE, AGM, MLA) largely ignored data augmentation for audio, while applying it only to the visual modality. This imbalance likely contributes to the overfitting observed in Figure 2.12, where the audio modality is particularly affected.

To test this, we applied audio augmentation to the CREMA-D dataset using the following transformations:

$$\mathbf{x}_{\text{aug}} = \mathcal{T}(\mathcal{F}(\mathbf{x})) + \mathcal{N}(0, 0.02^2) \tag{2.24}$$

where

- $\mathbf{x}$ is the original audio sample.
- $\mathcal{T}(\mathbf{x})$ represents the time masking operation, which randomly selects a time segment in the input and sets it to zero. This simulates temporal variations in the audio signal.

- $\mathcal{F}(\mathbf{x})$ represents the frequency masking operation, which randomly selects a portion of the frequency spectrum and sets it to zero. This simulates variations in the frequency domain.

- $\mathcal{N}(0, 0.02^2)$ represents adding Gaussian noise with mean 0 and standard deviation 0.02, which helps increase the variability of the data and prevents overfitting.

We found that, while not improving single-modality performance (Figure 2.15 a), by just slowing down the learning speed of the audio modality, the joint training results were significantly boosted (Figure 2.15 b, c). However, when we did the same for AVE, even though the audio modality improved about 10 accuracy percentage points (Figure 2.16 a), joint training performance did not improve as expected, and both modalities in the multimodal model were still suppressed (Figure 2.16 b, c). This suggests that data augmentation is helpful but not a universal fix.

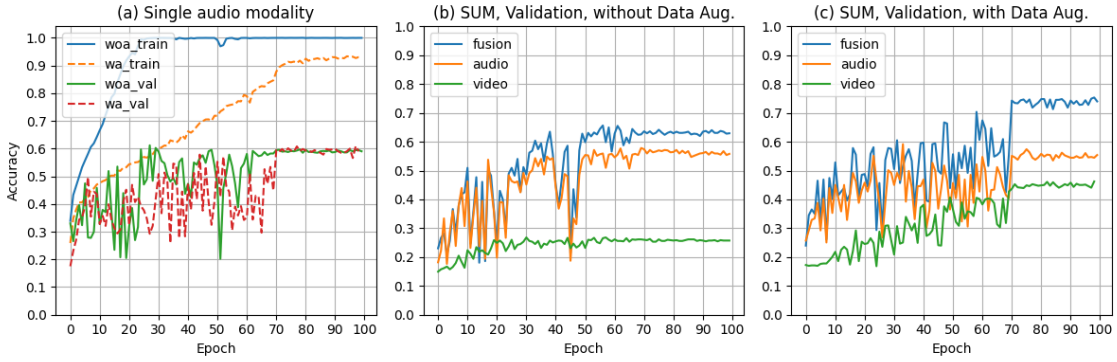



Figure 2.15: Accuracy of single modal (audio) model (a), with (*wa*) and without (*woa*) data augmentation, and multimodal model with sum fusion on CREMAD dataset without (b) and with (c) Audio Augmentation

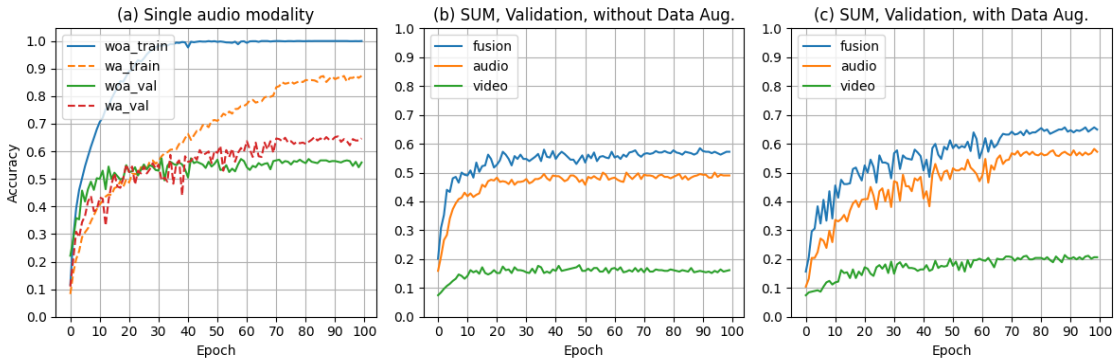



Figure 2.16: Accuracy of single modal (audio) model (a), with (*wa*) and without (*woa*) data augmentation, and multimodal model with sum fusion on AVE dataset without (b) and with (c) Audio Augmentation

Additionally, in more complex tasks, such as medical data or text-based modalities, data augmentation is difficult to apply effectively. Unlike images or audio, modifying

text while keeping its meaning intact is challenging. Similarly, in medical datasets, applying transformations while preserving clinical relevance is non-trivial.

### 2.4.2 Dropout

Dropout [34] is another effective technique for mitigating overfitting during the training phase and has been widely validated across various neural network architectures. Compared to data augmentation, Dropout is simpler to implement while still providing significant regularisation benefits.

Dropout layers are also widely used in multimodal models. However, similar to data augmentation, previous studies have largely overlooked the specific effects of this technique on modality imbalance. While Dropout helps suppress overfitting and enhances generalisation, selecting an appropriate dropout rate remains a challenge. To investigate its impact, we applied Dropout to the audio embeddings $e_a$ extracted from the encoder (Figure 2.4), testing different dropout rates $p$, as:

$$\hat{e}_a = e_a \cdot r_a \cdot \frac{1}{1-p} \tag{2.25}$$

where $r_a$ is a binary random vector, with $r_{a,i} \in \{0, 1\}$ indicating whether the element is dropped (0) or kept (1). And $\frac{1}{1-p}$ is the scaling factor to maintain consistency between training and inference.

The results, presented in Figure 2.17 and Table 2.1, demonstrate that, similar to data augmentation, Dropout helps mitigate modality imbalance but does not completely resolve the issue.

Table 2.1: Performance of different dropout rates on CREMA-D, Late-Sum Fusion, Test set. The single-modal model performance is shown in the first row. And $p = 0.0$ represents joint training without the dropout layer.

| $p$ | **Fusion** | **Audio** | **Visual** |
|---|---|---|---|
| uni | - | 59.1 | 60.5 |
| 0.0 | 62.9 | 55.0 | 21.6 |
| 0.2 | 65.6 | 56.6 | 24.7 |
| 0.4 | 66.4 | 54.8 | 27.2 |
| 0.6 | 67.8 | 49.3 | 29.4 |
| 0.8 | 70.0 | 41.4 | 29.6 |

### 2.4.3 L2 Regularisation

L2 regularisation, also known as weight decay [35], is another popular technique used to mitigate overfitting in machine learning models. Unlike Dropout, which randomly drops units during training, L2 regularisation works by penalising large weights, effectively discouraging the model from becoming overly complex and thus

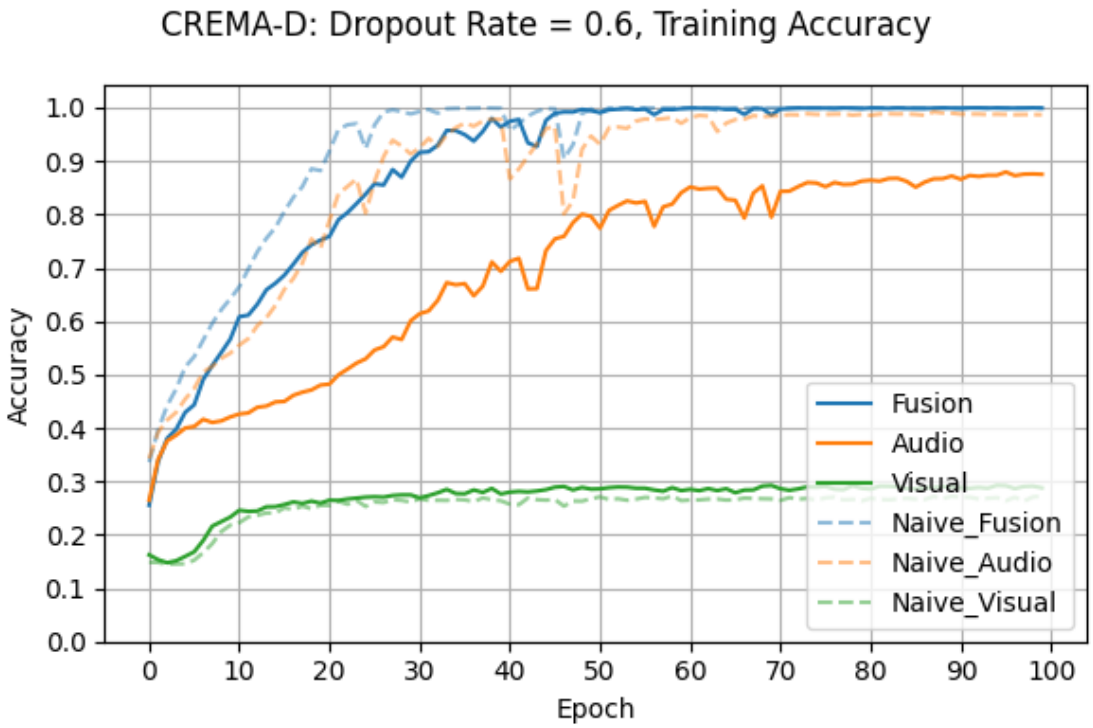



Figure 2.17: Applying Dropout on audio modality as defined in Equation 2.25. The curves represent the training accuracy of late-sum fusion on CREMA-D, and the faded curves indicate the accuracy without dropout.

preventing overfitting. This technique is commonly implemented by adding a penalty term to the loss function, proportional to the square of the L2 norm of the weights:

$$L_{\text{reg}} = \lambda \sum_i w_i^2 \tag{2.26}$$

where $w_i$ represents the individual weights of the model, and $\lambda$ is a hyperparameter that controls the strength of regularisation. By tuning $\lambda$, one can control the trade-off between fitting the data well and keeping the model simple.

However, similar to Dropout, L2 regularisation alone does not address the specific challenges posed by modality imbalance in multimodal training.

To explore the effects of L2 regularisation, we applied it to the model during training and compared different values of $\lambda$ to examine how it impacts the training and test performance across different modalities. Figure 2.18 and Table 2.2 present the results of our experiments on the CREMA-D dataset.

The results indicate that L2 regularisation helps slightly improve generalisation performance, particularly when the model is at risk of overfitting.

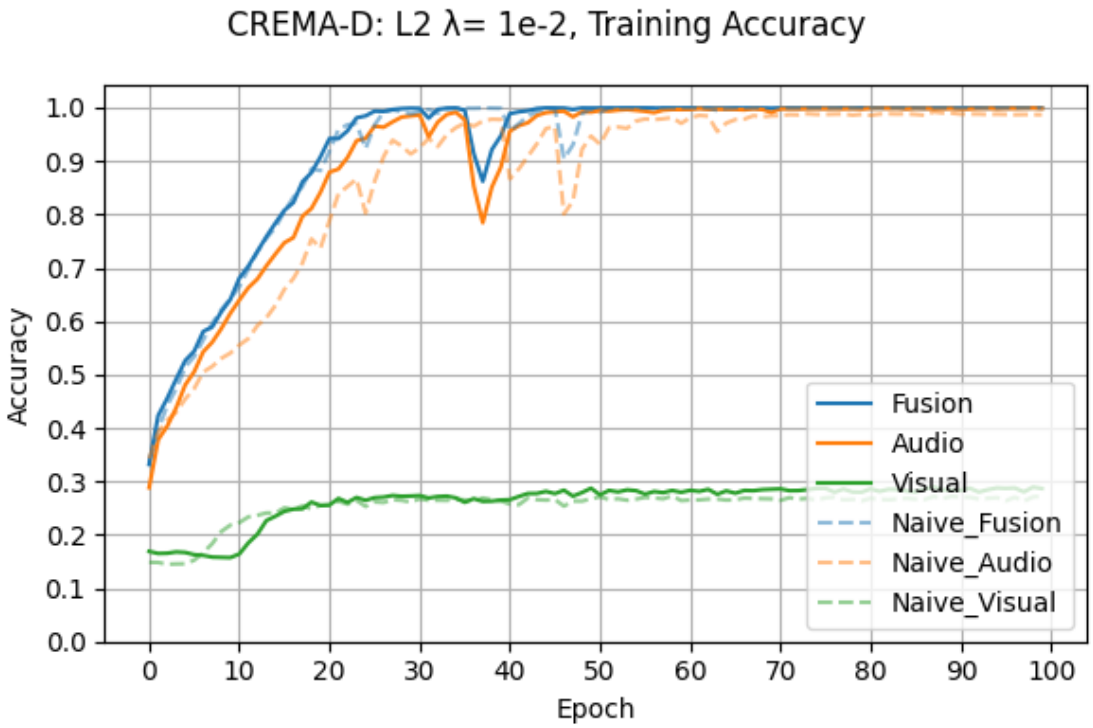



Figure 2.18: Effect of L2 regularisation on training accuracy for late-sum fusion on CREMA-D. The curves represent training accuracy with L2 regularisation (1e-2), while the faded curves indicate accuracy with the lowest L2 regularisation (1e-4, the default value).

## 2.5 Review of SOTA Methods

In the previous section, we analysed the causes of the modality imbalance and demonstrated that optimisation techniques designed for single-modality models do not effectively address this issue. In this section, we review recent state-of-the-art (SOTA) methods that aim to solve this problem. These methods are primarily based on three key ideas: (1) modulating the learning speed, either by dynamically adjusting the learning rate or by modulating the gradient of the faster modality; (2) introducing individual loss terms or auxiliary mechanisms to ensure that both encoders continue learning while mitigating modality conflicts and loss disappearance; and (3) employing new model structures and training methods to address the challenges posed by modality imbalance.

### 2.5.1 Modulating Learning Speed

Methods like OGM-GE mitigate the faster-learning modality's dominance by modulating its gradient:

$$\hat{g} = g \cdot (1 - \tanh(\alpha\rho)) \tag{2.27}$$

where $\alpha$ is a hyper-parameter, and $\rho$ represents the accuracy ratio between the two modalities. Similarly, MSLR adjusts the learning rate of each modality based on

Table 2.2: Performance of different L2 regularisation $\lambda$ on CREMA-D, Late-Sum Fusion, Test set. The single-modal model performance is shown in the first row. The row with $\lambda = 1e-4$ is the default value used in all experiments in this study.

| $\lambda$ | **Fusion** | **Audio** | **Visual** |
|---|---|---|---|
| - | - | 59.1 | 60.5 |
| 1e-4 | 62.9 | 55.0 | 21.6 |
| 1e-3 | 62.5 | 56.5 | 22.9 |
| 1e-2 | 63.9 | 58.0 | 24.3 |
| 1e-1 | 66.8 | 53.2 | 16.2 |

validation accuracy:

$$\hat{lr_m} = lr_m \times \frac{acc_t^m}{E[acc_{t-N,t}^m]} \tag{2.28}$$

However, as shown in Figure 2.19, these approaches still converge to suboptimal solutions.

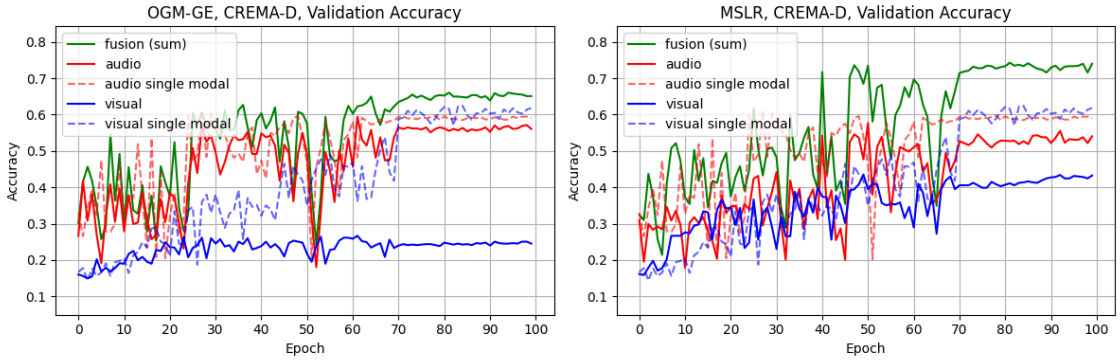



Figure 2.19: Applied OGM-GE and MSLR methods to the CREMA-D dataset, Late-Sum Fusion. For OGM-GE, $\alpha = 0.1$ as specified in the original paper, with a learning rate of $1 \times 10^{-3}$. For MSLR, the learning rates were set to $1 \times 10^{-3}$ for the audio modality and $1 \times 10^{-2}$ for the visual modality, following the paper and selecting values based on unimodal fine-tuning.

### 2.5.2 Introducing Independent Loss Terms

A promising alternative is introducing independent loss terms for each modality to maintain balanced learning. For instance, Gradient Blending (GB) [5], demonstrated in Figure 2.20, blends fusion loss with individual modality losses:

$$L = w_1 L_{multi} + w_2 L_{m_1} + w_3 L_{m_2} \tag{2.29}$$

Similarly, PMR (Figure 2.21 ) employs a combination of prototypical loss and entropy regularisation:

$$L = L_{sum} + L_{PCE} - L_r \tag{2.30}$$

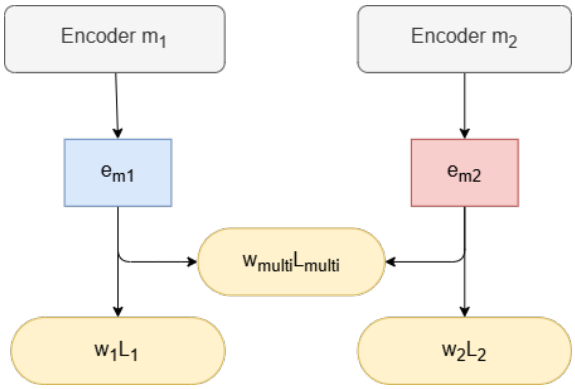



Figure 2.20: Gradient Blending method: Training encoders with a helper classifier. $e_{m_i}$ represents the feature embeddings from the encoders (blue and red boxes), while the yellow boxes indicate classifiers applied to these features. $L_{multi}$ denotes the fusion loss, and $L_i$ represents the helper classifier loss.

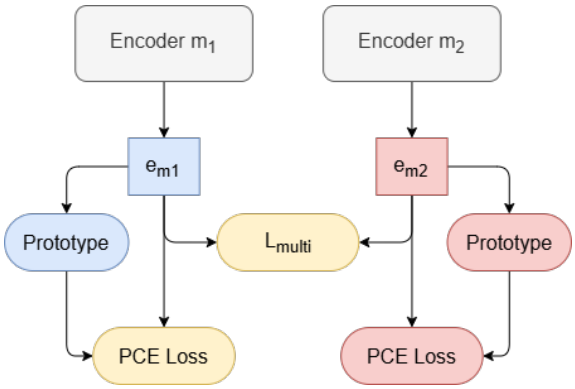



Figure 2.21: PMR method (simplified, ignoring the PER component): Training encoders with PCE loss, which is computed based on the distance to the prototype of each class.

These approaches mitigate premature stagnation in encoders. The helper functions in the Gradient Blending method and the prototypes in PMR work similarly to probes, as described in Section 2.2. Introducing new loss terms improves encoder performance, producing more meaningful embeddings. However, the fusion layers still suffer from the same underlying issues, as evidenced by Figure 2.22. Moreover, the individual modality accuracies still fall short of the performance achieved by the training helper methods represented by the dashed lines.

### 2.5.3 Alternating Training

As observed above, even with the introduction of individual loss terms, the fusion layer may still suffer from modality imbalance. To address gradient conflicts while maintaining effective information exchange between modalities, MLA adopts a staged training approach with a new fusion structure. It separates the training process into distinct phases and divides the multimodal model into two components: (1) encoders and (2) a shared head, as illustrated in Figure 2.23.

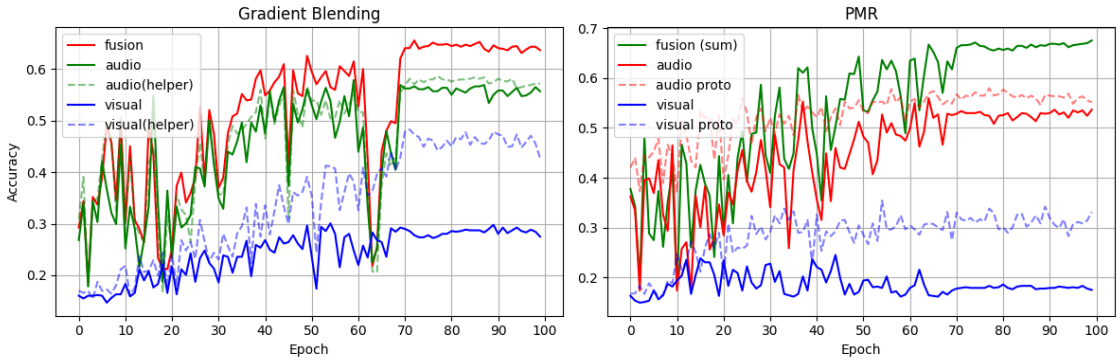



Figure 2.22: Applying the Gradient Blending and PMR methods to the CREMA-D dataset with SUM fusion. Both methods enhance fusion performance. However, despite improvements in encoder performance (indicated by helper accuracy for both modalities in the Gradient Blending method and prototype accuracy for both modalities in the PMR method, shown as faint dashed lines), the fusion layer still exhibits modality imbalance.

$$out = \sum_{m}^{M} H_{share}(e_m) = \sum_{m}^{M} W[e_m] + b \tag{2.31}$$

where the shared head $H_{share}$ is a simple linear layer, $W$ represents its weight parameters and $b$ is the bias term.

### Runtime Fusion

In MLA, for runtime fusion, two methods are proposed to compute the weight of each modality. The simpler method is **Fixed-weight fusion**, where the weight of each modality, $w_m$, is a user-defined hyperparameter that requires fine-tuning.

The second method is a dynamic approach based on the entropy of the predictions as defined in Equation 2.32. For each sample $s$ and each modality $m$ in the modality set $M$, the entropy $e_{s,m}$ is computed as:

$$e_{s,m} = -p_{s,m}^{T} \log(p_{s,m}) \tag{2.32}$$

Then, the weight $w_{s,m}$ is determined using the following formula:

$$w_{s,m} = \frac{\exp\left(\max_{j \in M} e_{s,j} - e_{s,m}\right)}{\sum_{k \in M} \exp\left(\max_{j \in M} e_{s,j} - e_{s,k}\right)} \tag{2.33}$$

This formulation ensures that modalities with lower entropy (i.e., more confident predictions) are assigned higher weights, while those with higher entropy (less confident) contribute less to the fusion process.

Despite not explicitly mentioning the existence of the gradient disappearance problem, MLA effectively addresses it by introducing modality-specific optimisation while

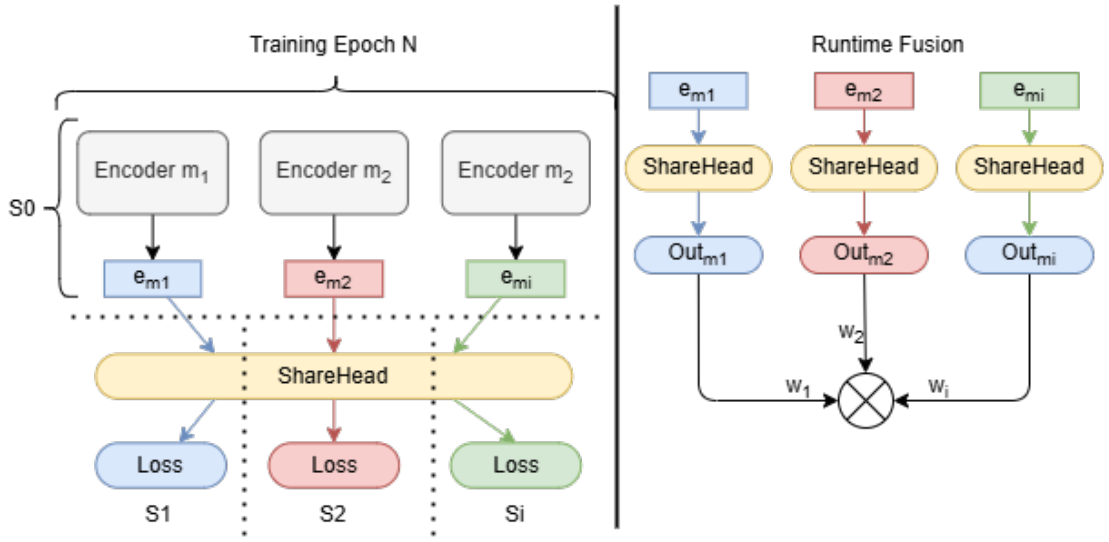



Figure 2.23: MLA training process within an epoch and runtime fusion. The yellow box represents the shared head, which is utilised by all modalities. During training, each epoch is divided into $i + 1$ steps, where $i$ is the number of modalities. In step 0, all encoders generate feature embeddings $e_{m_i}$. Then, in step $i$, the shared head is applied to $e_{m_i}$, the corresponding loss $L_i$ is computed, and backpropagation is performed to update both encoder$_{m_i}$ and the shared head.

using a shared classifier, significantly improving performance. This solution arises from optimising modality-specific losses rather than relying on a single joint loss, which tends to vanish as the combined modalities approach near-perfect performance. A real case using the CREMA-D dataset is shown in Figure 2.24 (a, b).

## 2.6 Problems Unsolved in Previous Works

### 2.6.1 Modality Prediction Bias

On one hand, MLA aims to entirely tackle the modality imbalance problem during the training by overcoming the gradient disappearance phenomenon. On the other hand, during the prediction phase, as shown in Figure 2.24 (b), we can notice that although both modalities have similar performance (shown by audio and visual curves), to achieve the best possible results, the runtime fusion must rely more on the visual modality, rather than the audio one. The curve $f_{55}$ displays the accuracy resulting from assigning an even contribution to both modalities for prediction, and curve $f_{37}$ demonstrates the performance of assigning 70% contribution to the visual modality and 30% to the audio one. As illustrated in the figure, the curve $f_{37}$ exceeds the curve $f_{55}$, showing that, despite modalities having similar uni-modal performances, assigning higher weight to the visual modality improves the overall prediction. In addition, the entropy-based fusion mechanism, represented by curve $f_{entropy}$, displays a similar behaviour to the $f_{55}$ curve, underperforming the $f_{37}$ one. From Figure 2.24 (a,c,d), in (a), we observe that the audio modality reaches an overfitting state much earlier than the visual modality, where continuous training results in an extremely

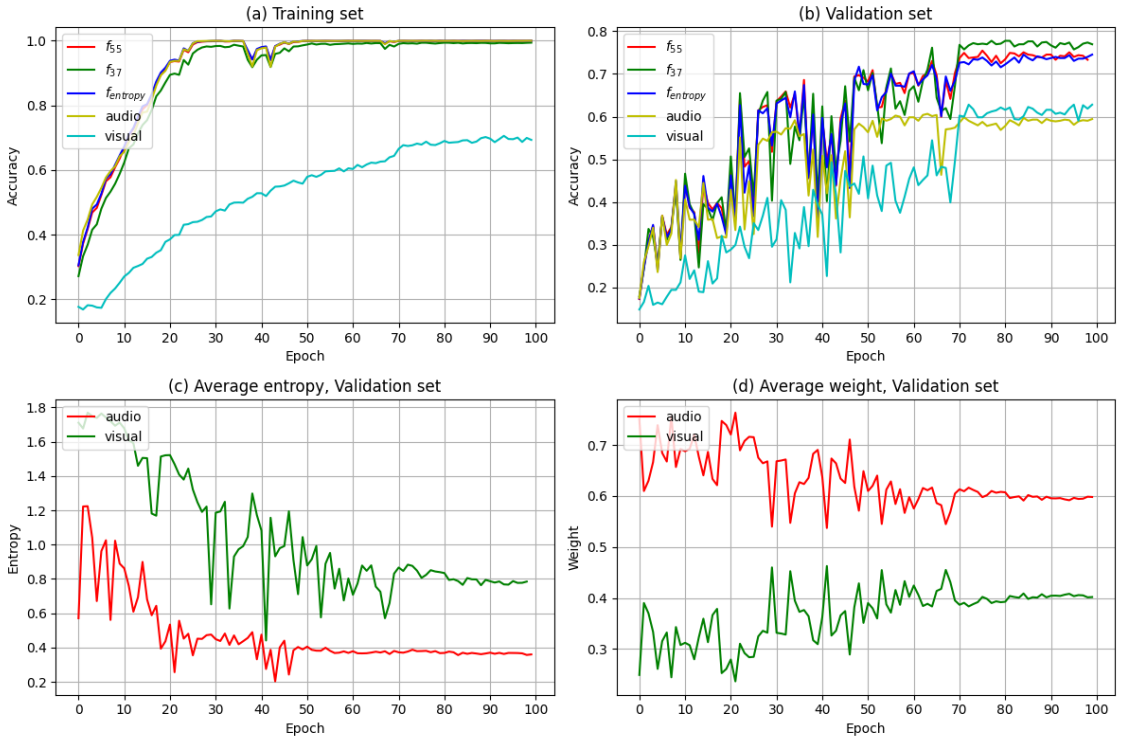



Figure 2.24: MLA method. In (a,b), $f_{55}$ is the fixed weight fusion with weights of 0.5 for both the audio and visual modalities. $f_{entropy}$ is the entropy-based weight fusion; $f_{37}$ corresponds to assigning a weight of 0.3 to the audio modality and 0.7 to the visual one, and yields the best result. (c) shows the average entropy computed with Equation 2.32 for both modalities on the test set. (d) shows the average weight computed with entropy, as defined in Equation 2.33, for both modalities on the test set.

low entropy for this modality, as shown in (c). Consequently, the lower entropy for audio increases the model's confidence in this modality, resulting in a higher weight assigned to it according to the dynamic entropy-based fusion method. This weight distribution illustrated in (d), where audio is assigned 60% weight and visual 40%, does not follow the optimal one, where audio should be assigned 30% contribution and visual 70%, as shown in curve $f_{37}$ of (b). Therefore, we introduce the problem of uneven modality prediction contribution in this work as **Modality Prediction Bias**.

#### 2.6.1.1 Modality Logit Magnitude

Regardless of which training architecture and fusion mechanism are employed in a multimodal model, in classification tasks, the model assigns a probability for each class. The final predicted class is obtained from extracting the class that displays the highest probability among all classes. These probabilities are, in turn, obtained by applying the softmax function (2.34), to the logit scores generated by the classification layer.

$$p_i = \frac{\exp(logits_i)}{\sum_{j=1}^{N} \exp(logits_j)} \tag{2.34}$$

Depending on which fusion mechanism is adopted, how to obtain these logit scores may vary. Taking MLA's fixed fusion mechanism with equal contribution, applied to two modalities as an example (2.31), the final fused logits ($logits_f$) obtained before computing the softmax operation are given by the following expressions, (2.35):

$$\begin{aligned} logits_1 &= W\, e_1 + b, \\ logits_2 &= W\, e_2 + b, \\ logits_f &= logits_1 + logits_2 - b \end{aligned} \tag{2.35}$$

From the previous expressions, we conclude that the final fused logits ($logits_f$) directly depend on the modality-specific logits ($logits_1$ and $logits_2$). Consequently, differently scaled logits will have a totally different impact on the final prediction, with higher-magnitude modality logits dominating the expression.

#### 2.6.1.2 Logit Magnitude and Entropy Behaviour in Training

As previously described, if the magnitude of the logits of modalities produced by the model differ, larger magnitude logits will have a higher contribution towards the final prediction. While a higher contribution from certain modalities may be advantageous in specific scenarios, this is not universally beneficial in all cases. To formalise how modality confidence evolves, we consider the entropy $H$ of the model's predictive distribution. Entropy measures the uncertainty of a probability distribution $p$, where:

$$H(p) = -\sum_{i=1}^{N} p_i \log p_i, \tag{2.36}$$

and $p_i$ is the predicted probability for class $i$. A uniform distribution over $N$ classes yields maximum entropy, indicating complete uncertainty (i.e., no single class is favoured). In contrast, assigning nearly all probability mass to a single class yields minimum entropy, indicating strong confidence in that class. When using Cross-Entropy Loss to train a classifier, the goal is to maximise the predicted probability of the correct class. Formally, we minimise:

$$\text{CrossEntropyLoss} \;=\; -\sum_{i=1}^{N} y_i \, \log p_i, \tag{2.37}$$

where $y_i$ is the ground-truth label and $p_i$ is given by (2.34).

During training, as the Cross-Entropy Loss is minimised, the model continually increases the correct class' logit and decreases the incorrect class' logits. Over

time, this process spreads the coordinates of the logit vector, where one becomes increasingly large (positive), while the others become more negative. Since the Euclidean norm

$$\|\mathbf{z}\|_2 = \sqrt{\sum_{i=1}^{N}(\text{logits}_i)^2} \tag{2.38}$$

grows whenever at least one coordinate's absolute value increases, raising the correct logit and lowering the incorrect logits inflates $\|\mathbf{z}\|_2$. Therefore, the norm of the logits increases throughout training as the model grows more confident in the correct class and suppresses the wrong classes.

If a modality's parameters are updated efficiently, that modality's logits for the correct class will grow faster, and those for the incorrect classes will decrease more sharply. This not only increases the overall norm of its logit vector but also decreases its predictive entropy at a faster rate, signalling higher confidence. By contrast, slower-learning modalities experience a more gradual growth in logit magnitudes and hence retain higher entropy for a longer portion of the training.

Figure 2.25 shows how the logit magnitudes (a) and entropy curves (b) for the different modalities evolve as training progresses in the CREMA-D dataset. From Figure 2.24 (a), we know that the fastest learning modality is audio. Figure 2.25 (b) reveals that faster-learning modalities (audio) display a more rapid decrease in entropy, due to their faster parameter convergence. The result is disproportionately large logits for these modalities relative to those that converge more slowly, as displayed in Figure 2.25 (a). Not having balanced logits among modalities will directly introduce an inherent preference towards the higher-magnitude modalities, leading to the problem of **Modality Prediction Bias**.

However, despite faster learning modalities displaying faster logit magnitude growth, this does not imply that faster-learning modalities always produce larger logit magnitudes, as, depending on the modality-specific embeddings generated by the backbone encoders and fusion structure adopted, the scale of the logits will vary.

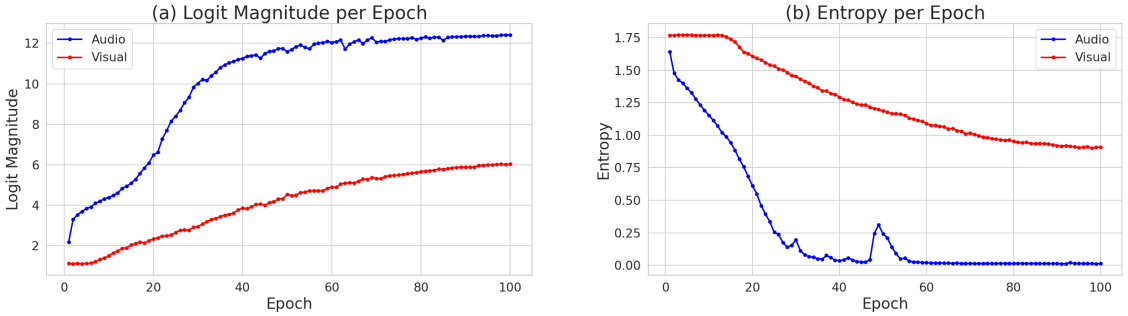



Figure 2.25: (a) Logit Magnitudes and (b) Entropy of Audio and Visual Modalities per epoch in the MLA approach.

### 2.6.2 Generalising Modality Prediction Bias to Different Fusion Strategies

As previously mentioned, modalities that display higher logit magnitudes will have a higher contribution towards the final prediction. If modality logits display different scales, an undesired bias towards certain modalities may be created. An initial thought about attempting to remove modality bias in prediction is to have all modality logits on the same exact scale. However, if not done carefully, altering the scale of each modality's logits impacts the model in different ways. During training, it will directly interfere with the computed Cross-Entropy Loss. This loss does not simply evaluate whether a prediction is right or wrong, but how far from the truth the prediction is, aiming to maximise the correct answer and minimise the wrong ones. Moreover, before computing the loss function, the softmax operation (2.34) is applied to the logits to obtain class probabilities. Since softmax involves an exponential transformation, differences in the scale of the logits can lead to disproportionately large differences in the resulting probabilities across classes. Therefore, altering the scale of the logits directly impacts the scale of the computed loss, which subsequently affects the magnitude of gradient updates and the convergence behaviour of the encoders during training. As a result, any modifications to logit scales must be designed with careful consideration of maintaining stable and effective gradient updates.

#### Defining Different Forms of Bias

Different fusion mechanisms, such as summation and concatenation, can be manually replicated by combining the different unimodal logits as shown in Eq. 2.9 and Eq. 2.7. From these expressions, we conclude that the final fused logits directly depend on the modality-specific embeddings and weight parameters in the classifier. When using linear classifiers in the form $\mathbf{z} = W^\top \mathbf{e} + b$, there are two main properties included in the logit vectors, the **scale** and the **direction**, as demonstrated in 2.39. For simplicity, we fix the bias parameter to 0. $c$ is a given class; $w_c$ the weights of the linear layer corresponding to that class; $e$ the input modality embeddings and $\theta_c$ the angle between $w_c$ and $e$.

$$logit_c = \mathbf{w}_c^\top \mathbf{e} = \underbrace{\|\mathbf{w}_c\|_2 \|\mathbf{e}\|_2}_{\text{scale}} \underbrace{\cos\theta_c}_{\text{direction}} \tag{2.39}$$

Consequently, two distinct forms of individual bias can arise from a modality's logits. First, the **scale** term, controlled by both the encoder's embedding norm and the classifier's weight norm, can let one modality numerically overwhelm the others. Second, the **direction** term, $\cos\theta_c$, captures how confidently that modality points towards or away from a class. Importantly, minimising cross-entropy amplifies any initial advantage in either scale or direction as gradients push the already-strongest logit upward while suppressing the rest. Thus, these two contributions constitute distinct biases that must be isolated and mitigated individually. Scale bias stems from heterogeneity in encoder architectures and regularisation schemes, which can

place each modality on a different norm regime. Directional bias, in contrast, evolves with the optimisation dynamics, where modalities whose parameters rotate towards the class weight vector more rapidly secure a larger angular margin earlier in training.

**Note:** This analysis is only viable for simple fusion mechanisms like Early-Sum, Late-Sum and Concatenation fusion that follow the linear classifier form. In contrast, it is not applicable to more complex fusion strategies such as Gated and FiLM fusion, which do not follow the linear classifier form.

### 2.6.3 The Need for a General Method

MLA fixes the issue of loss disappearance. However, not only does it not address the modality prediction bias issue, but it also changes the fusion structure. All fusion strategies like Summation, Concatenation, Gated and FiLM have their specific features and behave differently depending on the tasks and datasets. Therefore, there is a great need for a general method that is capable of being applied to different fusion mechanisms, bringing together both the benefits of using different fusion strategies and MLA's alternating training technique.

## 2.7 System Aware Optimisation for the Training Stage

During the research, we found that conventional Pytorch built-in optimisation techniques are not able to fully utilise the potential of our hardware, particularly with high performance GPUs.

Figure 2.26 and Algorithm 1 show a typical simplified process of training a model in a system with CPUs and GPUs. Basically, it can be divided into two main parts: **loading and processing data** and **forwarding and back-propagation**. These two parts are also the main parts that most optimising methods focus on [16] [21] [20] [18]. In the following sections, we first introduce the tools used for profiling. We then discuss the theoretical foundations of accelerating model training using low-precision floating-point formats. Finally, we review several built-in optimisation techniques in PyTorch and challenges overlooked in scenarios like our cases.

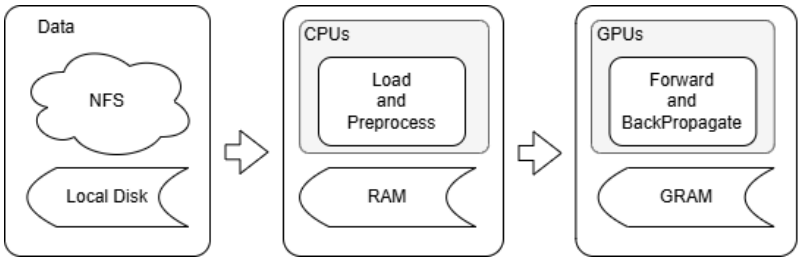



Figure 2.26: A typical training epoch: loading data - processing data - forward and update the model on GPU

**Algorithm 1** Simplified Model Training Process with CPU and GPU

**Input:** Data $\{X, Y\}$, model $\theta$, optimiser $\mathcal{O}$, loss function $\mathcal{L}$, Batches $B$
**for** b = 1 to $B$ **do**
  **CPU:** Load and process $\{x_b, y_b\}$
  **GPU:** Transfer $\{x_b, y_b\}$ to GPU memory
  **GPU:** Forward: Compute model output $y_{pred} = f(x_b; \theta)$
  **GPU:** Loss and Backward: $\mathcal{L}(y_{pred}, y_b) \rightarrow \nabla_\theta \mathcal{L}$
  **GPU:** Update $\theta \leftarrow \mathcal{O}(\theta, \nabla_\theta \mathcal{L})$
**end for**

### 2.7.1 Tools for Profiling Training Optimisation

Performance analysis is essential for understanding and optimising system efficiency. Given that our experiments are conducted on the platform **Alvis** [36] and with **PyTorch** [37], we employ the following profiling tools to analyse different aspects of system performance.

**PyTorch Profiler** - is an integrated profiling tool within the PyTorch framework, designed for detailed performance analysis of deep learning models. It enables tracking of **GPU utilisation**, **Memory consumption**, **Kernel execution time**, and **Data loading efficiency**. This tool helps identify potential bottlenecks, such as inefficient data pipelines or suboptimal GPU computation, as illustrated in Figure 2.27.

**Alvis Monitor** - While PyTorch Profiler focuses primarily on GPU performance and PyTorch runtime metrics, a broader system-level analysis is required to monitor **CPU utilisation**, **network activity**, **disk I/O**, and **power consumption**. To achieve this, we use the Alvis monitoring System, a built-in monitoring dashboard provided by the platform. This tool offers real-time insights into system-wide resource utilisation, helping to assess workload distribution and energy efficiency. Example metrics are illustrated in Figure 2.28.

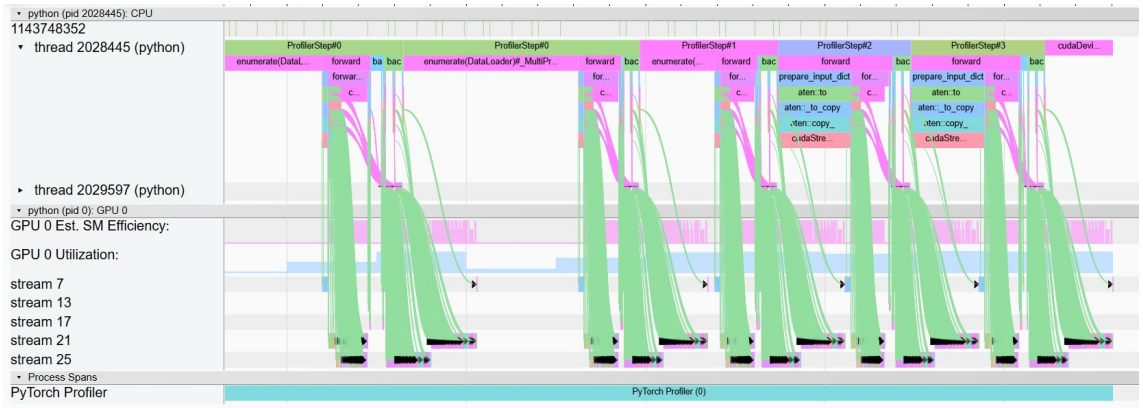



Figure 2.27: Example of Pytorch Profiler, showing the detail of GPU workload for the CREMA-D dataset, first epoch and the first 5 batch steps.

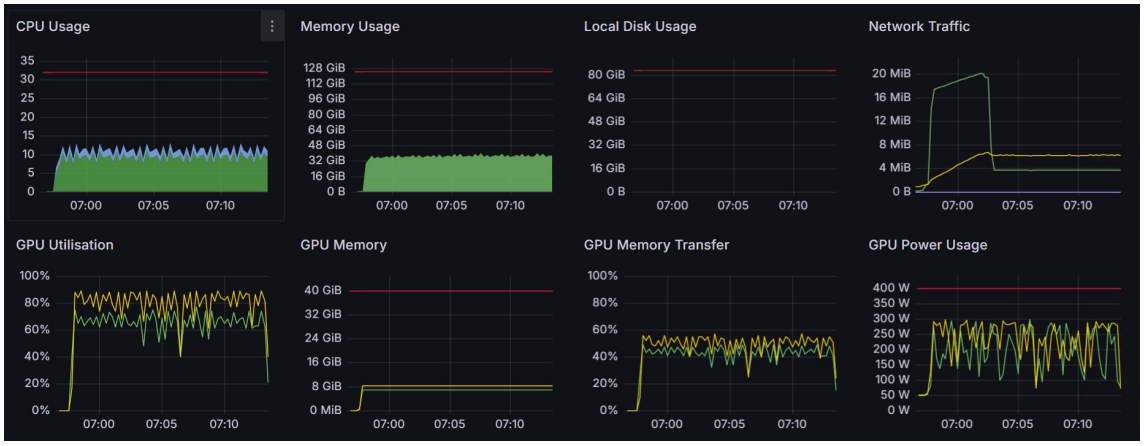



Figure 2.28: Alivs Monitor example, showing information of CPU, IO and GPU Power Usage.

### 2.7.2 Accelerating Matrix Operations with Low-Precision Floating-Point Formats

In our study, we use two types of encoders. For the visual and audio modalities, we use models based on ResNet-18 [38], a convolutional neural network (CNN) architecture where convolution operations dominate the computation. At a low level, these convolution operations are typically converted into matrix multiplications for efficient execution [39]. For the textual modality, we use a RoBERTa [40] model, which is based on the Transformer architecture and primarily composed of standard matrix multiplications.

Since matrix multiplication is a core operation in both encoders, optimising it is important for improving overall performance. Among various optimisation strategies, one particularly effective method targets memory efficiency, as memory bandwidth is often a key bottleneck in matrix operations.

Reducing the precision of floating-point formats is a widely adopted method to help

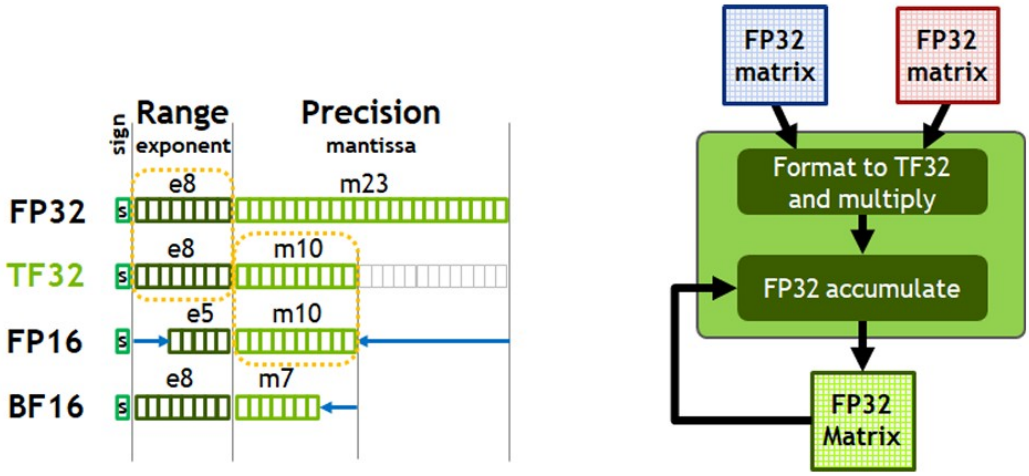



Figure 2.29: (Left) The range of different formats. (Right) TF32 supports FP32 input and output data, enabling easy integration into Deep learning and HPC programs [42].

address this issue. Lower-precision formats reduce the size of data transfers, which in turn reduces memory I/O overhead, improves cache utilisation, and increases computational throughput. These benefits not only accelerate computation but also contribute to lower power consumption [41].

Modern GPUs are provided with dedicated hardware units, such as NVIDIA's Tensor Cores, designed specifically to accelerate low-precision computations. These units support a range of formats, including FP16, BF16, and others, as shown in Figure 2.29, and enable high-throughput execution of matrix operations. Table 2.3 illustrates the theoretical performance gains achievable with different data formats on the NVIDIA A100 GPU.

Table 2.3: Theoretical peak performance of NVIDIA A100 under different data formats [42].

| Format | TFLOPS (Theoretical) | Speedup vs. FP32 |
|---|---|---|
| FP32 | 19.5 | 1.0× |
| TF32 | 156 | 8.0× |
| FP16 | 312 | 16.0× |
| BF16 | 312 | 16.0× |
| INT8 | 624 | 32.0× |

### 2.7.3 Contemporary Optimisation Strategies and Problems in Pytorch

#### 2.7.3.1 Methods for Optimising Data Loading Pipeline

For typical datasets such as images, audio, and text, data augmentation and preprocessing can introduce considerable computational overhead, which may become a bottleneck in the training pipeline [43]. To maintain high throughput and avoid underutilisation of computational resources, efficient data loading is essential. PyTorch provides various built-in optimisations to accelerate data loading and improve GPU utilisation. These include multi-process data loading, data prefetching, and the use of pinned memory, allowing users to fully leverage modern hardware architectures [44].

**Multi-process Data Loading**

Data loading and preprocessing are typically CPU-intensive tasks, making parallelisation a natural choice for improving performance. While multi-threading might appear suitable for such tasks, Python's **Global Interpreter Lock (GIL)** [45], making true parallel execution across threads is not possible within a single Python process. To bypass this limitation, PyTorch adopts a multi-processing method to data loading, where each worker process runs independently across multiple CPU cores.

Figure 2.30 explains PyTorch's multi-process data loader mechanism, where multiple worker processes load and preprocess data in parallel before transferring it to the GPU. This significantly reduces data-loading bottlenecks, particularly when working with large datasets or computationally expensive augmentations.

Table 2.4 presents a practical study of data loading performance with varying numbers of worker processes. As observed, increasing the number of processes can accelerate data loading, but excessive parallelism may introduce diminishing returns due to CPU resources and inter-process communication overhead.

Table 2.4: Performance with varying numbers of worker processes on the AVE training set. The reported time per epoch is averaged over 10 epochs with a batch size of 64. Experiments were conducted on an Alvis A100 node with 16 CPU cores.

| Process Number | Time (s) |
|---|---|
| 1 | 53.6 |
| 2 | 42.5 |
| 4 | 33.5 |
| 8 | 9.4 |
| 16 | 7.6 |
| 32 | 6.5 |
| 64 | 8.3 |

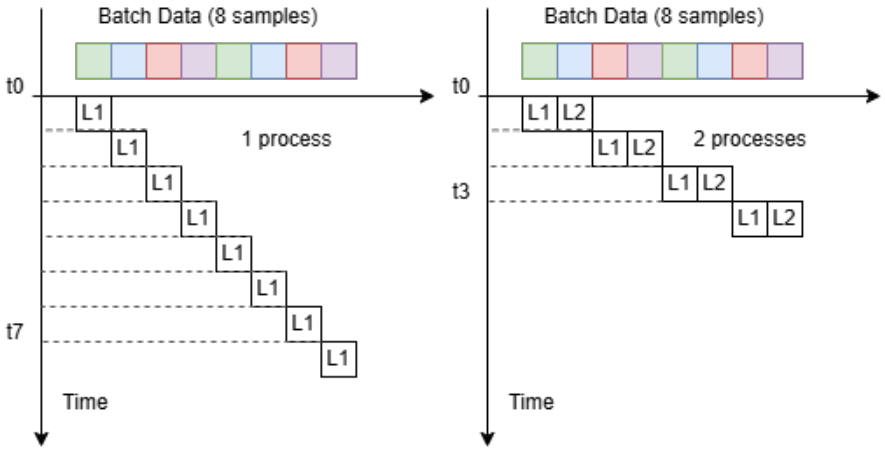



Figure 2.30: Example of single process loader VS. multi-process data loader. Each worker processes independently, loads and preprocesses data, improving throughput.

### Data Prefetching

To further optimise the data pipeline, PyTorch implements **data prefetching**, a technique that enables worker processes to asynchronously load the next batch of data while the current batch is being processed on the GPU. This mechanism effectively eliminates idle time between batches, ensuring a continuous and efficient data flow to the model, thus improving overall throughput.

The prefetching mechanism is controlled by the **prefetch factor** parameter in PyTorch's DataLoader. Each worker process maintains a prefetch queue, where it loads a specified number of batches ahead of time. Figure 2.31 illustrates this mechanism, where multiple workers continuously fetch and prepare data, reducing wait times during training.

Table 2.5 shows the performance impact of different prefetch factors. While moderate prefetching improves efficiency, excessive prefetching may increase memory consumption without a significant speedup.

Table 2.5: Impact of prefetching on data loading performance. Prefetching reduces the average idle time per epoch (averaged over 10 epochs). Experiments were conducted on the AVE training set with 16 worker processes and a batch size of 64. Initialisation time is excluded from the measurements. For prefetch factor 0, idle time is estimated due to PyTorch limitation; the estimation method and corresponding code are provided in Appendix A.2.1.

| **Prefetch Factor** | **Idle Time (s)** |
|---|---|
| 0 | 4.6 (estimated) |
| 1 | 0.76 |
| 2 | 0.72 |
| 4 | 0.74 |
| 8 | 0.68 |

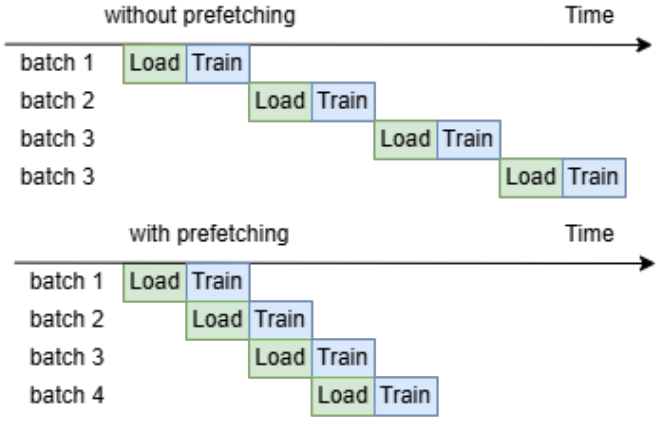



Figure 2.31: Illustration of PyTorch's data prefetching mechanism. Workers load future batches in parallel with GPU execution, reducing idle time.

### Pinned Memory

Transferring data from CPU memory to GPU memory can become a bottleneck due to memory paging and CPU-GPU communication latency. PyTorch addresses this issue with **pinned memory** (or page-locked memory [46]), where data is allocated in a special region of system memory that allows faster transfers to the GPU due to the fact that the page will not be swapped to the cache.

By setting `pin_memory=True` in the `DataLoader`, PyTorch ensures that batches reside in pinned memory before being transferred to the GPU, accelerating data movement and reducing latency. This is particularly beneficial for large batch sizes or for training on multiple GPUs.

Figure 2.32 shows the difference between standard memory (pageable memory) transfer and pinned memory transfer, where pinned memory enables direct GPU access without the need for intermediate CPU buffering.

Table 2.6 compares the training time with and without pinned memory enabled, demonstrating the performance benefits of using pinned memory for efficient data transfer.

Table 2.6: Impact of pinned memory on training time. AVE training set with a batch size of 64, worker processes is 16 and prefetch factor is 4.

| Pin Memory | Epoch Latency (s) |
|---|---|
| Disabled | 1.17 |
| Enabled | 0.33 |

### Persistent Workers

Reconsidering the code shown in Figure 2.33, the data loader will re-initialise the loading worker thread at every epoch [47]. To reduce the time spent in creating the processes, PyTorch allows users to keep all workers and reuse them by setting

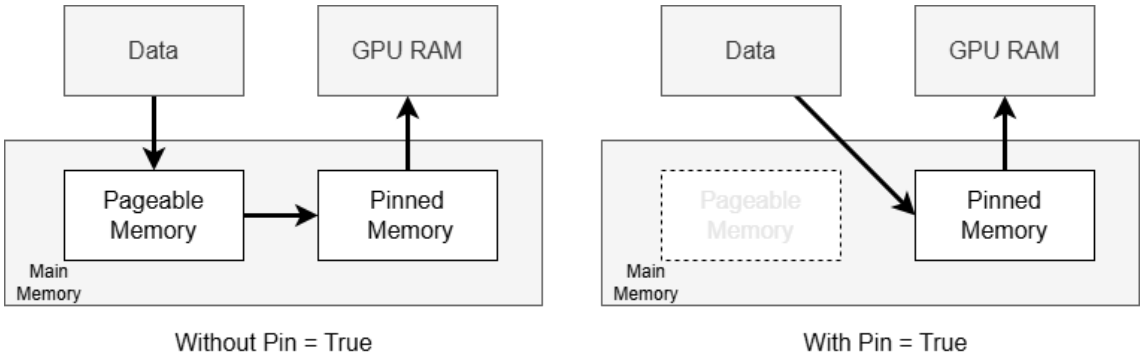



Figure 2.32: Comparison of standard vs. pinned memory transfers. Pinned memory enables direct GPU access, reducing data transfer latency.

**persistent_worker**. The initialisation time with and without this parameter is shown in Table 2.7.

```
    for epoch in epoch_N:
        for step, data in enumerate(dataloader):
            # do train
```

Figure 2.33: A typical code sample of using PyTorch Dataloader.

Table 2.7: Dataloder initialisation (pipeline warm up) time (averaged over 10 epochs). AVE training set, batch size 64, workers 16, prefetch factor 2. Alivs A100 node.

| Persistent Worker | Initializing time (s) |
|---|---|
| Disabled | 2.39 |
| Enabled | 1.05 |

#### 2.7.3.2 Dataloader Initialisation Bottleneck Preventing Higher GPU Utilisation

With GPUs becoming increasingly powerful and cloud-based GPU servers offering flexible resource allocation, forward and training speeds have significantly improved. However, previously negligible overheads can now become notable bottlenecks that hinder overall efficiency. One such issue is the initialisation of the dataloader at the beginning of each epoch, as shown in the example code of Figure 2.33, especially when we adopt a training-validation pattern, there will be 2 stop points due to the training set loader and validation set loader.

For instance, when training on a consumer-grade GPU, such as the RTX 2080 Ti. For small- to medium-scale models and datasets, such as training on the CREMAD dataset with two ResNet18 [38] as encoders for audio and video modalities with structure shown in Figure 2.3, the percentage of time spent on dataloader initialisation is minimal, as shown in Table 2.8. The initialisation step takes only 1.49 seconds in a training epoch, accounting for approximately 3% of the total training time.

However, when running on high-performance hardware like the A100, the impact is considerably different. The profiling results (Figure 2.34) reveal that dataloader initialisation at the beginning of each epoch takes approximately 1 second, contributing to around 7% of the total training runtime of 1 epoch. Moreover, as illustrated in Figure 2.35, this initialisation delay leads to a drop in GPU utilisation during that period, causing inefficient resource usage. With the validating step, the overhead is more significant to the total time, around 12.5%, as shown in Table 2.8.

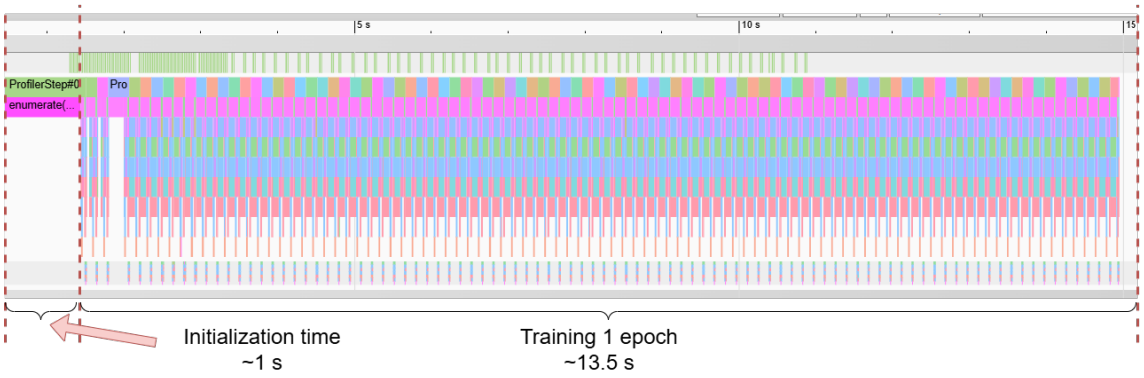



Figure 2.34: Time distribution analysis on A100, highlighting significant overhead from dataloader initialisation. Batch size is 64; Training 1 epoch on CREMAD; dataloader is set with prefactor 2, persistent worker, 16 workers.

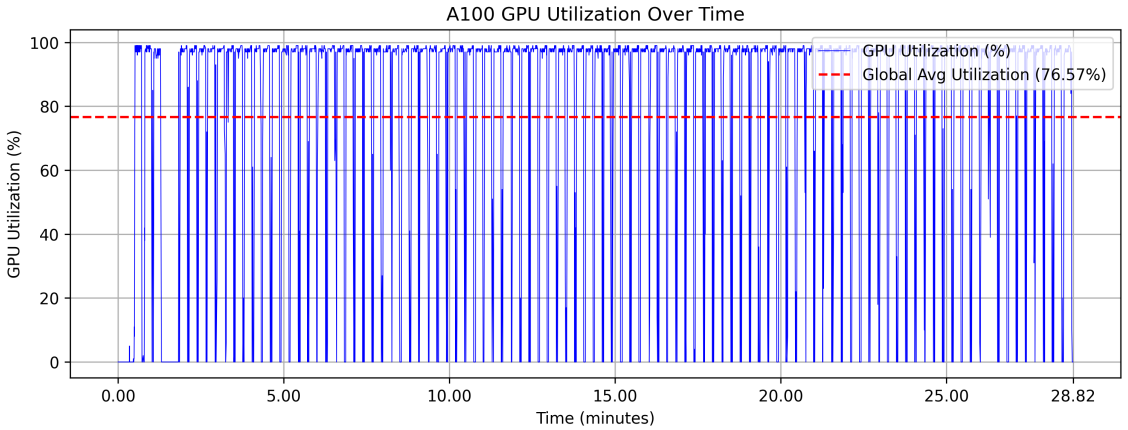



Figure 2.35: GPU utilisation on A100 node, train 100 epochs, showing reduced usage due to dataloader initialisation delays.

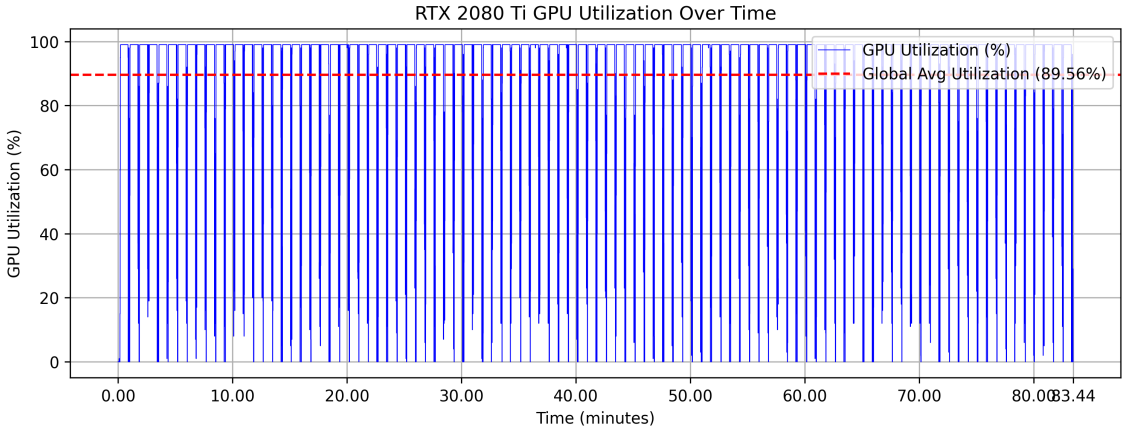



Figure 2.36: GPU utilisation on RTX 2080 Ti, train 100 epochs.

Table 2.8: Time distribution across different devices during a single training-validation step on the CREMA-D dataset with optimal data loader setting. Selected at the 50th epoch.

| Step | RTX 2080 Ti Time (s) | A100 Time (s) |
|---|---|---|
| Training set loader Initialize | 1.49 | 1.00 |
| Training | 42.45 | 13.48 |
| Validating set loader Initialize | 1.54 | 1.00 |
| Validating | 1.39 | 0.54 |
| Total | 49.64 | 16.02 |
| Overhead | 6% | 12.5% |

Therefore, addressing this initialisation bottleneck is essential to fully leverage high-performance GPUs and prevent underutilisation during training.

#### 2.7.3.3 Multi-GPU Training Strategies

Using multi-GPU to optimise the forward and back propagation steps is a main topic of current research in the AI field. PyTorch integrated two methods for supporting multi-GPUs training, called **Data Parallelism** [48] and **Distributed Data Parallelism** [49].

**Data Parallelism (DP)** : Figure 2.37 shows this method, which makes a model run in data parallelism level. Notably, the workload on GPU 0 is heavier than that on GPU 1. And because data dispatching, loss computing, and model updating are all computed on GPU 0, the communication overhead is not ignorable when the batch size is small.

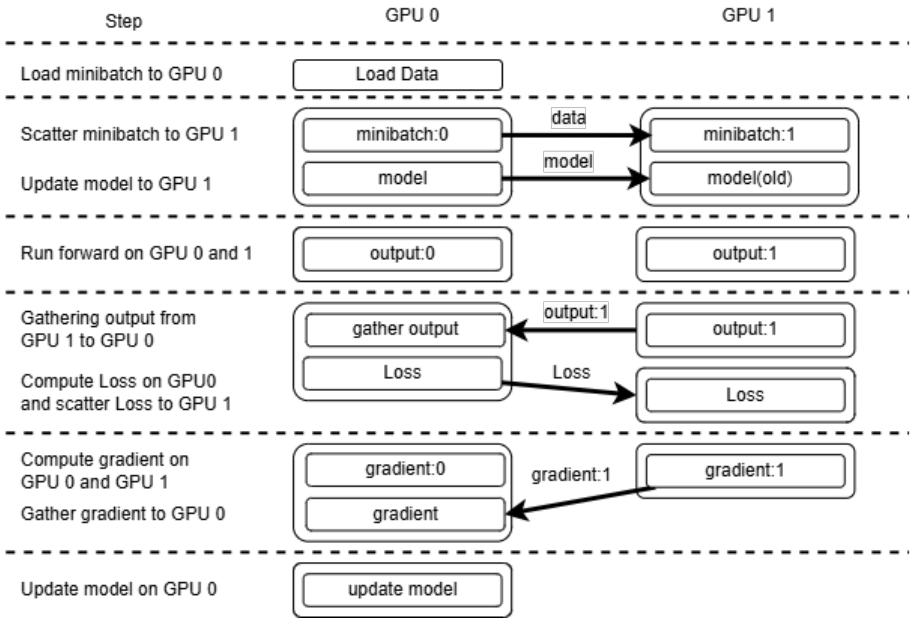



Figure 2.37: Data Parallelism, using 2 GPUs: GPU 0 and GPU 1, GPU 0 is the master GPU, loss and gradient are gathered in GPU 0, model has to be updated from GPU 0 GPU 1 at each epoch.

**Distributed Data Parallelism (DDP)**: Data Parallelism (DP) provides a straightforward implementation in PyTorch, where the user can apply multi-GPU training with just a slight change in the code. However, the performance is suboptimal as the overhead is not ignorable when we want to keep GPUs being used at a high efficiency. Therefore, the method Distributed Data Parallelism, shown in Figure 2.38, is proposed, which computes loss and performs updating on each GPU, just broadcasting the gradient among all GPUs to keep them having the same gradient information. While communication overhead is reduced and the workload is well-balanced across devices, using PyTorch's built-in DDP training method still presents challenges. In particular, it requires additional effort to manage metric collection and logging across different ranks. This makes it less user-friendly when experimenting with new algorithms or working with open-source codebases that were not originally designed for parallel training, often requiring significant refactoring to make them compatible with DDP.

#### 2.7.3.4 Model Structure is ignored when parallelising

Regardless of whether the method is Data Parallelism (DP) or Distributed Data Parallelism (DDP), the underlying structure of the model is often overlooked. Both techniques focus solely on data-level parallelism, where each GPU (or a group of GPUs) runs a full copy of the model. These approaches are efficient when the batch size is large (e.g., 2048) and the model is large-scale, which demands significant computational resources.

However, in research settings, where we typically train with smaller or medium-scale

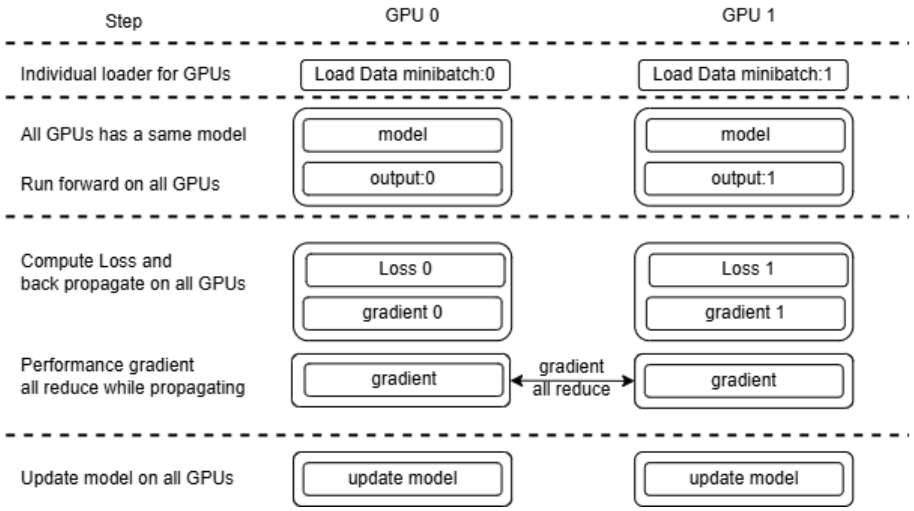



Figure 2.38: Distributed Data Parallelism

models and datasets (e.g., using a batch size of 64), the situation is different. When running on high-performance hardware like the A100, the bottleneck shifts from computational resources to other factors. In such cases, simply adding more GPUs through data parallelism does not lead to performance improvements, as illustrated in Figures 2.39 and 2.40. We can observe that when we apply data parallelism with 2 GPUs to the model, the forwarding time increases from around 40 ms to 74 ms, and the backward time reduces from around 95 ms to 52 ms. Though the overall speed is improved, the low GPU efficiency is a new problem.

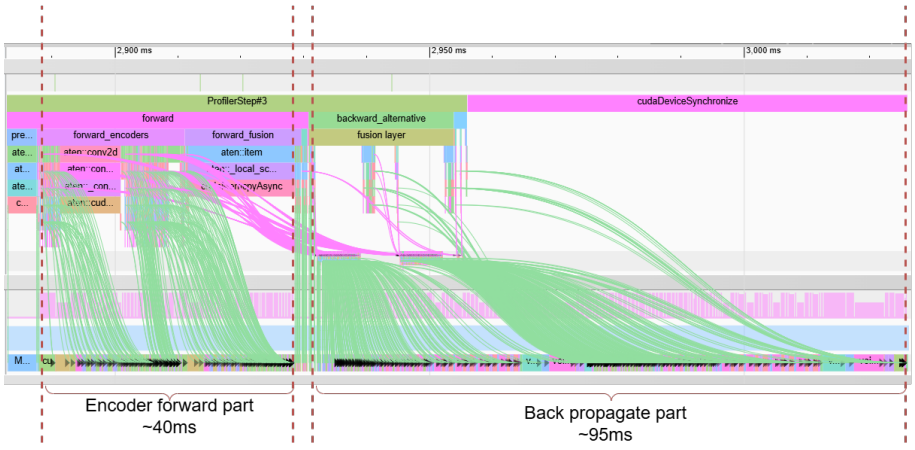



Figure 2.39: Profiling of a batch forward and backward, with a single A100 GPU and no parallelism, no optimisation, on CREMAD training set, batch size is 64.

Given that the model architecture includes a fusion part with multiple encoders, where each encoder acts as an individual sub-model communicating through the fusion layer, applying data-level parallelism without considering the model's structure is not optimal.

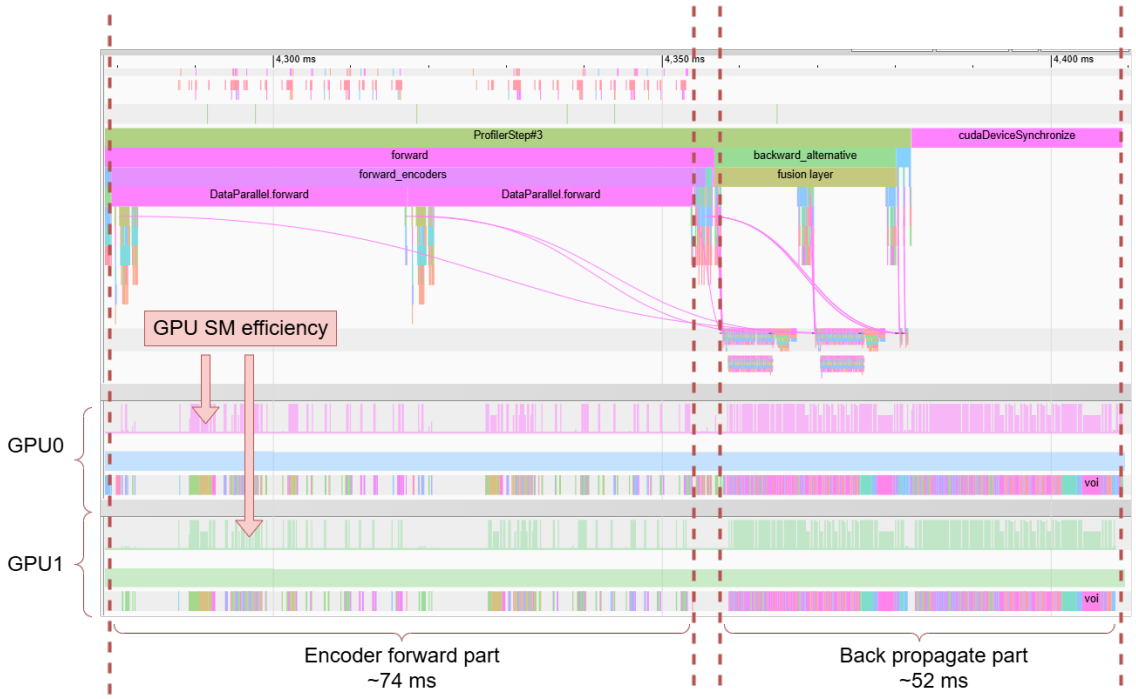



Figure 2.40: Performance with two A100 GPUs using Data Parallelism (DP).

# 3

# Methods

## 3.1 Introducing Different Metrics

Firstly, let us define some metrics that will be used to analyse and compare the training techniques that follow in the next sections.

### 3.1.1 Measuring the Imbalance

In previous works, many metrics have been proposed to measure the imbalance during training. However, these methods are all based on run-time information and do not consider the single modality performance we can get, which means they are not able to directly reflect the imbalance level. Thus, with information from the single-modality performance, we propose a testing time metric, IBF (Imbalance factor), to show the imbalance. For given $M$ modalities, we have the single-modality performance $\{sacc_m\}, m \in M$. Similarly, we can get the modality performance $\{macc_m\}, m \in M$ in a multimodal model by adding additional helpers (probes/classifiers) to each modality output. The definition of IBF is shown in Equation 3.1.

$$IBF = 1 - E[\sum_{m}^{M} min(\frac{macc_m}{sacc_m}, 1)] \tag{3.1}$$

### 3.1.2 Measuring the Bias

To isolate the two-parameter modality bias issue given in Equation 2.39, we employ a cosine classifier for both the sum and concatenation fusion pipelines. Originally proposed in [50] and later extended to multimodal settings in [51], the cosine classifier normalises the modality embeddings as well as the classifier weight vectors, eliminating any preference for modalities that produce larger-scale logits. As in [51], we add a scaling factor $s$ that rescales the combined uni-modal cosine-similarity scores before the softmax layer, ensuring stable convergence. Because both the embeddings and the weights are $L_2$-normalised, the resulting logits differ only in their direction, so any residual imbalance directly reflects the model's confidence in each modality, i.e., a pure modality-bias signal. Regarding the scale parameter, we set it according to [51]'s suggested expression:

$$s \geq \frac{C-1}{2(C+1)} \log\left(\frac{(C-1)p}{1-p}\right) \tag{3.2}$$

To measure the modality-specific bias, we extract the entropy of each modality's prediction in a multimodal context. In concatenation and sum fusion, this is done by feeding only one modality's embedding into the shared prediction layer and setting the others to zero. In Gated and FiLM fusion, this is done by switching and masking the control modality across all modalities. We further explain these mechanisms in Section 3.2.1 for sum and concatenation fusion and Section 3.2.2 for Gated and FiLM fusion. The softmax function (Equation 3.3) converts logits $z_i$ into a probability vector $p_i$. The uncertainty of this distribution is measured by the entropy (Equation 3.4), which is maximal when $p$ is uniform (complete uncertainty) and minimal when nearly all probability mass concentrates on a single class (strong confidence).

$$p_i = \frac{\exp(z_i)}{\sum_{j=1}^{N} \exp(z_j)} \tag{3.3}$$

$$H(p) = -\sum_{i=1}^{N} p_i \log p_i \tag{3.4}$$

We then convert each modality's entropy into a confidence score, $S_m$ (Equation 3.5), followed by normalisation to express the bias, $B_m$, as relative weights (Equation 3.6). $C$ is the number of classes, $M$ the number of modalities ($m = 1, ..., M$) and $\bar{H}_m$ the mean entropy of modality $m$ across the dataset.

$$S_m = 1 - \frac{\bar{H}_m}{\log C}, \quad 0 \leq S_m \leq 1 \tag{3.5}$$

$$B_m = \frac{S_m}{\sum_{j=1}^{M} S_j}, \quad \sum_{m=1}^{M} B_m = 1 \tag{3.6}$$

### 3.1.3 Measuring Modality Reliability

Reliability is a metric that defines how aligned each modality's training and validation performances are. It is defined by the following expression:

$$\text{Reliability}_m = \frac{\text{train loss}_m}{\text{validation loss}_m}. \tag{3.7}$$

If a modality's reliability is one, it means that its training and validation performances are the same. A value larger than one means that the modality is generally performing better in the validation set than in the training set. In contrast, if the value is less than one, then the modality should be performing better in training than in validation.

## 3.2 Generalising Alternating Training to Late Fusion Structures to Solve Modality Imbalance

From the theoretical explanation, we concluded that separating the update of each modality is crucial to overcoming the modality imbalance problem, just as MLA does. Therefore, we attempt to apply this method to joint learning architectures, capable of supporting various fusion mechanisms. This method displays the same computational cost as a standard joint training approach, where each modality, despite being optimised separately, is updated every epoch.

### 3.2.1 Symmetric Fusion Structure

We define a fusion mechanism as symmetric when all modalities are treated equally, without giving some modalities more influence, direction or control over the others. In such methods, if modalities are permuted, the fusion results remain unchanged.

As shown in Figures 2.3, 2.4, and 2.5, both Summation fusion mechanisms as well as Concatenation fusion are symmetric. For this case, it is easy to remove the conflict (disturbance) from other modalities by applying a method similar to that in [8], masking the input to the fusion layer with zero, as shown in Figure 3.1. This method allows us to extract unimodal predictions in a multimodal context. The training process is demonstrated in Algorithm 2. For each modality $i \in M$, the feature embedding is first obtained from the encoder $\Phi_i$ using the input $x_i$ and its corresponding parameters $\theta_i$. Then, to compute the modality-specific loss $\mathcal{L}_i$, all other modality feature embeddings $\{e_{j,j\neq i}\}$ are masked with zeros. The masked embeddings are passed through the fusion structure $\Phi_f$ with parameters $\theta_f$ to generate a modality-specific prediction $\hat{y}_i$. Finally, the parameters $\theta$ of both fusion and the corresponding encoder $\Phi_i$ are updated using $\mathcal{L}_i$. This notation will be used for all presented algorithms.

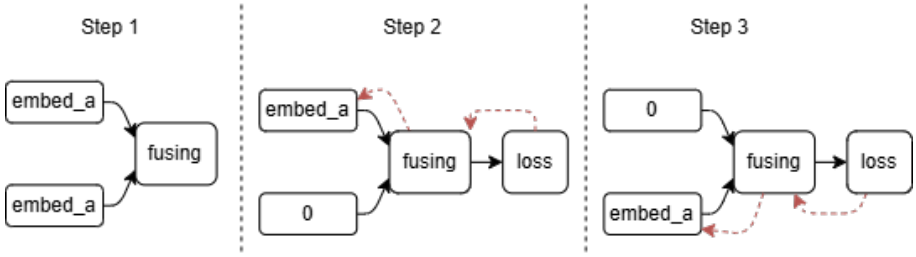



Figure 3.1: Alternating Training for Models with Late Fusion Mechanisms.

**Algorithm 2** Alternating Training (Sum/Concatenation fusion)

**Input:** $\Phi_f$, $\theta_f$, $\Phi_{i,i\in M}$, $\theta_{i,i\in M}$, $\{x_i\}_{i\in M}$, $y$, $L_{CE}$, $\sigma$
$\forall\, i \in M:\ e_i \leftarrow \Phi_i(x_i;\theta_i)$
**for** $i \in M$ **do**
    $e_j \leftarrow 0 \quad \forall\, j \neq i$
    $\hat{y}_i = \sigma(\Phi_f(\{e_i, e_j\};\theta_f))$
    $\mathcal{L}_i \leftarrow L_{\text{CE}}\big(\hat{y}_i,\, y\big)$
    Update $(\theta_f, \theta_i)$ with $\mathcal{L}$
**end for**

### 3.2.2 Asymmetric Fusion Structure

Fusion methods shown in Figure 2.6 and Figure 2.7 are more complex. In these cases, some modalities are given more influence, direction or control over the others, and permuting modalities does not provide the same fusion results. We define such a fusion mechanism as asymmetric when all modalities are not treated equally.

Due to the asymmetric nature and bimodality support limitation of Gated and Film fusion, it is unreasonable to use only one modality as the control modality throughout the entire training process. Based on the alternating training method used for Sum and Concat fusion, we propose the switching and masking method shown in Algorithm 3. There are two differences: first, the fusion structure will be updated via a standard forward and backwards pass using all unmasked embeddings before updating each encoder, ensuring the fusion mechanism works as expected; second, during each encoder update, the control modality $e_c$ is switched to the other modality, which is masked by a constant $c$ (e.g., zeros or ones), depending on the fusion structure. This method allows us to extract unimodal predictions in a multimodal context.

**Algorithm 3** Alternating Training (Gated / FiLM fusion)

**Input:** $\Phi_f$, $\theta_f$, $\Phi_{i,i\in M}$, $\theta_{i,i\in M}$, $\{x_i\}_{i\in M}$, $y$, $L_{CE}$, $\sigma$
$\forall\, i \in M:\ e_i \leftarrow \Phi_i(x_i;\theta_i)$
Update $\theta_f$ with $\{e_i\}$
**for** $i \in M$ **do**
    $e_j = c,\ j \neq i,\ c = 0$ (gated) or 1 (FiLM)
    $\hat{y} \leftarrow \sigma(\Phi_f(\{e_i, e_j\};\theta_f; e_c = e_j))$
    $\mathcal{L} \leftarrow L_{\text{CE}}\big(\hat{y},\, y\big)$
    Update $(\theta_f, \theta_i)$ with $\mathcal{L}$
**end for**

### 3.2.3 The Weakness of Alternating Training Against the Problem of Modality Prediction Bias

As previously mentioned, despite alternating modality training significantly improving performance by mitigating the gradient disappearance issue, differences in convergence speed across modalities can still introduce bias into the overall model. Figure 3.2a displays, for every epoch, the training accuracy for each modality in the audio-visual-text IEMOCAP dataset. As illustrated in the figure, there is a high discrepancy between the learning speed of each modality. This discrepancy is also evident in the bias metrics shown in Figure 3.2b, where the audio modality exhibits a stronger bias compared to both the text and visual modalities, with the visual modality displaying a significantly smaller bias compared to both audio and text. Furthermore, the bias of each modality directly corresponds to its learning speed and accuracy. The fastest-learning and most overfitting modalities exhibit the highest bias metrics throughout training. These results suggest that overtraining faster-learning modalities will impact the model's overall prediction behaviour, as overfitting modalities tend to contribute more heavily to the final decision.

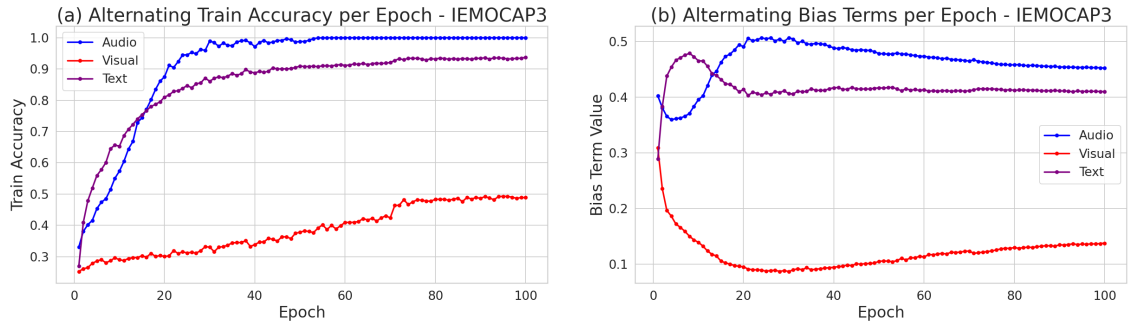



Figure 3.2: IEMOCAP Audio-Visual-Text (a) Train Accuracies (b) Bias Metrics. To extract the Bias Metrics, we used a cosine classifier in Alternating Training.

## 3.3 Solving Modality Imbalance and Prediction Bias with Training Techniques

In this section, we introduce training techniques that, by building on Alternating Training's core principle, aim to minimise the harmful consequences of the problems of modality imbalance and prediction bias. Recognising that the primary cause of modality prediction bias stems from the overtraining and subsequent overfitting of fast-converging modalities, we propose training strategies that focus on determining the appropriate timing for updating each modality. These strategies account for factors such as convergence speed, entropy, and reliability. The underlying idea is to selectively skip the training of certain modalities in specific epochs, thereby allowing individual control over each modality's learning dynamics. This approach addresses issues of overfitting, as well as modality overconfidence and underconfidence, while incorporating the core principles of alternating training to mitigate the modality

imbalance problem. We begin by introducing a method that relies on predefined hyperparameters, and subsequently present alternative strategies that dynamically adapt training based on the model's behaviour and progression.

### 3.3.1 With Hyperparameters - Alternating Multimodal Skip Training (AMST)

To solve the issue of modality prediction bias with the usage of hyperparameters, we introduce AMST, Alternating Multimodal Skip Training. In AMST, we have a hyperparameter per modality that indicates which epochs that modality should be trained. This way, we allow the slower modalities to train much more frequently than the faster ones. In this scenario, multiple modalities may be trained in the same epoch as their skip parameters are independent. However, all modalities are optimised independently, just as in the Alternating Method in Section 3.2.

Given a modality $m$ with respective integer skip parameter $skip_m$, and current training epoch *epoch*, the decision to train $m$ follows (3.8). Algorithm 4 presents the pseudo-code for AMST's mechanism.

$$\text{Train}(m) = \begin{cases} True, & \text{if } epoch \bmod skip_m = 0, \\ False, & \text{otherwise.} \end{cases} \tag{3.8}$$

As demonstrated in Figure 3.3 (a), by training faster learning modalities less frequently than slower learning modalities, we are able to balance their learning speeds, with training accuracy curves displaying far more gradual increases. Consequently, throughout the whole training process, we achieve a better balance of the different modality bias terms compared to the alternating method in Figure 3.2 (b). Even though the visual modality's bias is still significantly lower than the other modalities' bias terms, it increases from around 12% in Alternating to around 21% in AMST. Not only does this method minimise the issue of modality bias, but it also provides efficiency benefits, consuming fewer resources than standard joint training approaches as well as the alternating method described in Section 3.2. As in some epochs, faster-learning modalities' training is skipped, the total number of optimisation epochs for these modalities will be lower.

However, this method's main downside is the fact that it requires fine-tuning for each modality's skip parameter, which can be a tedious and unreasonable task in large-scale data scenarios.

**Note**: For this method, cosine classifiers were not used, as it was developed and published prior to our investigation into their applicability. The subsequent techniques, however, adopt cosine classifiers to eliminate the influence of scale bias.

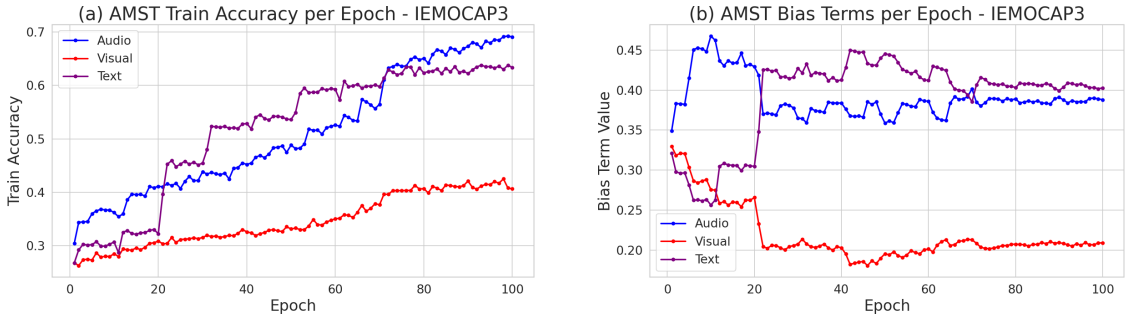



Figure 3.3: AMST IEMOCAP Audio, Visual and Text Training (a) Accuracies (b) Bias Terms.

### 3.3.2 Without Hyperparameters

This section presents different techniques that automatically and dynamically determine which modality should be optimised at each step of the training process. Both of the methods described follow a similar procedure where the only difference is the metric used for computations. In these methods, only one modality is trained per epoch, meaning that, to make fair comparisons with previous methods, we only finish training when one modality reaches the training cap, defined as the total number of optimisation epochs in the alternating method.

To address the issue of modality prediction bias discussed in Section 2.6.1, it is crucial to account for the varying learning speeds of each modality. Our objective is to prevent training from favouring certain modalities in a way that degrades overall model performance. For this purpose, as mentioned in Section 3.1.2, we utilise cosine classifiers, which entirely remove the scale bias parameter. This way, we are able to isolate the bias solely on the cosine similarity scores (direction), which measure how confidently each modality points towards or away from a class. We propose automatic and dynamic skipping mechanisms that determine, at each time step, which modality should be optimised. These approaches generalise the concept of alternating training across different fusion mechanisms.

#### 3.3.2.1 Entropy Training

From Section 2.6.1.2, we know that as training progresses and the model continuously fits to the training data, the prediction entropies tend to decrease. Furthermore, the faster a modality learns, the more pronounced these changes become. Given that the bias metric is directly achieved from the entropy function, as a first option, one might evaluate each modality through this metric (Equation 3.6). We, therefore, introduce a training strategy that prioritises less biased modalities, training, for every epoch, the modality with the lowest value for the bias metric. Consequently, we skip the most biased modalities, ensuring that underconfident modalities receive more optimisation steps, which leads to a more balanced learning process. Algorithm 5 demonstrates Entropy Training's procedure when the metric used is **Bias** (Equation 3.6).

**Algorithm 4** AMST

**Extra input**: Skip hyper-parameters $\{s_i\}, i \in M$, current epoch $E$
$\forall\, i \in M:\ e_i \leftarrow \Phi_i(x_i; \theta_i)$
**Training (Sum/Concatenation fusion):**
**for** $i \in M$ **do**
  $e_m = 0, \forall j \neq i$
  $\hat{y} \leftarrow \sigma(\Phi_f(\{e_i, e_j\}; \theta_f; e_c = e_j))$
  **if** $E \mod s_i == 0$ **then**
    $\mathcal{L} \leftarrow L_{\text{CE}}\big(\hat{y},\, y\big)$
    Update $(\theta_f, \theta_i)$ with $\mathcal{L}$
  **end if**
**end for**
**Training (Gated/FiLM fusion):**
Update $\theta_f$ with $\{e_i\}$
**for** $i \in M$ **do**
  $e_j = c$, $j \neq i$, $c = 0$ (Gated) or 1 (FiLM)
  $\hat{y} \leftarrow \sigma(\Phi_f(\{e_i, e_j\}; \theta_f; e_c = e_j))$
  **if** $E \mod s_i == 0$ **then**
    $\mathcal{L} \leftarrow L_{\text{CE}}\big(\hat{y},\, y\big)$
    Update $(\theta_f, \theta_i)$ with $\mathcal{L}$
  **end if**
**end for**

With this training technique, we allow training to progress until one of the modalities reaches the maximum established number of optimisation epochs.

Even though this method provides a dynamic and automatic way to determine modality optimisation at each time step, its main fallacy is the fact that it always aims to have the same level of confidence among all modalities throughout the training process. However, in some scenarios, it may be beneficial for different modalities to have different levels of confidence due to their different levels of predictive performance. For example, if a particular modality lacks informative value due to missing, noisy, or corrupted data, maintaining the same level of confidence in that modality as in others with higher-quality data can be inefficient. Instead, prioritising resource allocation in the training process to more informative modalities can potentially improve overall model performance by introducing a beneficial bias towards these modalities.

Figure 3.4 displays the training accuracy and bias terms of each modality for the IEMOCAP dataset, including audio, visual and textual modalities. On one hand, as shown in Figure 3.4b, all modalities exhibit approximately equal bias values by the end of training. On the other hand, Figure 3.4a shows that training accuracies remain below 55% for each modality, suggesting that the model may be missing out on further optimisation, particularly from faster-learning modalities, due to its focus on training the slow modality (visual). Overall, entropy training is an efficient technique to maintain confidence levels among modalities, that particularly shines

when modalities have similar levels of predictive performance, and the ideal scenario is an even contribution from all modalities.

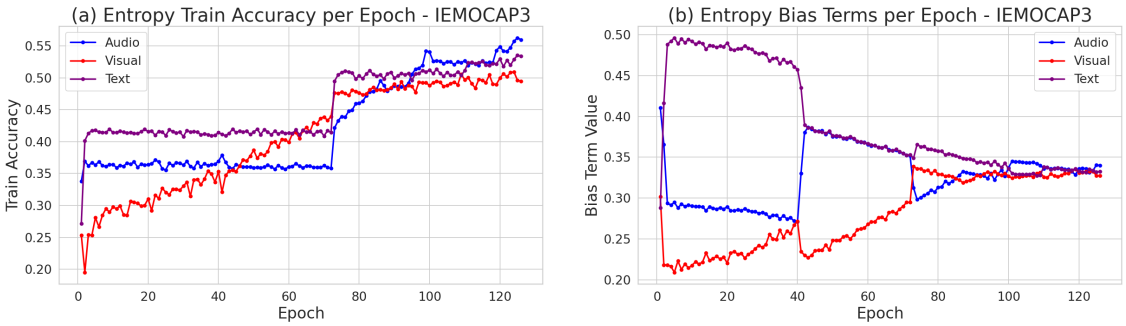



Figure 3.4: Entropy Training IEMOCAP Audio, Visual and Text Training (a) Accuracies (b) Bias Terms.

#### 3.3.2.2 Reliability Training

This training technique works similarly to the previously introduced entropy method. As mentioned before, entropy training's main disadvantage is how it avoids the existence of a bias towards any modality, by trying to maintain the same levels of confidence among all modalities, even if that bias proves to be beneficial. Because the entropy term is dominated by the lowest confidence modality, if this modality happens to be the slowest learning one, the optimisation process will continually focus on it. Faster–converging modalities will then receive comparatively little training, preventing them from fully exploiting the information they contain. Moreover, a modality with systematically lower softmax peaks may still classify a larger fraction of examples correctly, as its lower confidence does not necessarily imply lower accuracy. By forcing slow learners to "catch up", pure entropy regularisation can impede the overall multimodal performance.

We therefore guide the learning and skipping process using **Reliability**, a scale-free metric (Equation 3.7) that enables fair comparison across modalities. At each training epoch, we compute the reliability of each modality to determine which one should be optimised in the following epoch. The modality with the highest reliability is selected for optimisation, as it demonstrates the strongest generalisation to unseen data. To ensure meaningful application of this metric, it is essential to select training and validation sets that are appropriate and representative of the task at hand. This method is demonstrated in Algorithm 5, when **Reliability** (Equation 3.7) is used as the metric.

Since reliability captures the alignment between a modality's training and validation performance, prioritising the most reliable modality during training effectively means favouring the one that generalises best to unseen data at each epoch. This strategy does not directly respond to the learning speed or confidence of each modality, but rather to their generalisation capabilities. The rationale is that continuing to train an overfitting modality is less beneficial than training another modality that,

regardless of higher or lower performance, shows a closer match between training and validation results. Furthermore, faster-learning and more confident modalities will continuously receive updates as long as they generalise well to unseen data. Lastly, by promoting the training of modalities with stronger proportional generalisation, we also encourage growth in their associated bias metric.

Figure 3.5 shows training accuracy (a) and modality bias (b) over time on IEMOCAP under Reliability Training. In contrast to Alternating Training (Figure 3.2 (a)), where the audio and text modalities converge earlier than the visual modality, Reliability Training results in more synchronised convergence across all modalities. Regarding modality bias, Alternating Training (Figure 3.2 (b)) yields the highest bias in audio, followed by text and then visual. Under Reliability Training, this order shifts, with text becoming the most biased modality, followed by audio, while visual bias increases significantly from approximately 12% in the baseline to nearly 25%. Although the bias terms are less evenly distributed across modalities compared to Entropy Training, the audio and text modalities achieve over 70% training accuracy, allowing for greater information extraction than in Entropy Training.

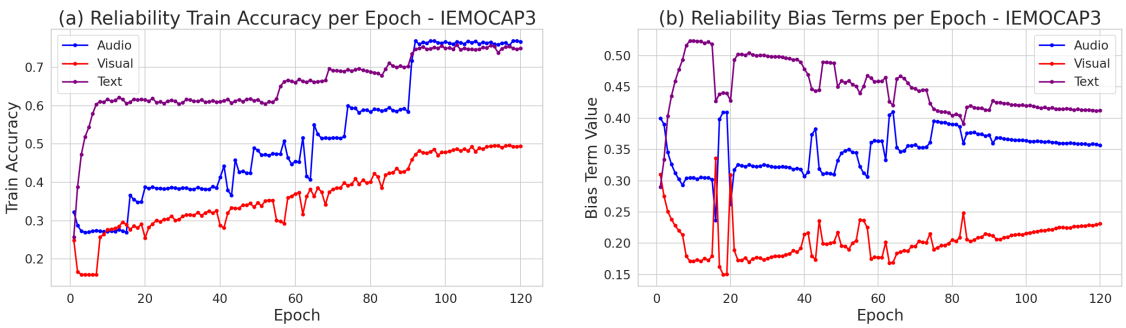



Figure 3.5: Reliability Training IEMOCAP Audio, Visual and Text Training (a) Accuracies (b) Bias Terms.

**Algorithm 5** Entropy/Reliability Training

**Extra input**: Metrics $\{r_i\}, i \in M$
$\forall\, i \in M:\ e_i \leftarrow \Phi_i(x_i; \theta_i)$
**Training (Sum/Concatenation fusion):**
**for** $i \in M$ **do**
  $e_j \leftarrow 0 \quad \forall\, j \neq i$
  $\hat{y} \leftarrow \sigma(\Phi_f(\{e_i, e_j\}; \theta_f; e_c = e_j))$
  $\mathcal{L} \leftarrow L_{\text{CE}}\big(\hat{y},\, y\big)$
  **if** $r_i == \max(\{r_i\})$ **then**
    Update $(\theta_f, \theta_i)$ with $\mathcal{L}$
  **end if**
**end for**
**Training (Gated/FiLM fusion):**
Update $\theta_f$ with $\{e_i\}$
**for** $i \in M$ **do**
  $e_j = c$, $j \neq i$, $c = 0$ (gated) or 1 (FiLM)
  $\hat{y} \leftarrow \sigma(\Phi_f(\{e_i, e_j\}; \theta_f; \text{control} = e_j))$
  $\mathcal{L} \leftarrow L_{\text{CE}}\big(\hat{y},\, y\big)$
  **if** $r_i == \max(\{r_i\})$ **then**
    Update $(\theta_f, \theta_i)$ with $\mathcal{L}$
  **end if**
**end for**

## 3.4 System-Aware Optimisation in Training

In this research, we first describe how our new training methods reduce computational overhead and avoid unnecessary computation at the algorithm level. Given that a large number of experiments are required, including baseline evaluations and the exploration of novel methods, accelerating the training process is critical for saving time and improving overall research efficiency.

To this end, we propose several optimisations to accelerate training: (1) leveraging low-precision formats such as TF32 and BF16 for faster matrix computation; (2) hiding dataloader warm-up overhead through a lightweight fix; and (3) introducing Branch-Level Parallelism (BLP), a simple yet effective model-parallelism strategy that exploits the natural multi-branch structure of our network to enhance hardware utilisation and at the same time keep code easy to implement.

### 3.4.1 Reducing Computation from Algorithm-level

We generalise Alternating Training to late fusion structures. In this architecture, all modalities are optimised the same number of times, which is defined by the total number of training epochs. Therefore, there is no computational optimisation regarding the training process. However, with the introduction of the new training methods

proposed, such as AMST, Entropy and Reliability Training, not all modalities are optimised the same number of times. To properly explain how each method behaves, let $E$ be the total number of training epochs. In Alternating Training, all modalities are optimised for $E$ epochs.

Regarding AMST, each modality's optimisation is guided by its respective skip parameter. This means that, if the current epoch is a multiple of the skip parameter, the modality should be optimised. As a consequence, more than one modality may be optimised per epoch. For this training procedure, we let the number of epochs, $E$, be the same as Alternating Training, and each modality will be optimised, at most, $E$ epochs.

On the other hand, with Entropy and Reliability Training, only one modality is optimised per epoch: the less confident modality in Entropy Training and the most reliable modality in Reliability Training. For fairness purposes, we allow training to proceed until one modality reaches the total number of optimisation epochs set in Alternating Training. For this purpose, let $T$ represent the total number of training epochs we allow each modality to go through, at most. The value of $T$ should be the same as the value of $E$ in Alternating Training. However, for these training procedures, to have one modality reach $T$ optimisation steps, we must increase the total number of optimisation epochs. The rest of the modalities will be optimised less than $T$ times.

All in all, Alternating Training does not offer computational advantages. However, in AMST, Entropy, and Reliability Training, since some modalities' training is skipped during certain epochs, the overall time spent on backpropagation is reduced. The main drawback of Entropy and Reliability Training is that they require loading the data more frequently, since the total number of epochs increases, and the data is loaded once per epoch.

### 3.4.2 Training with Hardware-Specific Floating Point Formats

It is straightforward to use different floating-point formats for training in PyTorch. Considering the trade-off between numerical precision and implementation effort, TF32 and BF16 are both practical and effective choices.

For TF32, only a minor modification is needed before initialising the training environment, as displayed in Figure 3.6.

```
1 torch.backends.cuda.matmul.allow_tf32 = True
2 torch.backends.cudnn.allow_tf32 = True
```

Figure 3.6: Enabling TF32 in PyTorch.

For BF16, slightly more changes are required, as shown in Figure 3.7.

```
with autocast(dtype=torch.bfloat16):
    output = model(input)
    loss = loss_fn(output, target)
```

Figure 3.7: Using BF16 for training in PyTorch.

Both approaches require minimal code modifications, making them highly suitable for rapid experimentation and prototyping in fast-paced research settings.

### 3.4.3 Hiding Data Loader Initialisation Overhead with Multi-threading

As discussed in Section 2.7.3.2, even when using a data loader with optimal parameter settings, including persistent workers, pinned memory, and multiple workers, the initialisation overhead (pipeline warm-up) at the beginning of each epoch is still non-ignorable. Many methods can optimise this overhead. For example, optimising the time loading data from storage to memory by moving all data to local disk and preloading all data into memory. But considering these are not a perfect solution as the overhead of conducting data augmentation is unavoidable in a single-thread pipeline pattern, and we want an easy implementation and minimal code modification. Therefore, we propose a **multi-threaded initialisation** strategy. This method shares a similar motivation with multi-process data loaders: It uses a background thread to pre-initialise the data loader (loading the first batch) for the next epoch, overlapping computation with the training step of the present epoch. The scheduling strategy is illustrated in Algorithm 6.

**Algorithm 6** Optimized Loader

```
Input: Loader_queue, Epoch_N
Initialise Loader_queue
for epoch = 1 to Epoch_N do
    loader = Loader_queue.get()
    Loader_queue.initialize_new_thread()
    train(loader)
end for
```

### 3.4.4 Branch-Level Parallelism

Instead of adopting data-level parallelism (DP), which introduces significant communication overhead and limits scalability, we leverage a parallelism strategy inspired by Gpipe [52], a model-level parallelism method that assigns different layers of a model to different devices for pipelined execution.

However, unlike Gpipe, our challenge does not lie in insufficient memory or compute capacity on a single GPU. The main challenge lies in the limited hardware utilisation caused by communication overhead in the Data Parallelism (DP) approach. On the

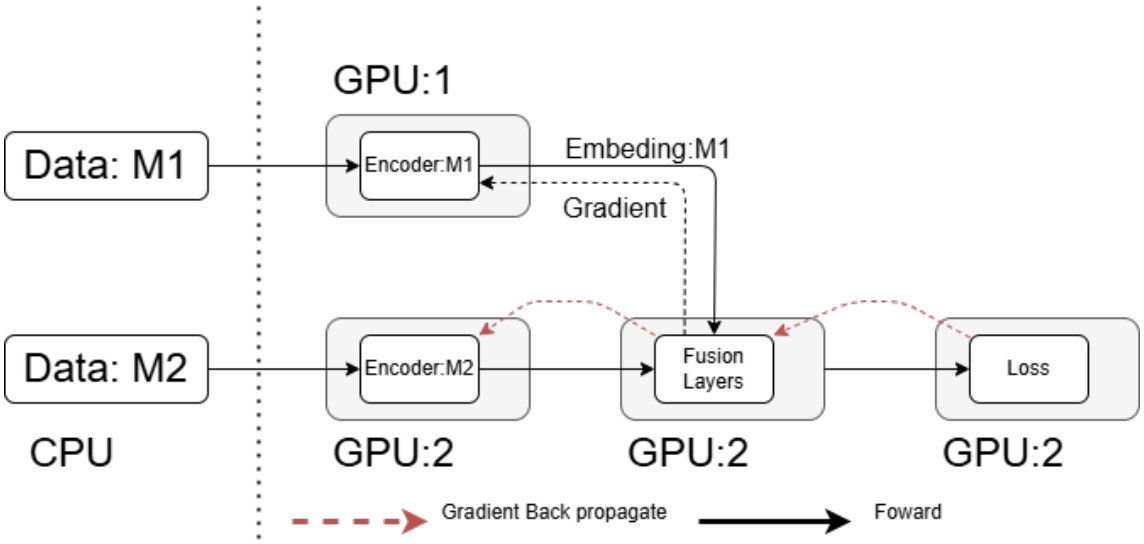



Figure 3.8: Branch-level parallelism, 2 modalities $M1, M2$

other hand, while Distributed Data Parallelism (DDP) offers better scalability, it introduces significant debugging difficulty and requires substantial code refactoring, which is impractical in a rapidly evolving codebase.

Given the specific structure of our model, consisting of multiple independent encoder branches, we propose a branch-level parallelism strategy, as shown in Figure 3.8. In this design, each encoder branch is assigned to a separate GPU and executed concurrently. The fusion and loss computation are performed centrally on a designated master device. The computed loss is then propagated back to each device for an independent backward pass and parameter updates.

This method achieves efficient parallel computation across branches with minimal communication overhead. It also preserves most of the existing code structure, requiring changes only to the forward and backward pass, while leaving logging and extra code largely unchanged.

Moreover, device assignment for each encoder can be flexibly chosen based on the computational workload of each modality, allowing for better load balancing and resource utilisation across GPUs.

# 4

# Experiments, Results and Discussion

## 4.1 Experimental Data

To evaluate our experiments, we employ five benchmark datasets spanning audio, visual, and textual modalities: Crowd-Sourced Emotional Multimodal Actors Datase (CREMA-D) [4] comprises 7,442 audio-visual clips annotated with six basic emotions; the Audio-Visual Event (AVE) [22] dataset contains 4,143 ten-second videos covering 28 event categories; the Multimodal Sentiment Analysis (MVSA) dataset [23] includes 5,129 Twitter image-text pairs labeled positive, neutral, or negative; Interactive Emotional Dyadic Motion Capture (IEMOCAP) [24] offers aligned audio, visual, and textual recordings annotated with emotion labels. To further demonstrate the applicability of our method, we evaluate it on two versions of this dataset: one comprising audio and visual modalities, and another comprising audio, visual, and text modalities. As a final dataset, UR-FUNNY [25] consists of 16,514 audio-visual segments with transcripts labelled as humorous or non-humorous.

## 4.2 Experimental Settings

All experiments are conducted using the exact same backbone encoders. We employ a ResNet18 [38] backbone for the audio and visual modalities, and a RoBERTa-based [40] encoder for the text modality. For the AVE dataset, we adopt the official train/validation/test split as provided in [22]. For the remaining datasets, we randomly partition the data into 80% training, 10% validation, and 10% testing subsets. For the visual modality, we extract three representative frames per video clip, except in the MVSA dataset, where only a single image is provided per sample. Audio inputs are transformed into filterbank features [53]. Text inputs are tokenised using the RoBERTa tokeniser. We use a shared learning rate of 0.001 across all modalities, with Stochastic Gradient Descent, [54], (SGD) as the optimiser, a weight decay of 1e-4, and a batch size of 64. For SOTA methods and the Alternating Training approach, the learning rate decay ratio is 0.1 and occurs at epoch 70, after all modalities have undergone 70 optimisation epochs. To ensure fairness, Reliability and Entropy Training use the same decay rate, but the decay is triggered when

any single modality reaches 70 optimisation epochs, meaning the others will have undergone fewer updates at that point. For state-of-the-art (SOTA) baselines, we adopt the original hyperparameters as specified in their respective works and use publicly available code when provided. All experiments were run on a computational cluster node with an A100 GPU, with 72GB of memory.

Regarding AMST, all modalities' skip parameters require fine-tuning and their optimal values may differ depending on each modality's predictive performance and task at hand. The visual skip parameter was set to 1 in all datasets (i.e., we do not skip its training). Concerning audio, its skip parameter was set to 5 for CREMA-D, 2 for AVE, 4 for IEMOCAP and 6 for UR-FUNNY. For text, we set the text skip parameter to 10 for MVSA, IEMOCAP and UR-FUNNY.

Due to the nature of both Entropy and Reliability Training, only one modality is optimised per training epoch. Consequently, training is terminated once any modality reaches its maximum number of allowed optimisation epochs. To ensure a fair comparison with state-of-the-art (SOTA) methods, we fix the total number of training epochs for all SOTA baselines to 100. For Entropy and Reliability Training, we set the total number of training epochs to 200 for bimodal tasks and 300 for trimodal tasks, effectively allowing up to 100 optimisation epochs per modality. However, since Reliability and Entropy Training halt when any single modality reaches this limit, the remaining modalities may receive fewer updates compared to those in standard SOTA training regimes. Furthermore, both these strategies utilise cosine classifiers with the scale parameter set according to [51].

To demonstrate our methods' applicability and algorithm-level optimisations and benchmark them against state-of-the-art approaches, we conduct four experiments: (1) we assess the imbalance metric (Equation 3.1) across all datasets and methods to quantify each approach's robustness to modality imbalance; (2) we conduct an ablation study by comparing each fusion mechanism under a standard joint training setup without optimisation strategies, and then progressively introducing the alternating method, as well as AMST, the entropy-based and reliability-based training approaches; (3) we benchmark the overall performance of our methods against leading SOTA techniques; (4) And, finally,we investigate the model's preference for certain modalities under different training strategies by analysing the number of optimisation epochs per modality and evaluating their modality-specific bias metrics; For system level optimisation, (5) we compare the performance of training a single-modal encoder using TF32 and BF16 on both A100 and A40 GPUs across different batch sizes. This evaluation is conducted for all modalities across all datasets. (6) To show the impact of data loader optimization on training efficiency, we measure and compare the training and validation time per epoch under various conditions, including different datasets, batch sizes, and hardware devices. (7) To evaluate the speed-up achieved by different parallelization strategies, we conduct experiments under the same conditions as in (6).

## 4.3 Results and Discussion

The following sections display the results of all experiments performed. We first present and discuss the Algorithm-level improvements, and after the HPC-Aware Optimisations and Computational Efficiency.

### 4.3.1 Algorithm-level Experimental Results

In this section, we present all the results across all datasets of the algorithm-level experiments: Measuring the imbalance in each method; comparing the introduced training structures one by one; benchmarking the overall performance of our methods against SOTA methods; evaluating and analysing the train bias metrics of each modality and their respective number of optimisation epochs in each training strategy; and comparing computational expenses across all methods. All results correspond to the average of three seeds. We further provide each value's standard error in Appendix A.1.

#### 4.3.1.1 Measuring the Imbalance

The performance of each modality across all datasets, both in unimodal settings and in multimodal contexts, is presented in Table 4.1, covering SOTA methods with joint training, alongside our Alternating (Alt), AMST, Entropy (Ent) and Reliability (Rel) Training approaches. Additionally, we report the respective Imbalance Factor (IBF) scores for each method, providing insight into their effectiveness at addressing modality imbalance when present. As a baseline, we employ a naive training strategy, denoted by the "Naive" column in Table 4.1, which involves a straightforward joint learning approach using concatenation-based fusion, without the application of any performance-enhancing techniques. This setup allows us to assess the presence and extent of modality imbalance in each dataset.

Based on the results shown in the "Naive" column, we observe that the datasets most affected by modality imbalance are CREMA-D, AVE, and IEMOCAP, whereas MVSA and URFUNNY exhibit near-zero IBF values, indicating minimal imbalance. As demonstrated by the table, current SOTA methods do not properly address Modality Imbalance in datasets where this property is most present. All SOTA methods have similar IBF values compared to the Naive method, with only MSLR providing slight improvements regarding this metric. These results are expected, as all these methods optimise a single joint loss and are susceptible to the problem of gradient disappearance. For these methods, the visual modality, which happens to be the slow-learning one, is extremely penalised compared to the faster-learning audio and text modalities.

Alternating Training and all the methods derived from it, AMST, Entropy and Reliability Training, consistently achieve the lowest IBF values across all datasets, with particularly strong performance on the most imbalanced datasets. These results stem from the fact that modalities are individually optimised and are able to achieve values close to their full predictive potential. These methods avoid the gradient

Table 4.1: Comparison of accuracies and IBF metrics of different methods on all datasets (**Concat Fusion**, test set). V: visual, A: audio, T: text. Single: single-modal baseline. Alt: Alternating. Lower IBF means less imbalance. The best results are in bold and the second-best underlined.

| Dataset | | Single | Naive | OGM | PMR | MSLR | Alt | AMST | Ent | Rel |
|---|---|---|---|---|---|---|---|---|---|---|
| CREMA-D | V | 60.5 | 22.4 | 24.7 | 18.5 | 41.2 | 57.0 | 59.7 | **61.8** | 61.7 |
| | A | 59.1 | 55.1 | 53.6 | 50.5 | 48.8 | 59.7 | 61.7 | **62.6** | 61.7 |
| | IBF | 0 | 0.349 | 0.342 | 0.420 | 0.247 | 0.029 | 0.007 | **0.000** | **0.000** |
| AVE | V | 28.5 | 14.4 | 15.5 | 11.1 | 22.0 | 31.0 | 30.1 | 31.0 | **31.1** |
| | A | 55.3 | **57.2** | 50.8 | 49.4 | 50.4 | 56.2 | **57.2** | 50.8 | 54.5 |
| | IBF | 0 | 0.247 | 0.269 | 0.359 | 0.158 | **0.000** | **0.000** | 0.041 | 0.007 |
| MVSA | V | 57.7 | 56.9 | 56.5 | 56.7 | 54.1 | 57.8 | **59.1** | 56.9 | 57.9 |
| | T | 70.8 | 68.3 | 68.8 | 66.0 | 68.2 | **70.3** | 70.2 | 68.1 | 69.9 |
| | IBF | 0 | 0.025 | 0.025 | 0.043 | 0.050 | **0.004** | **0.004** | 0.026 | 0.006 |
| IEMOCAP | V | 46.5 | 26.7 | 26.7 | – | 31.9 | 33.6 | 40.2 | **45.7** | 43.0 |
| | A | 49.2 | 43.1 | 41.6 | – | 48.3 | 45.9 | 48.3 | 46.2 | **50.0** |
| | T | 62.3 | 58.8 | 56.7 | – | 49.6 | 60.9 | 57.2 | 52.3 | **61.1** |
| | IBF | 0 | 0.202 | 0.223 | – | 0.179 | 0.122 | 0.079 | 0.080 | **0.032** |
| UR-FUNNY | V | 49.2 | 49.8 | 50.3 | – | 50.0 | **50.6** | 49.5 | 49.5 | 50.4 |
| | A | 59.6 | 55.8 | 55.8 | – | **60.0** | 56.3 | 58.5 | 57.3 | 57.0 |
| | T | 69.0 | 68.5 | 68.3 | – | 65.7 | 69.0 | 69.4 | 67.9 | **70.0** |
| | IBF | 0 | 0.024 | 0.025 | – | 0.016 | 0.018 | **0.006** | 0.018 | 0.015 |

disappearance problem and do not suffer from negligible gradient updates. However, we observe that among our newly introduced methods, Entropy Training generally results in the highest IBF values. This is because it strives for perfect confidence balance across all modalities, often becoming stuck on a slow-learning modality that never reaches the confidence level of the others. As a result, the training process neglects the faster-learning modalities, preventing them from reaching their full potential.

This highlights our method's superior ability to effectively utilise information from each modality compared to existing methods.

#### 4.3.1.2 Comparing Performance of Training Strategies Across Different Fusion Structures

The results for four different fusion mechanisms, concatenation, summation, gated, and FiLM fusion, across various training strategies, are presented in Table 4.2. These strategies include a baseline joint training approach without optimisation techniques (Naive), basic Alternating Training generalised to joint training (Alternating), AMST, a training strategy that uses hyperparameters, and two training strategies without hyperparameters that implement cosine classifiers: Entropy and Reliability Training.

From these results, several important conclusions can be drawn. Alternating Training, compared to Naive Training, clearly improves prediction performance for the

CREMA-D, AVE, and IEMOCAP2 datasets across all fusion mechanisms. These datasets not only exhibited significant modality imbalance, previously identified in Table 4.1 by their higher IBF metrics, but are also bimodal datasets with less complex modality interactions. Although Alternating Training also reduces the IBF metric for IEMOCAP3, there is no improvement in fusion performance compared to Naive Training. This dataset involves three modalities and consequently more complex behaviours between them. This suggests that dealing with Modality Imbalance by itself in more complex multimodal scenarios may not be sufficient to obtain ideal results. For MVSA and URFUNNY, datasets with initially low IBF metrics, the results under Alternating Training remain similar to those of Naive Training.

Next, we analyse AMST's results. This approach improves upon Alternating Training by initially addressing modality bias, particularly in CREMA-D and IEMOCAP2. In these datasets, the modalities have similar unimodal performance, and a more balanced modality bias distribution leads to better outcomes. For the remaining datasets, no clear improvements are observed, though performance remains comparable. However, it is important to keep in mind that the skip parameters were fine-tuned in order to obtain the results for this method.

Entropy Training, while improving accuracies for CREMA-D and IEMOCAP2, generally leads to decreased overall prediction performance compared to Alternating Training. In these datasets, both modalities achieve similar unimodal prediction accuracies, as shown in Table 4.1 (IEMOCAP V and A rows under the Single column). Entropy Training promotes equal modality confidence, as reflected in the bias metrics shown in Table 4.4 (Ent row). As previously mentioned, if one modality learns significantly slower than the others, this strategy tends to focus excessively on optimising that modality, while giving insufficient attention to the faster-learning ones. This leads to underoptimisation of the latter and ultimately worsens overall performance. Additionally, by enforcing equal confidence across modalities, their contribution to the final prediction becomes similar, even in cases where some modalities perform much better than others. As a result, in datasets where modalities differ notably in predictive accuracy, such as AVE, IEMOCAP3, and UR-FUNNY, this method can harm overall results. In summary, this strategy is most effective when all modalities exhibit comparable predictive strength.

Finally, Reliability Training proves to be the most balanced and well-performing training technique. Its overfitting-aware training mechanism leads to strong performance across all datasets and fusion mechanisms, demonstrating both robustness and adaptability. As demonstrated in Table 4.2, for all datasets except AVE, where this method performs similarly to the best method, Reliability Training achieves either the best or the second-best accuracy across all methods. Focusing the training process on the least overfitting modalities proves to be beneficial as all modalities are optimised as long as they generalise well to unseen data. This way, all modalities can achieve values close to their full potential. In addition, by avoiding the training of overfitting modalities, the problem of Modality Prediction Bias is addressed, where these modalities would previously dominate the prediction. Moreover, for IEMOCAP3, a dataset where Alternating Training successfully addressed Modality

Imbalance but still resulted in underwhelming performance, Reliability Training yields improved results over both the Alternating and Naive methods by additionally addressing Modality Prediction Bias.

Table 4.2: Accuracy of four fusion methods under Naive, Alternating, AMST, Entropy, and Reliability Training. For each dataset the best results are in bold and the second best are underlined.

| Method | Fusion | CREMAD | AVE | MVSA | IEMOCAP2 | IEMOCAP3 | URFUNNY |
|---|---|---|---|---|---|---|---|
| Naive | Concat | 64.2 | 58.8 | 71.0 | 49.2 | 69.3 | <u>71.6</u> |
| | Sum | 65.5 | 60.7 | 71.6 | 50.7 | 68.2 | 70.2 |
| | Gated | 64.2 | 52.5 | 71.9 | 47.9 | – | – |
| | FiLM | 62.1 | 59.7 | 72.4 | 50.2 | – | – |
| Alternating | Concat | 74.6 | <u>64.4</u> | 73.0 | 55.5 | 66.4 | 69.1 |
| | Sum | 72.9 | 62.5 | 72.6 | 53.5 | 66.9 | 70.5 |
| | Gated | 70.1 | 62.3 | 71.2 | 54.8 | – | – |
| | FiLM | 75.9 | 60.2 | 71.0 | 54.4 | – | – |
| AMST | Concat | 77.7 | 64.2 | 71.4 | **60.2** | 65.5 | 71.2 |
| | Sum | 77.1 | 62.7 | 72.2 | 57.7 | 67.2 | 71.0 |
| | Gated | 76.2 | **64.9** | **73.4** | 53.7 | – | – |
| | FiLM | 77.7 | 59.5 | 71.2 | 51.6 | – | – |
| Entropy | Concat | **79.2** | 60.2 | 70.3 | 58.7 | 64.7 | 68.9 |
| | Sum | 78.0 | 59.1 | 72.7 | 57.9 | 64.6 | 68.4 |
| | Gated | 77.3 | 58.5 | 72.0 | 57.0 | – | – |
| | FiLM | 77.6 | 52.6 | 71.8 | 54.5 | – | – |
| Reliability | Concat | <u>78.5</u> | 63.8 | 73.0 | **60.2** | <u>71.2</u> | **72.3** |
| | Sum | 78.1 | 64.1 | <u>73.2</u> | <u>59.3</u> | **71.3** | 71.1 |
| | Gated | 74.7 | 62.9 | 70.1 | 58.6 | – | – |
| | FiLM | 77.4 | 59.0 | 71.4 | 57.1 | – | – |

#### 4.3.1.3 Comparing Performance with SOTA Methods

Table 4.3 compares the results of four state-of-the-art (SOTA) methods and the baseline Naive approach against our proposed approaches: base Alternating Training, AMST, Entropy Training, and Reliability Training. From the results, we observe that our methods outperform all SOTA baselines across every dataset. Alternating Training performs similarly to MLA, which is expected given their methodological similarities. However, Alternating Training still outperforms MLA across all datasets, likely due to differences in the fusion strategies used. In this table, Alternating relies on concatenation fusion, while MLA employs an entropy-based fusion mechanism.

AMST builds on Alternating Training and improves its performance across all datasets except AVE, where Alternating Training remains the best-performing method. This aligns with each modality's predictive performance in AVE, where it is advantageous to train more heavily and maintain a higher bias towards the faster-learning modality, audio, which achieves nearly twice the predictive performance of the visual modality. On the other hand, Entropy Training only surpasses other methods on CREMA-D, which highlights some of its limitations. Specifically, faster-learning modalities tend to be under-optimised under Entropy Training, which can hurt performance in

datasets like AVE, where certain modalities (e.g., audio) have significantly better performance than others (e.g., visual). As shown in Table 4.4 (Entropy row), this method trains the audio modality fewer times than any other, which contributes to its lower performance. That said, even though Entropy Training doesn't deliver the best overall results, it can still be valuable in scenarios where all modalities have similar levels of predictive capabilities, as shown for CREMA-D and IEMOCAP2.

Finally, our most balanced, well-rounded, and robust method is Reliability Training. It consistently outperforms all other approaches across most datasets and achieves performance comparable to the best methods on the remaining ones. Compared with previous SOTA methods (OGM, PMR, MSLR and MLA), the most significant improvements are observed on the CREMA-D, AVE, IEMOCAP2, and IEMOCAP3 datasets, which aligns with our earlier analysis identifying considerable modality imbalance in these datasets. By effectively addressing both the Modality Imbalance and the Modality Prediction Bias problems, notable performance gains are achieved. In contrast, for MVSA and UR-FUNNY datasets, where Modality Imbalance was shown to be minimal, our improvements primarily stem from addressing Modality Prediction Bias. Even though the gains in these cases are smaller, they remain meaningful, particularly when compared to the closest alternating SOTA method, MLA.

Table 4.3: Comparison of Alternating, AMST, Entropy and Reliability Training against SOTA methods across six datasets. We use concat fusion for all methods and MLA's entropy-based fusion for MLA. The best results are in bold and the second best are underlined.

| Method | CREMA-D | AVE | MVSA | IEMOCAP2 | IEMOCAP3 | UR-FUNNY |
|---|---|---|---|---|---|---|
| Naive | 64.2 | 58.8 | 71.0 | 49.2 | <u>69.3</u> | <u>71.6</u> |
| OGM | 63.7 | 59.0 | 70.9 | 51.6 | 67.1 | 70.3 |
| PMR | 67.5 | 58.8 | <u>72.3</u> | 51.9 | – | – |
| MSLR | 76.5 | 63.8 | 71.5 | 51.8 | 61.4 | 69.2 |
| MLA | 73.7 | 63.6 | 71.4 | 53.9 | 65.7 | 70.3 |
| Alternating | 74.6 | **64.4** | **73.0** | 55.5 | 66.4 | 69.1 |
| AMST | 77.7 | <u>64.2</u> | 71.4 | **60.2** | 65.5 | 71.2 |
| Entropy | **79.2** | 60.2 | 70.3 | <u>58.7</u> | 64.7 | 68.9 |
| Reliability | <u>78.5</u> | 63.8 | **73.0** | **60.2** | **71.2** | **72.3** |

#### 4.3.1.4 Analysing Training Bias Metrics and Number of Training Epochs

In this section, we analyse the bias metric for each modality and the number of optimisation epochs per modality across all datasets, under four training strategies: basic alternating training, AMST, Entropy Training, and Reliability Training. As shown in Table 4.4, under the basic alternating method, the model exhibits a strong bias towards faster-learning modalities, specifically, audio and text, while neglecting the visual modality. In this approach, all modalities are optimised an equal number of times, regardless of their learning dynamics.

For AMST, the skip parameters were fine-tuned to yield the bias metrics that maximised performance for each dataset. In CREMA-D, where both modalities have similar predictive performance, an approximately 50–50 bias is ideal, and AMST approximately achieves this balance. For AVE, where the visual modality's performance is considerably lower, the optimal bias favours audio. Among the methods, AMST produces bias values most closely aligned with this target, similar to those of baseline Alternating Training. Across the remaining datasets, AMST's bias values show a degree of alignment with those of Reliability Training. This is expected, as Reliability Training is our best-performing method, and AMST's parameters were fine-tuned individually for each dataset. However, the performance differences between these methods arise from the nature of their skipping mechanisms. AMST employs a static skipping strategy, whereas Reliability Training uses a dynamic approach.

For Entropy Training, bias values are the most balanced, displaying near-uniform bias across all datasets except UR-FUNNY. For UR-FUNNY, the visual modality's bias parameter remains significantly lower than those of the other modalities, reflecting its slower learning speed and poorer predictive performance. However, as previously discussed, this method underperforms compared to Reliability Training, mainly due to the under-optimisation of the faster-learning, more confident modalities. Under Entropy Training, the visual modality is consistently prioritised, while the audio and text modalities are optimised substantially fewer times.

Reliability Training offers a middle ground between basic Alternating and Entropy Training. While its bias metrics are not as evenly balanced as those of Entropy Training, the discrepancies between modalities are significantly less pronounced than in basic Alternating Training. Additionally, we observe that in datasets such as CREMA-D and AVE, the faster-learning modalities are optimised more frequently than under Entropy Training. However, this trend does not fully hold in IEMOCAP3 and UR-FUNNY. In these datasets, the text modality receives significantly fewer updates under Entropy Training compared to Reliability Training, while the audio modality is optimised more frequently in Entropy Training.

**Note**: All SOTA methods perform 100 training epochs for every modality, just as Alternating Training.

#### 4.3.1.5 Algorithm-Level Computational Optimisations

Table 4.5 shows the average training time per epoch for each dataset and the speedup obtained compared to the Naive baseline. The speedup reported here and in the following sections is computed using Equation 4.1. Compared to the baseline Naive method, most methods introduce minimal overhead, with the notable exception of PMR, which nearly doubles the training time due to the additional cost of prototype computation. Among our proposed methods, Alternating Training (**ALT**) maintains nearly similar training time to the Naive baseline, which aligns with its design goal of avoiding significant computational overhead. **AMST**, which selectively skips encoder backpropagation via control mechanisms, significantly reduces training time,

Table 4.4: Train Bias Metrics and number of trained epochs across different datasets for all modalities. Alt: Alternating Training; AMST: Alternating Multimodal Skip Training; Ent: Entropy Training; Rel: Reliability Training.

| | | CREMA-D | | AVE | | MVSA | | IEMO2 | | IEMO3 | | | UR-FUNNY | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | A | V | A | V | T | V | A | V | A | V | T | A | V | T |
| Relative Bias | Alt | 0.68 | 0.32 | 0.66 | 0.34 | 0.75 | 0.25 | 0.76 | 0.24 | 0.37 | 0.12 | 0.50 | 0.23 | 0.02 | 0.76 |
| | AMST | 0.51 | 0.49 | 0.63 | 0.37 | 0.57 | 0.43 | 0.66 | 0.34 | 0.38 | 0.21 | 0.41 | 0.42 | 0.04 | 0.54 |
| | Ent | 0.51 | 0.49 | 0.51 | 0.49 | 0.58 | 0.42 | 0.50 | 0.50 | 0.33 | 0.32 | 0.34 | 0.34 | 0.09 | 0.57 |
| | Rel | 0.57 | 0.43 | 0.55 | 0.45 | 0.62 | 0.38 | 0.59 | 0.41 | 0.35 | 0.22 | 0.43 | 0.32 | 0.04 | 0.64 |
| Trained Epochs | Alt | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| | AMST | 20 | 100 | 50 | 100 | 10 | 100 | 26 | 100 | 26 | 100 | 10 | 18 | 100 | 10 |
| | Ent | 15 | 100 | 17 | 100 | 4 | 100 | 14 | 100 | 18 | 100 | 5 | 9 | 100 | 2 |
| | Rel | 18 | 100 | 22 | 100 | 11 | 100 | 15 | 100 | 12 | 100 | 14 | 7 | 100 | 8 |

Table 4.5: Comparisons on average training time per epoch (in seconds, on single A100, averaged over 100 epochs) between state-of-the-art methods and Alternating, AMST, Entropy and Reliability Training. Speedup compared to the Naive baseline is shown in parentheses. (Concat Fusion, except MLA)

| Method | CREMA-D | AVE | MVSA | IEMOCAP | UR-FUNNY |
|---|---|---|---|---|---|
| Naive | 13.6 | 8.0 | 7.9 | 15.9 | 23.9 |
| MSLR | 13.6 | 8.0 | 7.8 | 16.3 | 24.0 |
| OGM | 14.2 | 8.8 | 8.5 | 16.4 | 25.6 |
| PMR | 26.2 | 36.7 | 9.8 | – | – |
| MLA | 13.9 | 9.3 | 7.9 | 15.9 | 23.8 |
| ALT | 13.9 (1.0×) | 9.2 (0.9×) | 7.9 (1.0×) | 15.5 (1.0×) | 24.0 (1.0×) |
| AMST | 9.9 (1.4×) | 6.8 (1.2×) | 4.9 (1.6×) | 10.9 (1.5×) | 15.0 (1.6×) |
| ENT | 11.2 (1.2×) | 5.9 (1.4×) | 5.5 (1.4×) | 13.5 (1.2×) | 25.8 (0.9×) |
| REL | 11.1 (1.2×) | 5.9 (1.4×) | 5.6 (1.4×) | 13.3 (1.2×) | 21.5 (1.1×) |

reaching 1.2× to 1.6× speedup across all datasets. For the Entropy (**ENT**) and Reliability (**REL**) Training methods, we also observe considerable improvements, typically 1.2× to 1.4× speedup, except on the UR-FUNNY dataset, where slightly longer training time is due to an increased number of epochs. These results prove that our methods not only match or outperform the accuracy of the MLA baseline but also substantially improve training efficiency.

$$\text{Speedup} = \frac{T_{\text{unoptimized}}}{T_{\text{optimized}}} \tag{4.1}$$

### 4.3.2 System level Optimisations

In this section, we present the analyse of techniques used to accelerate our training process, focusing on three key aspects: (1) the impact of hardware-specific floating-point formats (TF32 and BF16); (2) the effect of dataloader pipeline warm-up time

across various datasets and devices; and (3) the comparison among three different parallelisation strategies: Data Parallelism (DP), Distributed Data Parallelism (DDP), and Branch-Level Parallelism (BLP).

#### 4.3.2.1 Performance Improvement with TF32 and BF16

Tables 4.6 and 4.7 present the performance improvement achieved by enabling TF32 matrix multiplication on A100 and A40 GPUs, respectively. Notably, TF32 support for convolution operations is enabled by default in PyTorch. As a result, encoders for the visual and audio modalities, which are based on CNNs, show minimal performance change. In contrast, the text modality, which relies heavily on matrix multiplication in transformer-based encoders, benefits significantly from this optimisation.

On the A100 GPU, enabling TF32 matrix multiplication results in a speedup ranging from 2.61× on the IEMOCAP dataset to 3.33× on the MVSA dataset. Similarly, on the A40 GPU, speedups range from 1.29× to 1.65× across different datasets. These variations are primarily due to differences in the average text length and the number of samples across datasets, both of which influence the computational workload of the textual encoder.

Tables 4.8 and 4.9 present the performance comparison between BF16 and TF32 on A100 across various datasets and modalities, using a batch size of 64. Overall, BF16 achieves similar performance to TF32, with speedup ratios generally around 1.0. This limited improvement is likely due to the batch size being relatively small, where the computational demand does not fully utilise the benefits of BF16's higher throughput. Since the majority of our experiments are conducted with a batch size of 64, TF32 is adopted as the default precision format throughout our study.

Table 4.6: Comparison of single-encoder training time per epoch with and without TF32 matrix multiplication on A100 (seconds; average over 10 epochs; batch size = 64).

| Dataset | Modality | No TF32 | TF32 | Speedup |
|---|---|---|---|---|
| CREMAD | V | 7.54 | 7.57 | 0.99× |
| | A | 6.26 | 6.25 | 1.00× |
| AVE | V | 4.52 | 4.53 | 1.00× |
| | A | 4.27 | 4.33 | 0.99× |
| MVSA | V | 1.72 | 1.71 | 1.01× |
| | T | 20.06 | 6.03 | **3.33×** |
| IEMOCAP | V | 6.90 | 6.90 | 1.00× |
| | A | 6.21 | 6.09 | 1.02× |
| | T | 15.06 | 5.76 | **2.61×** |
| UF | V | 10.17 | 10.29 | 0.99× |
| | A | 8.34 | 8.41 | 0.99× |
| | T | 22.48 | 6.98 | **3.22×** |

Table 4.7: Comparison of single-encoder training time per epoch with and without TF32 matrix multiplication on A40 (seconds; average over 10 epochs; batch size = 64).

| Dataset | Modality | No TF32 | TF32 | Speedup |
|---|---|---|---|---|
| CREMAD | V | 14.42 | 14.81 | 0.97× |
| | A | 13.01 | 14.69 | 0.89× |
| AVE | V | 9.51 | 9.43 | 1.01× |
| | A | 10.62 | 8.55 | **1.24×** |
| MVSA | V | 3.82 | 4.86 | 0.79× |
| | T | 22.49 | 14.18 | **1.59×** |
| IEMOCAP | V | 14.74 | 13.82 | 1.07× |
| | A | 12.59 | 12.57 | 1.00× |
| | T | 15.90 | 12.31 | **1.29×** |
| UF | V | 19.02 | 18.60 | 1.02× |
| | A | 17.37 | 16.55 | 1.05× |
| | T | 26.09 | 15.80 | **1.65×** |

Table 4.8: Comparison of CNN-based visual encoder training time with BF16 and TF32 on A100 (seconds; average over 10 epochs; batch size = 64).

| Dataset | BF16 | TF32 | Speedup |
|---|---|---|---|
| CREMAD | 6.71 | 7.57 | **1.13×** |
| AVE | 4.41 | 4.53 | **1.03×** |
| MVSA | 1.75 | 1.71 | 0.98× |
| IEMOCAP | 6.56 | 6.90 | 0.95× |
| UF | 8.94 | 10.29 | **1.15×** |

Table 4.9: Comparison of training time for Transformer-based textual encoder using BF16 and TF32 on A100 (seconds; average over 10 epochs; batch size = 64).

| Dataset | BF16 | TF32 | Speedup |
|---|---|---|---|
| MVSA | 5.05 | 6.03 | **1.19×** |
| IEMOCAP | 6.27 | 5.76 | 0.92× |
| URFUNNY | 7.43 | 6.98 | 0.94× |

#### 4.3.2.2 Impact of Data Loader on GPU Utilisation

**On Different Datasets**

Table 4.10 shows the comparison between the standard PyTorch Data Loader and the optimised Data Loader on the A100 GPU with a batch size of 64. The table reports the average training and validation time per epoch (in seconds) and GPU

utilisation for different datasets. The results indicate that the optimised Data Loader consistently reduces training time across all datasets, achieving speedup ratios ranging from 1.12 to 1.37. Additionally, GPU utilisation improves significantly, with increases up to 23.4%, indicating that the optimisation effectively helps data loading bottlenecks and better leverages the hardware resources. Figures 4.1 and 4.2 display the GPU utilisation over time. The gaps between epochs observable in Figure 4.1 disappear in Figure 4.2, where the utilisation curve also becomes more stable.

Table 4.10: Comparison of training and validation time with GPU utilisation using Standard (Std) vs. Optimised (Opt) dataloaders on A100 (batch size = 64).

| Dataset | Std Time (s) | Opt Time (s) | Speedup | Std Util. (%) | Opt Util. (%) | Util. Gain |
|---|---|---|---|---|---|---|
| CREMAD | 16.03 | 13.60 | **1.18×** | 76.58 | 90.55 | +13.97% |
| AVE | 10.95 | 8.00 | **1.37×** | 62.50 | 85.92 | +23.42% |
| MVSA | 9.00 | 7.94 | **1.13×** | 81.29 | 93.25 | +11.96% |
| IEMOCAP | 19.50 | 16.40 | **1.19×** | 77.60 | 92.07 | +14.47% |
| URFUNNY | 27.29 | 24.32 | **1.12×** | 83.04 | 93.11 | +10.07% |

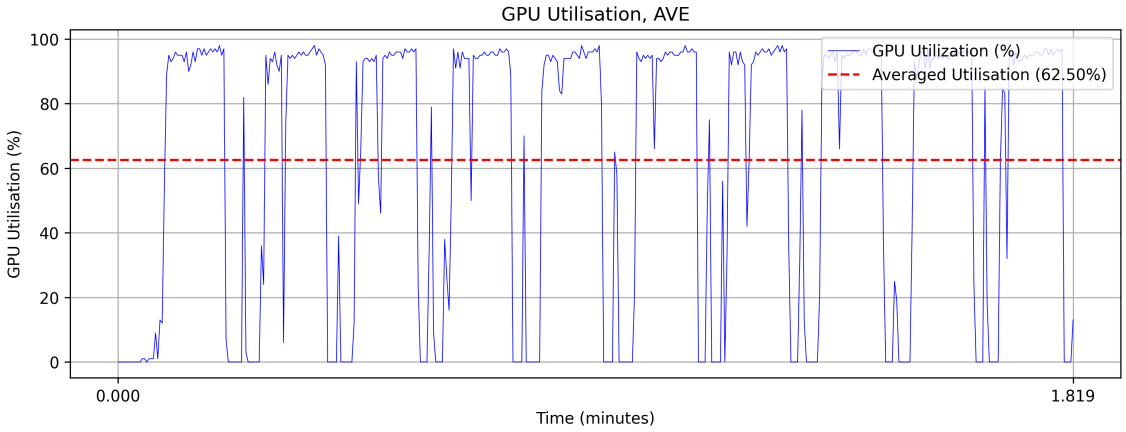



Figure 4.1: GPU utilisation on the AVE dataset using a non-optimised data loader.

### On Different Batch Size

Table 4.11 summarises the training performance across three typical batch sizes, using the standard and optimised Data Loader on the A100 GPU. The results are shown for two representative datasets: CREMA-D and IEMOCAP.

We observe consistent performance improvements while using the optimised Data Loader across all tested batch sizes. Specifically, the training time is reduced while GPU utilisation increases significantly. For CREMA-D, the optimised loader improves utilisation from 80.10% to 87.86% at batch size 32, and from 65.75% to 91.03% at batch size 128, while reducing the training time by up to 1.39×. A similar trend is seen in IEMOCAP, where the highest GPU utilisation gain (25.21%) and speedup (1.36×) are also achieved at batch size 128.

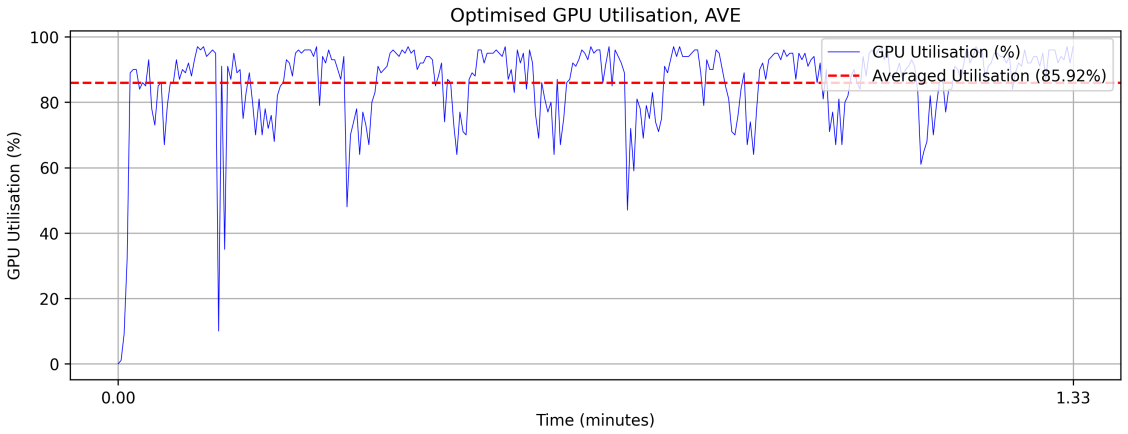



Figure 4.2: GPU utilisation on the AVE dataset using an optimised data loader.

These results suggest that the optimised Data Loader consistently enhances GPU utilisation and training efficiency, and its benefits become more noticeable with larger batch sizes.

Table 4.11: Comparison of training and validating time with GPU utilisation using Standard (Std) and Optimised (Opt) dataloader on an A100 GPU across different batch sizes.

| Dataset | Batch Size | Loader Type | Time (s) | GPU Util. (%) | Speedup | Util. Gain (%) |
|---|---|---|---|---|---|---|
| CREMA-D | 32 | Std | 15.90 | 80.10 | | |
| | | Opt | 14.57 | 87.86 | 1.09× | +7.76 |
| | 64 | Std | 16.03 | 76.58 | | |
| | | Opt | 13.60 | 90.55 | 1.18× | +13.97 |
| | 128 | Std | 18.44 | 65.75 | | |
| | | Opt | 13.29 | 91.03 | 1.39× | +25.28 |
| IEMOCAP | 32 | Std | 19.30 | 80.39 | | |
| | | Opt | 17.50 | 89.61 | 1.10× | +9.22 |
| | 64 | Std | 19.50 | 77.60 | | |
| | | Opt | 16.40 | 92.07 | 1.19× | +14.47 |
| | 128 | Std | 22.65 | 68.93 | | |
| | | Opt | 16.64 | 94.14 | 1.36× | +25.21 |

### On Different Devices

Table 4.12 compares the performance of standard and optimised Data Loaders across two different GPU devices (A40 and A100) on the AVE and UR-FUNNY datasets. The optimised Data Loader consistently outperforms the standard version on both devices regarding training-validation time and GPU utilisation. On the AVE dataset, the optimised loader reduces runtime by 1.23× on A40 and 1.37× on A100, with GPU utilisation gains of 17.81% and 23.42%, respectively. These results highlight the benefits of optimisation, especially on newer and more capable devices like A100. For the UR-FUNNY dataset, while the absolute runtime improvement is smaller, there is still a noticeable speedup: 1.06× on A40 and 1.12× on A100, accompanied

by GPU utilisation improvements of 6.20% and 10.07%, respectively. The relatively small time gain can be attributed to the high baseline time of the dataset.

Table 4.12: Comparison of training and validating time with GPU utilisation using Standard (Std) and Optimised (Opt) dataloader across different GPU devices (batch size = 64).

| Dataset | Device | Loader Type | Time (s) | GPU Util. (%) | Speedup | Util. Gain |
|---|---|---|---|---|---|---|
| AVE | A40 | Std | 17.71 | 75.73 | | |
| | | Opt | 14.39 | 93.54 | 1.23× | +17.81 |
| | A100 | Std | 10.95 | 62.50 | | |
| | | Opt | 8.00 | 85.92 | 1.37× | +23.42 |
| UR-FUNNY | A40 | Std | 51.43 | 91.22 | | |
| | | Opt | 48.52 | 97.42 | 1.06× | +6.20 |
| | A100 | Std | 27.29 | 83.04 | | |
| | | Opt | 24.32 | 93.11 | 1.12× | +10.07 |

#### 4.3.2.3 Comparison of Speedup Using Different Parallelism Strategies

**On Different Datasets**

Table 4.13 shows the speedup gained by Data Parallelism (DP), Distributed Data Parallelism (DDP), and Branch-Level Parallelism (BLP) methods across different datasets, using a batch size of 64. It can be noticed that DP provides only limited speedup at this batch size, ranging from 1.01× to 1.22×. DDP achieves the highest speedup, from 1.67× to 2.12×. Our method, BLP, also shows significant improvement over DP, with speedups ranging from 1.16× to 1.68×. Furthermore, BLP delivers more consistent speedup across different datasets, averaging around 1.50×, except for the MVSA dataset. This discrepancy can be attributed to the unbalanced encoder workloads in the MVSA dataset, as shown in Table 4.6.

Table 4.13: Training and validating time (in seconds) and speed up with batch size 64 on A100, using optimised dataloader.

| Dataset | Single GPU Time | DP Time | DDP Time | BLP Time | DP Speedup | DDP Speedup | BLP Speedup |
|---|---|---|---|---|---|---|---|
| CREMA-D | 13.60 | 13.009 | 8.147 | 9.000 | 1.05× | 1.67× | 1.51× |
| AVE | 8.00 | 7.687 | 4.778 | 5.290 | 1.04× | 1.67× | 1.51× |
| MVSA | 7.94 | 6.519 | 4.865 | 6.837 | 1.22× | 1.63× | 1.16× |
| IEMOCAP | 16.40 | 16.244 | 7.861 | 9.817 | 1.01× | 2.09× | 1.67× |
| UF | 24.32 | 23.649 | 11.451 | 14.447 | 1.03× | 2.12× | 1.68× |

**On Different Batch Sizes**

Table 4.14 shows the speedup of the three parallelisation strategies across different batch sizes. The DP method shows limited scalability at smaller batch sizes. Specifically, at a batch size of 32, DP suffers overheads that outweigh its benefits, resulting in a slowdown with a speedup of only 0.72× on CREMA-D and 0.63× on IEMOCAP. However, DP performance improves as the batch size increases, achieving speedups of 1.37× and 1.52× at batch size 128 for CREMA-D and IEMOCAP, respectively.

DDP consistently gives the highest speedup across all batch sizes and datasets, with speedups ranging from 1.56× to 2.43×. Its performance is relatively stable and only slightly affected by the batch size.

Our proposed BLP method displays notable improvement over DP, consistently outperforming it at all batch sizes. BLP achieves speedups close to those of DDP, ranging from 1.41× to 1.90×, highlighting its effectiveness as a possible selection.

Table 4.14: Training time (in seconds) and speedup on different batch sizes with optimised dataloader.

| Dataset | Batch Size | Single GPU (s) | DP (s) | DDP (s) | BLP (s) | DP Speedup | DDP Speedup | BLP Speedup |
|---|---|---|---|---|---|---|---|---|
| CREMA-D | 32 | 14.570 | 20.161 | 9.332 | 10.205 | 0.72× | 1.56× | 1.43× |
| | 64 | 13.600 | 13.009 | 8.147 | 9.000 | 1.05× | 1.67× | 1.51× |
| | 128 | 13.287 | 9.722 | 7.719 | 8.396 | 1.37× | 1.72× | 1.58× |
| IEMOCAP | 32 | 17.502 | 27.921 | 10.800 | 12.390 | 0.63× | 1.62× | 1.41× |
| | 64 | 16.400 | 16.244 | 7.861 | 9.817 | 1.01× | 2.09× | 1.72× |
| | 128 | 16.636 | 10.939 | 6.858 | 8.751 | 1.52× | 2.43× | 1.90× |

**On Different Devices**

Table 4.15 presents the training and validation times along with speedups of DP, DDP, and BLP methods evaluated on an NVIDIA A40 GPU with a batch size of 64, complementing the results previously reported on the A100.

Similar to the A100, both BLP and DP strategies achieve significant speedup over single-GPU training on the A40. DDP remains the most effective method, delivering speedups ranging from 1.61× to 2.24× across the evaluated datasets. Notably, DP performs better on the A40 compared to the A100, with speedups between 1.32× and 1.42×. This can be explained by the fact that the A100 is more powerful than the A40; thus, on the less powerful A40, the gains from utilising additional computing resources outweigh the communication overhead more effectively.

Importantly, our proposed BLP method again demonstrates competitive performance on the A40, achieving speedups close to those of DDP on most datasets, for example, 1.64× on CREMA-D, 1.58× on AVE, and over 2.0× on both IEMOCAP and URFUNNY. This confirms that BLP maintains robust acceleration capability across these two hardware platforms.

The exception is the MVSA dataset, where BLP attains a comparatively lower speedup of 1.17×, likely due to workload imbalance among encoders, consistent with observations made on the A100.

Table 4.15: Training and validating time (in seconds) and speed of three parallelism methods on A40 (batch size = 64)

| Dataset | Single (s) | DP (s) | DDP (s) | BLP (s) | DP Speedup | DDP Speedup | BLP Speedup |
|---|---|---|---|---|---|---|---|
| CREMAD | 25.15 | 18.099 | 15.074 | 15.337 | 1.39× | 1.67× | 1.64× |
| AVE | 14.39 | 10.795 | 8.234 | 9.103 | 1.33× | 1.75× | 1.58× |
| MVSA | 17.06 | 12.888 | 10.580 | 14.641 | 1.32× | 1.61× | 1.17× |
| IEMOCAP | 32.60 | 22.984 | 14.532 | 15.980 | 1.42× | 2.23× | 2.04× |
| URFUNNY | 48.52 | 34.352 | 21.613 | 23.807 | 1.41× | 2.24× | 2.04× |

#### 4.3.2.4 Overall Speed Up

Table 4.16 compares training and validation times on the A100 GPU with default settings versus all optimisations applied (optimised dataloader, BLP, and TF32). The optimised setup achieves significant speedups across all datasets, ranging from 1.86× to 3.51×, demonstrating the effectiveness of the combined optimisations in reducing training time.

Table 4.16: Training and validation time (in seconds) and speedup on A100 using Default settings and All Optimisations (Optimised Dataloader + BLP + TF32)

| Dataset | Default Time (s) | Optimized Time (s) | Speedup |
|---|---|---|---|
| CREMAD | 16.750 | 9.000 | 1.86× |
| AVE | 11.259 | 5.290 | 2.13× |
| MVSA | 23.997 | 6.837 | 3.51× |
| IEMOCAP | 30.323 | 9.817 | 3.09× |
| URFUNNY | 42.728 | 14.447 | 2.96× |

# 5

# Conclusion

The goal of this research was to enhance the performance of multimodal models while improving training efficiency, scalability, and energy consumption. In the following sections, we summarise how we achieved our goals, the potential limitations of our work and ethical considerations.

## 5.1 Achieving Research Goals

The following list describes each of our proposed research goals and how and where we achieved them:

- **Understanding the Mathematical Foundations of the Modality Imbalance Problem**: In Section 2.3, we provide a mathematical explanation to the core issue of Modality Imbalance and conclude that the problem mainly arises from gradient disappearance due to joint modality optimisation with a single training loss function.

- **Identifying and Defining the New Modality Prediction Bias Problem**: In Section 2.6.1, we introduce this new problem, where models disproportionately rely on specific modalities during inference, even when others are equally informative. It stems from the overtraining of faster-learning modalities, which leads to overconfidence due to their tendency to overfit.

- **Formally Evaluating Modality Imbalance and Modality Prediction Bias Through Targeted Metrics**: In Section 3.1, we provide targeted metrics that allow us to measure how a model is affected by Modality Imbalance, how much each modality comparatively contributes towards the final prediction, providing us insight into the Prediction Bias, and finally the reliability of each modality, describing how well they generalise to unseen data.

- **Building on the Strengths of Previous SOTA Methods, Extending them to Multiple Late Fusion Structures**: In Section 3.2, we generalise Alternating Training, initially proposed in [15], to multiple late fusion strategies, such as concatenation, summation, gated and FiLM fusion. This generalisation allows all of the previous fusion structures not to suffer from the gradient disappearance problem, which is the main cause of Modality Imbalance.

- **Introducing Novel Methods that Effectively Address both Modality Imbalance and Modality Prediction Bias**: In Section 3.3, we introduce three different training methods that address both Modality Imbalance and Modality Prediction Bias. The first method, Alternating Multimodal Skip Training (AMST), despite requiring fine-tuning of hyperparameters, proves to be effective when the optimal values for these hyperparameters are found. Secondly, we introduce Entropy Training, which aims to balance the confidence across all modalities. However, it only proves effective when the modalities have comparable individual performance. As the last method, we introduce Reliability Training, which prioritises the training of the best-generalising modality at each training epoch. This technique proved to be the most robust and effective across all tasks, being the central approach of this work.

- **Analysing and Improving Training Efficiency**: A system-aware optimisation that overcomes the efficiency problem of the current multi-GPU training method, which ignores the structure of the model and the mismatch between high-power GPU devices and the data loading pipeline.

## 5.2 Limitations and Future Work

This work focuses on addressing modality imbalance and improving its understanding by introducing theoretical foundations and dedicated metrics for its analysis and measurement. We extend Alternating Training to support various late-fusion strategies and introduce the novel problem of modality prediction bias, defining, measuring, and mitigating it through various training approaches. Our proposed methods effectively handle both modality imbalance and prediction bias and generalise well across the different datasets listed in Section 1.5.

While our proposed methods support a wide range of late-fusion strategies, they have not been evaluated with more than three modalities. As more modalities are introduced, the learning process is expected to become increasingly complex due to the growing difficulty of managing inter-modal interactions.

Furthermore, our experiments solely focus on late fusion structures and do not evaluate imbalance in early or intermediate fusion mechanisms. These fusion stages rely on completely different behaviours from those of late fusion, which allows the development of new specific studies regarding Modality Imbalance and Prediction Bias for these fusion structures.

In addition, our experiments were conducted under clean data conditions, without considering noisy or missing modalities, which may require specific adjustments to the optimisation pattern. This could mean introducing new metrics to guide the training strategy according to the data conditions.

Finally, while our training techniques improve performance on Gated and FiLM fusion, we only provide a deep explanation of how modality prediction bias works with linear classifiers used in symmetric fusion structures, such as summation and

concatenation fusion. Further investigation is needed to understand how modality prediction bias impacts Gated and FiLM fusion. A starting point for this analysis involves examining the gating parameters of each fusion mechanism to understand how they behave under different training techniques. Since each training technique adjusts each modality's confidence in a different way, we expect these parameters to respond accordingly, leading to notable changes in their behaviour.

All in all, the previously mentioned areas present opportunities for future exploration and refinement, which we were unable to pursue due to time constraints.

## 5.3 Broader Impact, Ethical Considerations and Sustainability

In this work, we proposed different optimisations to multimodal models focusing on classification tasks and supervised learning. Our methods do not focus on any specific domain of machine learning, such as Emotion Recognition or Sentiment Analysis, but generalise to any classification task. Our proposed method solely operates based on the data it receives. We specifically selected publicly available datasets and analysed existing methods without violating any privacy regulations. However, there is always a risk that one's work could be misused. We believe our proposed multimodal optimisation techniques have many potential beneficial applications, such as in medical diagnosis.

In addition, one of the main goals of this work is to optimise the training process in multimodal models, both from algorithm and system-level perspectives, with a focus on computational efficiency. The application of these improvements to general multimodal models can help reduce computational demands, thereby lowering energy consumption and minimising environmental impact.

# Bibliography


[1] A. I. Middya, B. Nag, and S. Roy, "Deep learning based multimodal emotion recognition using model-level fusion of audio–visual modalities," *Knowledge-based systems*, vol. 244, p. 108 580, 2022.

[2] J. N. Acosta, G. J. Falcone, P. Rajpurkar, and E. J. Topol, "Multimodal biomedical ai," *Nature medicine*, vol. 28, no. 9, pp. 1773–1784, 2022.

[3] Y. Tang, H. He, Y. Wang, Z. Mao, and H. Wang, "Multi-modality 3d object detection in autonomous driving: A review," *Neurocomputing*, vol. 553, p. 126 587, 2023.

[4] H. Cao, D. G. Cooper, M. K. Keutmann, R. C. Gur, A. Nenkova, and R. Verma, "Crema-d: Crowd-sourced emotional multimodal actors dataset," *IEEE Transactions on Affective Computing*, vol. 5, no. 4, pp. 377–390, 2014. DOI: 10.1109/TAFFC.2014.2336244.

[5] W. Wang, D. Tran, and M. Feiszli, "What makes training multi-modal classification networks hard?" In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 12 692–12 702. DOI: 10.1109/CVPR42600.2020.01271.

[6] C. Du, T. Li, Y. Liu, *et al.*, "Improving multi-modal learning with uni-modal teachers.," en-US, *arXiv: Learning,arXiv: Learning*, Jun. 2021.

[7] X. Peng, Y. Wei, A. Deng, D. Wang, and D. Hu, " Balanced Multimodal Learning via On-the-fly Gradient Modulation," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2022, pp. 8228–8237. DOI: 10.1109/CVPR52688.2022.00806. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR52688.2022.00806.

[8] H. Li, X. Li, P. Hu, Y. Lei, C. Li, and Y. Zhou, "Boosting multi-modal model performance with adaptive gradient modulation," in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 22 157–22 167. DOI: 10.1109/ICCV51070.2023.02030.

[9] Y. Fan, W. Xu, H. Wang, J. Wang, and S. Guo, "Pmr: Prototypical modal rebalance for multimodal learning," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 20 029–20 038. DOI: 10.1109/CVPR52729.2023.01918.

[10] Y. Sun, S. Mai, and H. Hu, "Learning to balance the learning rates between various modalities via adaptive tracking factor," *IEEE Signal Processing Letters*, vol. 28, pp. 1650–1654, 2021. DOI: 10.1109/LSP.2021.3101421.

[11] N. Wu, S. Jastrzebski, K. Cho, and K. J. Geras, "Characterizing and overcoming the greedy nature of learning in multi-modal deep neural networks," in *Proceedings of the 39th International Conference on Machine Learning*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., ser. Proceedings of Machine Learning Research, vol. 162, PMLR, 17–23 Jul 2022, pp. 24 043–24 055. [Online]. Available: https://proceedings.mlr.press/v162/wu22d.html.

[12] Y. Yao and R. Mihalcea, "Modality-specific learning rates for effective multimodal additive late-fusion," in *Findings of the Association for Computational Linguistics: ACL 2022*, S. Muresan, P. Nakov, and A. Villavicencio, Eds., Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 1824–1834. DOI: 10.18653/v1/2022.findings-acl.143. [Online]. Available: https://aclanthology.org/2022.findings-acl.143.

[13] X. Lin, S. Wang, R. Cai, *et al.*, "Suppress and rebalance: Towards generalized multi-modal face anti-spoofing," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2024, pp. 211–221.

[14] Y. Wei, R. Feng, Z. Wang, and D. Hu, "Enhancing multimodal cooperation via sample-level modality valuation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 27 338–27 347.

[15] X. Zhang, J. Yoon, M. Bansal, and H. Yao, " Multimodal Representation Learning by Alternating Unimodal Adaptation," in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2024, pp. 27 446–27 456. DOI: 10.1109/CVPR52733.2024.02592. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR52733.2024.02592.

[16] E. J. Martinez-Noriega, C. Peng, and R. Yokota, "High-performance data loader for large-scale data processing," *Electronic Imaging*, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:270291745.

[17] I. Ofeidis, D. Kiedanski, and L. Tassiulas, "An overview of the data-loader landscape: Comparative performance analysis," in *2024 IEEE International Conference on Big Data (BigData)*, 2024, pp. 360–367. DOI: 10.1109/BigData62323.2024.10825421.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, ISSN: 0001-0782. DOI: 10.1145/3065386. [Online]. Available: https://doi.org/10.1145/3065386.

[19] J. Dean, G. S. Corrado, R. Monga, *et al.*, "Large scale distributed deep networks," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12, Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1223–1231.

[20] C.-C. Chen, C.-L. Yang, and H.-Y. Cheng, "Efficient and robust parallel dnn training through model parallelism on multi-gpu platform," *ArXiv*, vol. abs/1809.02839, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:52186266.

[21] S. Pal, E. Ebrahimi, A. Zulfiqar, *et al.*, "Optimizing multi-gpu parallelization strategies for deep learning training," *IEEE Micro*, vol. 39, no. 5, pp. 91–101, 2019. DOI: 10.1109/MM.2019.2935967.

[22] Y. Tian, J. Shi, B. Li, Z. Duan, and C. Xu, "Audio-visual event localization in unconstrained videos," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Sep. 2018.

[23] T. Niu, S. Zhu, L. Pang, and A. El Saddik, "Sentiment analysis on multi-view social data," in *MultiMedia Modeling: 22nd International Conference, MMM 2016, Miami, FL, USA, January 4-6, 2016, Proceedings, Part II 22*, Springer, 2016, pp. 15–27.

[24] C. Busso, M. Bulut, C.-C. Lee, *et al.*, "Iemocap: Interactive emotional dyadic motion capture database," *Language resources and evaluation*, vol. 42, pp. 335–359, 2008.

[25] M. K. Hasan, W. Rahman, A. Zadeh, J. Zhong, M. I. Tanveer, L.-P. Morency, *et al.*, "Ur-funny: A multimodal language dataset for understanding humor," *arXiv preprint arXiv:1904.06618*, 2019.

[26] P. P. Liang, A. Zadeh, and L.-P. Morency, "Foundations & trends in multimodal machine learning: Principles, challenges, and open questions," *ACM Comput. Surv.*, vol. 56, no. 10, Jun. 2024, ISSN: 0360-0300. DOI: 10.1145/3656580. [Online]. Available: https://doi.org/10.1145/3656580.

[27] A. Owens and A. A. Efros, "Audio-visual scene analysis with self-supervised multisensory features," en-US, in *Computer Vision – ECCV 2018,Lecture Notes in Computer Science*. Dec. 2017, pp. 639–658. DOI: 10.1007/978-3-030-01231-1_39. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-01231-1_39.

[28] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville, "Film: Visual reasoning with a general conditioning layer," en-US, *Proceedings of the AAAI Conference on Artificial Intelligence*, Jun. 2022. DOI: 10.1609/aaai.v32i1.11671. [Online]. Available: http://dx.doi.org/10.1609/aaai.v32i1.11671.

[29] D. Kiela, E. Grave, A. Joulin, and T. Mikolov, "Efficient large-scale multi-modal classification," en-US, *Proceedings of the AAAI Conference on Artificial Intelligence*, Nov. 2022. DOI: 10.1609/aaai.v32i1.11945. [Online]. Available: http://dx.doi.org/10.1609/aaai.v32i1.11945.

[30] P. P. Liang, A. Zadeh, and L.-P. Morency, "Foundations & trends in multimodal machine learning: Principles, challenges, and open questions," *ACM Computing Surveys*, vol. 56, no. 10, pp. 1–42, 2024.

[31] T. McGrath, A. Kapishnikov, N. Tomašev, *et al.*, "Acquisition of chess knowledge in alphazero," en-US, *Proceedings of the National Academy of Sciences*, Nov. 2022. DOI: 10.1073/pnas.2206625119. [Online]. Available: http://dx.doi.org/10.1073/pnas.2206625119.

[32] A. Mikołajczyk and M. Grochowski, "Data augmentation for improving deep learning in image classification problem," in *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, 2018, pp. 117–122. DOI: 10.1109/IIPHDW.2018.8388338.

[33] D. S. Park, W. Chan, Y. Zhang, *et al.*, "Specaugment: A simple data augmentation method for automatic speech recognition," *arXiv preprint arXiv:1904.08779*, 2019.

[34] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014, ISSN: 1532-4435.

[35] A. Y. Ng, "Feature selection, l1 vs. l2 regularization, and rotational invariance," in *Proceedings of the Twenty-First International Conference on Machine Learning*, ser. ICML '04, Banff, Alberta, Canada: Association for Computing Machinery, 2004, p. 78, ISBN: 1581138385. DOI: 10.1145/1015330.1015435. [Online]. Available: https://doi.org/10.1145/1015330.1015435.

[36] C3SE, *Alvis - chalmers gpu cluster*, 2025. [Online]. Available: https://www.c3se.chalmers.se/about/Alvis/.

[37] J. Ansel, E. Yang, H. He, *et al.*, "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation," in *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*, ACM, Apr. 2024. DOI: 10.1145/3620665.3640366. [Online]. Available: https://pytorch.org/assets/pytorch2-2.pdf.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[39] A. J. Peña, P. Valero-Lara, and M. Jordà, "Filling the performance gap in convolution implementations for nvidia gpus," in *GPU Technology Conference (GTC)*, Barcelona Supercomputing Center, San Jose, CA, 2019. [Online]. Available: https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9218-filling-the-performance-gap-in-convolution-implementations-for-nvidia-gpus.pdf.

[40] Y. Liu, M. Ott, N. Goyal, *et al.*, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[41] P. Micikevicius, *Mixed-precision training of deep neural networks*, 2017. [Online]. Available: https://developer.nvidia.com/blog/mixed-precision-training-deep-neural-networks/.

[42] NVIDIA Corporation, "Nvidia a100 tensor core gpu architecture," NVIDIA, Tech. Rep., 2020. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf.

[43] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of big data*, vol. 6, no. 1, pp. 1–48, 2019.

[44] S. Chilamkurthy, *Writing custom datasets, dataloaders and transforms*, 2025. [Online]. Available: https://pytorch.org/tutorials/beginner/data_loading_tutorial.html.

[45] *Global interpreter lock*, https://docs.python.org/3/glossary.html#term-global-interpreter-lock, Python 3 Glossary, 2024.

[46] N. Corporation, *Cuda c programming guide*, 2023. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#page-locked-memory.

[47] PyTorch, *Data loading and processing*, Accessed: 2025-04-04, n.d. [Online]. Available: https://pytorch.org/docs/stable/data.html.

[48] PyTorch Contributors, *Optional: Data parallelism*, 2017. [Online]. Available: https://pytorch.org/tutorials/beginner/blitz/data_parallel_tutorial.html.

[49] PyTorch Contributors, *Getting started with distributed data parallel*, 2019. [Online]. Available: https://pytorch.org/tutorials/intermediate/ddp_tutorial.html.

[50] H. Wang, Y. Wang, Z. Zhou, *et al.*, "Cosface: Large margin cosine loss for deep face recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 5265–5274.

[51] R. Xu, R. Feng, S.-X. Zhang, and D. Hu, "Mmcosine: Multi-modal cosine loss towards balanced audio-visual fine-grained learning," in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2023, pp. 1–5.

[52] Y. Huang, Y. Cheng, A. Bapna, *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *Advances in neural information processing systems*, vol. 32, 2019.

[53] B. McFee, C. Raffel, D. Liang, *et al.*, "Librosa: Audio and music signal analysis in python," en-US, in *Proceedings of the 14th Python in Science Conference,Proceedings of the Python in Science Conference*, Dec. 2014. DOI: 10.25080/majora-7b98e3ed-003.

[54] S.-i. Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing*, vol. 5, no. 4-5, pp. 185–196, 1993.

# A

# Appendix 1

## A.1 Results of 3 Seeds for All Methods and Datasets

Table A.1: Unimodal Accuracies and IBF scores for Single modality training in a unimodal scenario and in a Naive multimodal scenario (test set). Mean ± standard error over 3 random seeds. IBF is calculated based on the mean values.

| Dataset | | Single | Naive |
|---|---|---|---|
| CREMA-D | Visual | 60.5 ± 0.75 | 22.4 ± 1.37 |
| | Audio | 59.1 ± 0.49 | 55.1 ± 0.49 |
| | IBF | 0 | 0.349 |
| AVE | Visual | 28.5 ± 0.64 | 14.4 ± 0.17 |
| | Audio | 55.3 ± 0.56 | 57.2 ± 0.67 |
| | IBF | 0 | 0.247 |
| MVSA | Visual | 57.7 ± 1.21 | 56.9 ± 0.24 |
| | Text | 70.8 ± 0.10 | 68.3 ± 0.07 |
| | IBF | 0 | 0.025 |
| IEMOCAP | Visual | 46.5 ± 0.53 | 26.7 ± 1.48 |
| | Audio | 49.2 ± 1.41 | 43.1 ± 0.47 |
| | Text | 62.3 ± 0.30 | 58.8 ± 0.34 |
| | IBF | 0 | 0.202 |
| UR-FUNNY | Visual | 49.2 ± 0.78 | 49.8 ± 0.66 |
| | Audio | 59.6 ± 0.64 | 55.8 ± 3.26 |
| | Text | 69.0 ± 0.05 | 68.5 ± 0.58 |
| | IBF | 0 | 0.024 |

Table A.2: Unimodal Accuracies and IBF scores for SOTA methods OGM, PMR, and MSLR (test set) (concat fusion). Mean ± standard error over 3 random seeds. IBF is calculated based on the mean values.

| Dataset | | Single | OGM | PMR | MSLR |
|---|---|---|---|---|---|
| CREMA-D | Visual | 60.5 ± 0.75 | 24.7 ± 1.07 | 18.5 ± 0.59 | 41.2 ± 1.14 |
| | Audio | 59.1 ± 0.49 | 53.6 ± 0.90 | 50.5 ± 1.83 | 48.8 ± 0.95 |
| | IBF | 0 | 0.342 | 0.420 | 0.247 |
| AVE | Visual | 28.5 ± 0.64 | 15.5 ± 0.67 | 11.1 ± 2.07 | 22.0 ± 0.98 |
| | Audio | 55.3 ± 0.56 | 50.8 ± 1.00 | 49.4 ± 1.79 | 50.4 ± 2.00 |
| | IBF | 0 | 0.269 | 0.359 | 0.158 |
| MVSA | Visual | 57.7 ± 1.21 | 56.5 ± 1.49 | 56.7 ± 0.58 | 54.1 ± 0.51 |
| | Text | 70.8 ± 0.10 | 68.8 ± 0.73 | 66.0 ± 1.33 | 68.2 ± 0.63 |
| | IBF | 0 | 0.025 | 0.043 | 0.050 |
| IEMOCAP | Visual | 46.5 ± 0.53 | 26.7 ± 0.81 | – | 31.9 ± 1.62 |
| | Audio | 49.2 ± 1.41 | 41.6 ± 1.23 | – | 48.3 ± 0.41 |
| | Text | 62.3 ± 0.30 | 56.7 ± 1.39 | – | 49.6 ± 0.22 |
| | IBF | 0 | 0.223 | – | 0.179 |
| UR-FUNNY | Visual | 49.2 ± 0.78 | 50.3 ± 0.41 | – | 50.0 ± 0.12 |
| | Audio | 59.6 ± 0.64 | 55.8 ± 2.23 | – | 60.0 ± 0.62 |
| | Text | 69.0 ± 0.05 | 68.3 ± 0.64 | – | 65.7 ± 1.84 |
| | IBF | 0 | 0.025 | – | 0.016 |

| Dataset | | Single | Alternating | AMST | Entropy | Reliability |
|---|---|---|---|---|---|---|
| CREMA-D | Visual | 60.5 ± 0.75 | 57.0 ± 1.52 | 59.7 ± 0.61 | 61.8 ± 1.60 | 61.7 ± 0.38 |
| | Audio | 59.1 ± 0.49 | 59.7 ± 0.24 | 61.7 ± 0.12 | 62.6 ± 1.44 | 61.7 ± 0.31 |
| | IBF | 0 | 0.029 | 0.007 | 0.000 | 0.000 |
| AVE | Visual | 28.5 ± 0.64 | 31.0 ± 0.41 | 30.1 ± 0.85 | 31.0 ± 0.98 | 31.1 ± 0.79 |
| | Audio | 55.3 ± 0.56 | 56.2 ± 0.99 | 57.2 ± | 50.8 ± 1.83 | 54.5 ± 0.39 |
| | IBF | 0 | 0.000 | 0.000 | 0.041 | 0.007 |
| MVSA | Visual | 57.7 ± 1.21 | 57.8 ± 0.87 | 59.1 ± 1.15 | 56.9 ± 0.97 | 57.9 ± 0.24 |
| | Text | 70.8 ± 0.10 | 70.3 ± 0.31 | 70.2 ± 0.29 | 68.1 ± 1.97 | 69.9 ± 1.48 |
| | IBF | 0 | 0.004 | 0.004 | 0.026 | 0.006 |
| IEMOCAP | Visual | 46.5 ± 0.53 | 33.6 ± 4.24 | 40.2 ± 0.99 | 45.7 ± 1.09 | 43.0 ± 0.39 |
| | Audio | 49.2 ± 1.41 | 45.9 ± 3.17 | 48.3 ± 0.75 | 46.2 ± 1.04 | 50.0 ± 0.50 |
| | Text | 62.3 ± 0.30 | 60.9 ± 0.38 | 57.2 ± 0.39 | 52.3 ± 0.81 | 61.1 ± 0.55 |
| | IBF | 0 | 0.122 | 0.079 | 0.080 | 0.032 |
| UR-FUNNY | Visual | 49.2 ± 0.78 | 50.6 ± 0.78 | 49.5 ± 0.67 | 49.5 ± 0.35 | 50.4 ± 1.23 |
| | Audio | 59.6 ± 0.64 | 56.3 ± 1.31 | 58.5 ± 0.49 | 57.3 ± 2.15 | 57.0 ± 0.59 |
| | Text | 69.0 ± 0.05 | 69.0 ± 0.44 | 69.4 ± 0.18 | 67.9 ± 0.95 | 70.0 ± 0.49 |
| | IBF | 0 | 0.018 | 0.006 | 0.018 | 0.015 |

Table A.3: Unimodal Accuracies and IBF scores for Alternating, Entropy, and Reliability training (test set). Mean ± standard error over 3 random seeds. IBF is calculated based on the mean values.

#### A.1.0.1 Fusion results of different training strategies

Table A.4: Test set Accuracy of Concat, Sum, Gated, and FiLM fusion under Naive Training. Gated and FiLM only support 2 modalities. Mean ± standard error over 3 random seeds.

| | **Naive** | | | |
|---|---|---|---|---|
| Dataset | Concat | Sum | Gated | FiLM |
| CREMAD | 64.2 ± 0.91 | 65.5 ± 0.27 | 64.2 ± 1.37 | 62.1 ± 0.48 |
| AVE | 58.8 ± 0.49 | 60.7 ± 0.33 | 52.5 ± 0.87 | 59.7 ± 0.74 |
| MVSA | 71.0 ± 0.85 | 71.6 ± 0.57 | 71.9 ± 0.43 | 72.4 ± 0.43 |
| IEMOCAP2 | 49.2 ± 0.86 | 50.7 ± 0.18 | 47.9 ± 0.77 | 50.2 ± 0.56 |
| IEMOCAP3 | 69.3 ± 0.27 | 68.2 ± 0.27 | – | – |
| UR-FUNNY | 71.6 ± 0.13 | 70.2 ± 0.23 | – | – |

Table A.5: Test set Accuracy of Concat, Sum, Gated, and FiLM fusion under Alternating Training. Gated and FiLM only support 2 modalities. Mean ± standard error over 3 random seeds.

| | **Alternating** | | | |
|---|---|---|---|---|
| Dataset | Concat | Sum | Gated | FiLM |
| CREMAD | 74.6 ± 0.70 | 72.9 ± 0.47 | 70.1 ± 0.77 | 75.9 ± 0.29 |
| AVE | 64.4 ± 1.12 | 62.5 ± 1.07 | 62.3 ± 0.36 | 60.2 ± 0.29 |
| MVSA | 73.0 ± 0.84 | 72.6 ± 0.55 | 71.2 ± 0.46 | 71.0 ± 0.07 |
| IEMOCAP2 | 55.5 ± 0.36 | 53.5 ± 0.70 | 54.8 ± 1.70 | 54.4 ± 1.26 |
| IEMOCAP3 | 66.4 ± 0.54 | 66.9 ± 1.05 | – | – |
| UR-FUNNY | 69.1 ± 1.02 | 70.5 ± 0.72 | – | – |

Table A.6: Test set Accuracy of Concat, Sum, Gated, and FiLM fusion under AMST. Gated and FiLM only support 2 modalities. Mean ± standard error over 3 random seeds.

| | **AMST** | | | |
|---|---|---|---|---|
| Dataset | Concat | Sum | Gated | FiLM |
| CREMAD | 77.7 ± 1.11 | 77.1 ± 0.17 | 76.2 ± 0.29 | 77.7 ± 0.40 |
| AVE | 64.2 ± 0.62 | 62.7 ± 0.37 | 64.9 ± 1.00 | 59.5 ± 1.37 |
| MVSA | 71.4 ± 0.37 | 72.2 ± 0.84 | 73.4 ± 0.21 | 71.2 ± 0.85 |
| IEMOCAP2 | 60.2 ± 1.19 | 57.7 ± 1.12 | 53.7 ± 0.78 | 51.6 ± 0.99 |
| IEMOCAP3 | 65.5 ± 0.92 | 67.2 ± 0.54 | – | – |
| UR-FUNNY | 71.2 ± 0.23 | 71.0 ± 0.29 | – | – |

Table A.7: Accuracy of Concatenation, Summation, Gated, and FiLM fusion under Entropy Training. Gated and FiLM only support 2 modalities. Mean ± standard error over 3 random seeds.

| | **Entropy** | | | |
|---|---|---|---|---|
| Dataset | Concat | Sum | Gated | FiLM |
| CREMAD | 79.2 ± 0.76 | 78.0 ± 0.61 | 77.3 ± 0.38 | 77.6 ± 1.83 |
| AVE | 60.2 ± 0.30 | 59.1 ± 0.64 | 58.5 ± 1.00 | 52.6 ± 0.60 |
| MVSA | 70.3 ± 0.31 | 72.7 ± 0.44 | 72.0 ± 0.55 | 71.8 ± 0.55 |
| IEMOCAP2 | 58.7 ± 1.35 | 57.9 ± 0.81 | 57.0 ± 1.65 | 54.5 ± 0.55 |
| IEMOCAP3 | 64.7 ± 0.83 | 64.6 ± 0.70 | – | – |
| UR-FUNNY | 68.9 ± 0.43 | 68.4 ± 0.88 | – | – |

Table A.8: Accuracy of Concatenation, Summation, Gated, and FiLM fusion under Reliability Training. Gated and FiLM only support 2 modalities. Mean ± standard error over 3 random seeds.

| | **Reliability** | | | |
|---|---|---|---|---|
| Dataset | Concat | Sum | Gated | FiLM |
| CREMAD | 78.5 ± 0.23 | 78.1 ± 0.98 | 74.7 ± 1.60 | 77.4 ± 1.14 |
| AVE | 63.8 ± 0.30 | 64.1 ± 1.07 | 62.9 ± 0.50 | 59.0 ± 1.50 |
| MVSA | 73.0 ± 0.12 | 73.2 ± 0.12 | 70.1 ± 0.20 | 71.4 ± 0.10 |
| IEMOCAP2 | 60.2 ± 1.02 | 59.3 ± 0.28 | 58.6 ± 0.85 | 57.1 ± 56.4 |
| IEMOCAP3 | 71.2 ± 0.40 | 71.3 ± 0.52 | – | – |
| UR-FUNNY | 72.3 ± 0.43 | 71.1 ± 0.35 | – | – |

#### A.1.0.2 Comparison of Alternating, AMST, Entropy and Reliability Training with SOTA methods

Table A.9: Test set accuracy of Alternating, AMST, Entropy and Reliability Training compared to SOTA methods on CREMA-D, AVE, MVSA, and IEMOCAP2. We use concat fusion for all methods and MLA's entropy-based fusion for MLA. For MART, the results were obtained from Reliability Training. The best results are in bold and the second best are underlined. Mean ± standard deviation over 3 random seeds.

| Method | CREMA-D | AVE | MVSA | IEMOCAP2 |
|---|---|---|---|---|
| Naive | 64.2 ± 0.92 | 58.8 ± 0.49 | 71.0 ± 0.85 | 49.2 ± 0.86 |
| OGM | 63.7 ± 1.41 | 59.0 ± 1.19 | 70.9 ± 1.00 | 51.6 ± 0.24 |
| PMR | 67.5 ± 1.65 | 58.8 ± 1.56 | 72.3 ± 0.90 | 51.9 ± 0.24 |
| MSLR | 76.5 ± 0.33 | 63.8 ± 1.66 | 71.5 ± 0.90 | 51.8 ± 1.19 |
| MLA | 73.7 ± 0.30 | 63.6 ± 0.44 | 71.4 ± 0.35 | 53.9 ± 0.81 |
| Alternating | 74.6 ± 0.70 | 64.4 ± 1.12 | 73.0 ± 0.84 | 55.5 ± 0.36 |
| AMST | 77.7 ± 1.11 | 64.2 ± 0.62 | 71.4 ± 0.37 | 60.2 ± 1.19 |
| Entropy | 79.2 ± 0.76 | 60.2 ± 0.30 | 70.3 ± 0.31 | 58.7 ± 1.35 |
| Reliability | 78.5 ± 0.23 | 63.8 ± 0.30 | 73.0 ± 0.12 | 60.2 ± 1.02 |

Table A.10: Test set accuracy of Alternating, AMST, Entropy and Reliability Training compared to SOTA methods on IEMOCAP3 and UR-FUNNY. Same settings as Table A.9.

| Method | IEMOCAP3 | UR-FUNNY |
|---|---|---|
| Naive | 69.3 ± 0.27 | 71.6 ± 0.13 |
| OGM | 67.1 ± 0.93 | 70.3 ± 0.23 |
| PMR | – | – |
| MSLR | 61.4 ± 0.55 | 69.2 ± 0.90 |
| MLA | 65.7 ± 0.62 | 70.3 ± 0.24 |
| Alternating | 66.4 ± 0.54 | 69.1 ± 1.02 |
| AMST | 65.5 ± 0.92 | 71.2 ± 0.23 |
| Entropy | 64.7 ± 0.83 | 68.9 ± 0.43 |
| Reliability | 71.2 ± 0.40 | 72.3 ± 0.43 |

#### A.1.0.3 Modality train bias metrics and number of trained epochs

Table A.11: Train bias metrics & number of trained epochs for CREMA-D and AVE datasets. Mean ± standard deviation over 3 random seeds. Concatenation Fusion.

| | | CREMA-D | | AVE | |
|---|---|---|---|---|---|
| | | Audio | Visual | Audio | Visual |
| Relative Bias | Alternating | 0.68 ± 0.01 | 0.32 ± 0.01 | 0.66 ± 0.01 | 0.34 ± 0.01 |
| | AMST | 0.51 ± 0.00 | 0.49 ± 0.00 | 0.63 ± 0.00 | 0.37 ± 0.00 |
| | Entropy | 0.51 ± 0.00 | 0.49 ± 0.00 | 0.51 ± 0.00 | 0.49 ± 0.00 |
| | Reliability | 0.57 ± 0.01 | 0.43 ± 0.01 | 0.55 ± 0.01 | 0.45 ± 0.01 |
| Trained Epochs | Alternating | 100 ± 0.00 | 100 ± 0.00 | 100 ± 0.00 | 100 ± 0.00 |
| | AMST | 20 ± 0.00 | 100 ± 0.00 | 50 ± 0.00 | 100 ± 0.00 |
| | Entropy | 15 ± 0.88 | 100 ± 0.00 | 17 ± 1.15 | 100 ± 0.00 |
| | Reliability | 18 ± 0.33 | 100 ± 0.00 | 22 ± 1.76 | 100 ± 0.00 |

Table A.12: Train bias metrics & number of trained epochs for IEMOCAP2 and IEMOCAP3 datasets. Mean ± standard error over 3 random seeds. Concatenation Fusion.

| | | IEMOCAP2 | | IEMOCAP3 | | |
|---|---|---|---|---|---|---|
| | | Audio | Visual | Audio | Visual | Text |
| Relative Bias | Alternating | 0.76 ± 0.01 | 0.24 ± 0.01 | 0.37 ± 0.04 | 0.12 ± 0.01 | 0.50 ± 0.05 |
| | AMST | 0.66 ± 0.01 | 0.34 ± 0.01 | 0.38 ± 0.00 | 0.21 ± 0.01 | 0.41 ± 0.00 |
| | Entropy | 0.50 ± 0.00 | 0.50 ± 0.00 | 0.33 ± 0.01 | 0.32 ± 0.01 | 0.34 ± 0.01 |
| | Reliability | 0.59 ± 0.02 | 0.41 ± 0.02 | 0.35 ± 0.01 | 0.22 ± 0.01 | 0.43 ± 0.01 |
| Trained Epochs | Alternating | 100 ± 0.00 | 100 ± 0.00 | 100 ± 0.00 | 100 ± 0.00 | 100 ± 0.00 |
| | AMST | 26 ± 0.00 | 100 ± 0.00 | 26 ± 0.00 | 100 ± 0.00 | 10 ± 0.00 |
| | Entropy | 14 ± 1.76 | 100 ± 0.00 | 18 ± 4.91 | 100 ± 0.00 | 5 ± 0.67 |
| | Reliability | 15 ± 0.67 | 100 ± 0.00 | 12 ± 0.33 | 100 ± 0.00 | 14 ± 0.58 |

Table A.13: Train bias metrics & number of trained epochs for MVSA and UR-FUNNY datasets. Mean ± standard deviation over 3 random seeds. Concatenation Fusion.

| | | MVSA | | UR-FUNNY | | |
|---|---|---|---|---|---|---|
| | | Text | Visual | Audio | Visual | Text |
| Relative Bias | Alternating | 0.75 ± 0.02 | 0.25 ± 0.02 | 0.23 ± 0.06 | 0.02 ± 0.01 | 0.76 ± 0.06 |
| | AMST | 0.57 ± 0.03 | 0.43 ± 0.03 | 0.42 ± 0.03 | 0.04 ± 0.00 | 0.54 ± 0.03 |
| | Entropy | 0.58 ± 0.02 | 0.42 ± 0.02 | 0.34 ± 0.09 | 0.09 ± 0.01 | 0.57 ± 0.07 |
| | Reliability | 0.62 ± 0.01 | 0.38 ± 0.01 | 0.32 ± 0.03 | 0.04 ± 0.01 | 0.64 ± 0.03 |
| Trained Epochs | Alternating | 100 ± 0.00 | 100 ± 0.00 | 100 ± 0.00 | 100 ± 0.00 | 100 ± 0.00 |
| | AMST | 10 ± 0.00 | 100 ± 0.00 | 18 ± 0.00 | 100 ± 0.00 | 10 ± 0.00 |
| | Entropy | 4 ± 0.00 | 100 ± 0.00 | 9 ± 2.60 | 100 ± 0.00 | 2 ± 0.00 |
| | Reliability | 11 ± 0.33 | 100 ± 0.00 | 7 ± 0.58 | 100 ± 0.00 | 8 ± 0.33 |

## A.2 Dataloader

### A.2.1 Prefetching Estimation

Due to the limitation of Pytorch, we can not set 0 to prefetch parameters, thus, we set it to 1 and iterate the dataset with for, but without for-inner workload during the iteration, as code shown below:

**Algorithm 7** Idel time of Prefetch factor 0 estimation

1: **Input:** Data $\{X, Y\}$, Batches $B$
2: **for** b = 1 to $B$ **do**
3: **CPU:** Load $\{x_b, y_b\}$
4: **end for**

**Algorithm 8** Idel time of Prefetch factor $\geq 1$

1: **Input:** Data $\{X, Y\}$, Batches $B$
2: **for** b = 1 to $B$ **do**
3: **CPU:** Load $\{x_b, y_b\}$
4: **GPU:** Simulated Workload.
5: **end for**

## A.3 Improvement from Optimized loader on Different Datasets

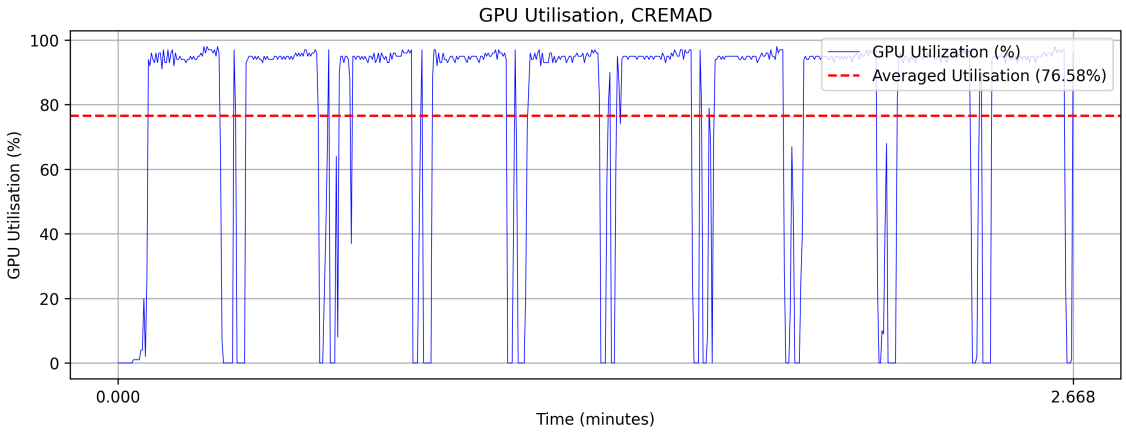



Figure A.1: GPU Utilisation on CREMA-D (Without Optimisation)

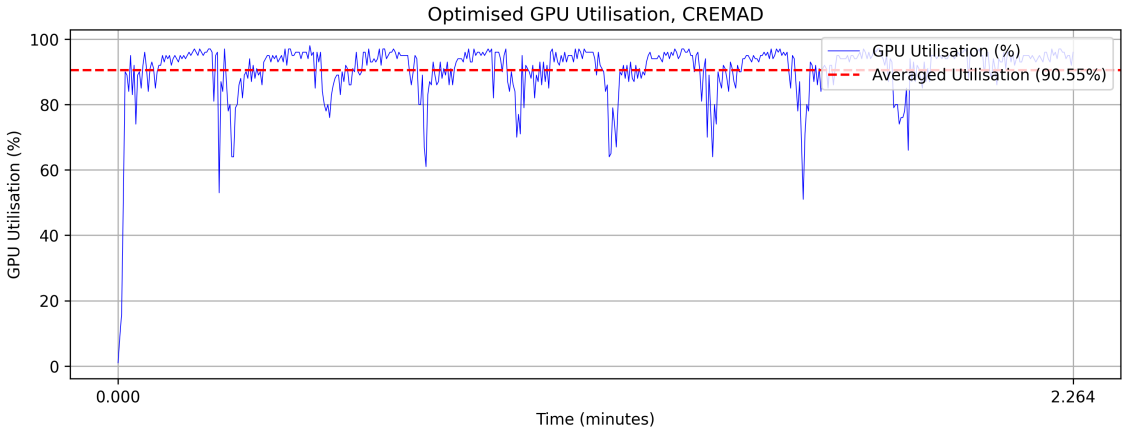



Figure A.2: GPU Utilisation on CREMA-D (With Optimisation)

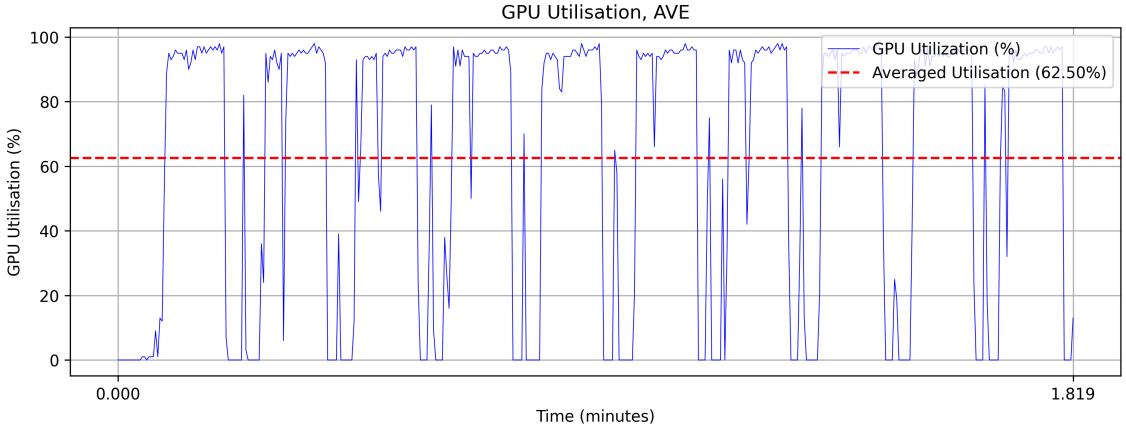



Figure A.3: GPU Utilisation on AVE (Without Optimisation)

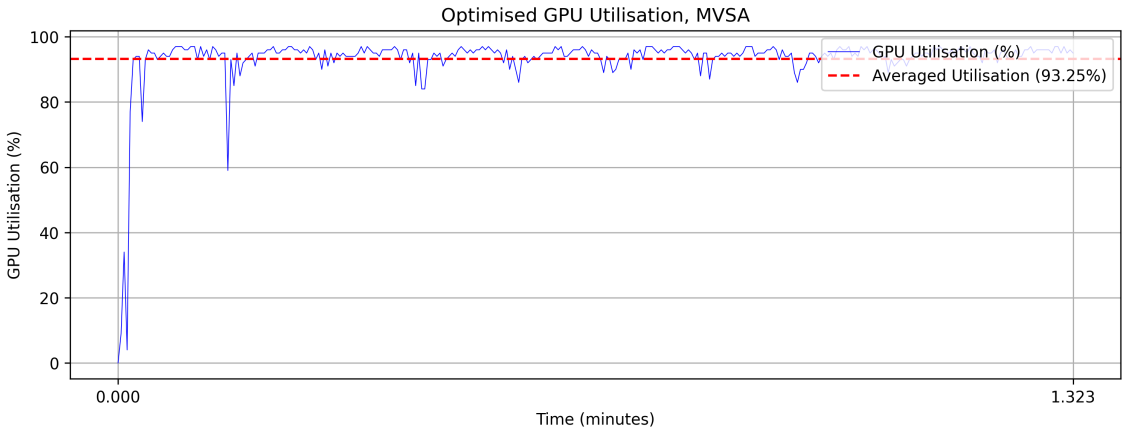



Figure A.4: GPU Utilisation on MVSA (With Optimisation)

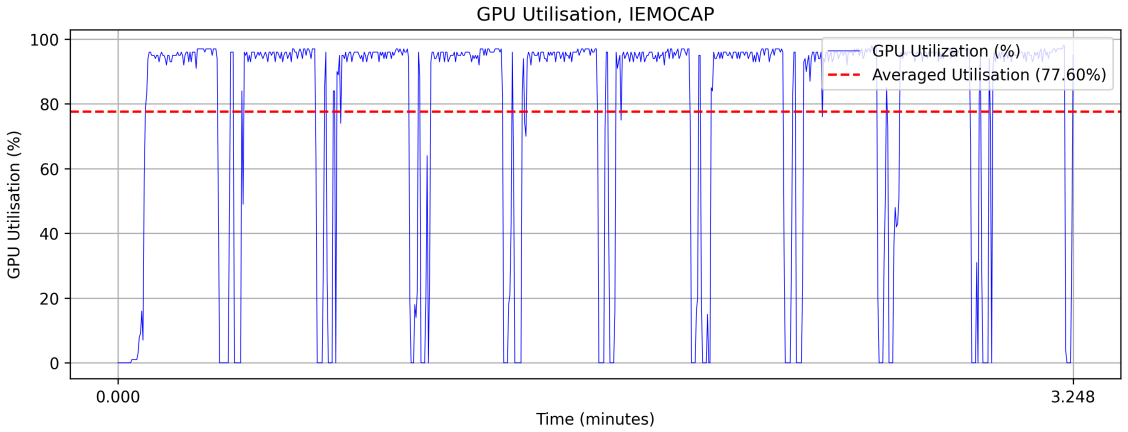



Figure A.5: GPU Utilisation on IEMOCAP (Without Optimisation)

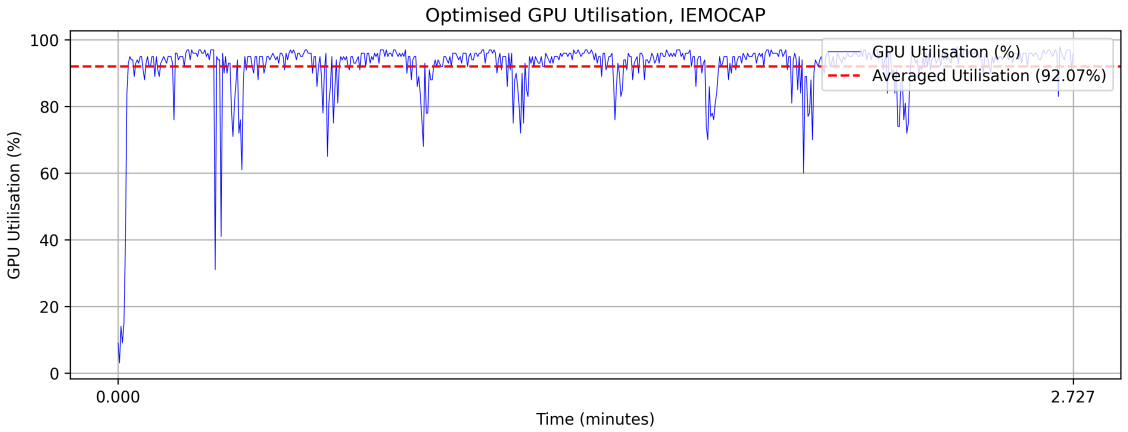



Figure A.6: GPU Utilisation on IEMOCAP (With Optimisation)

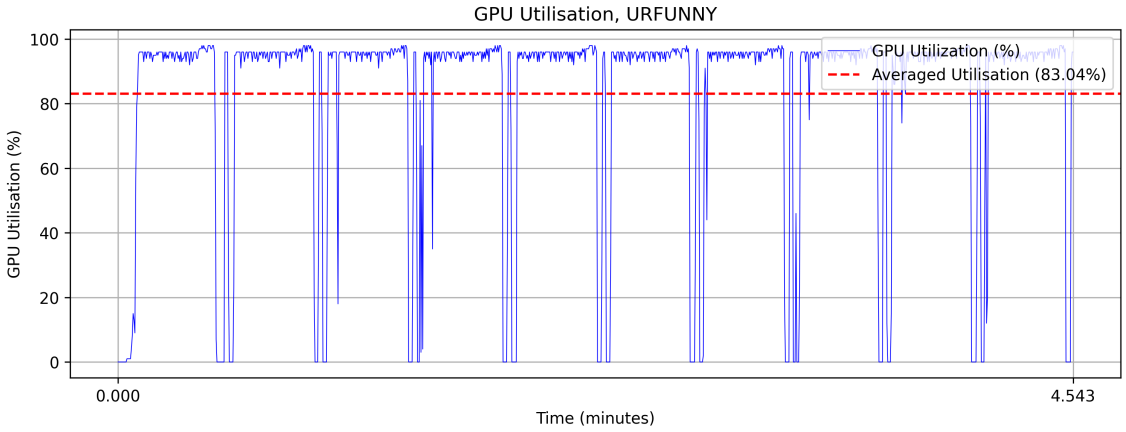



Figure A.7: GPU Utilisation on UR-FUNNY (Without Optimisation)

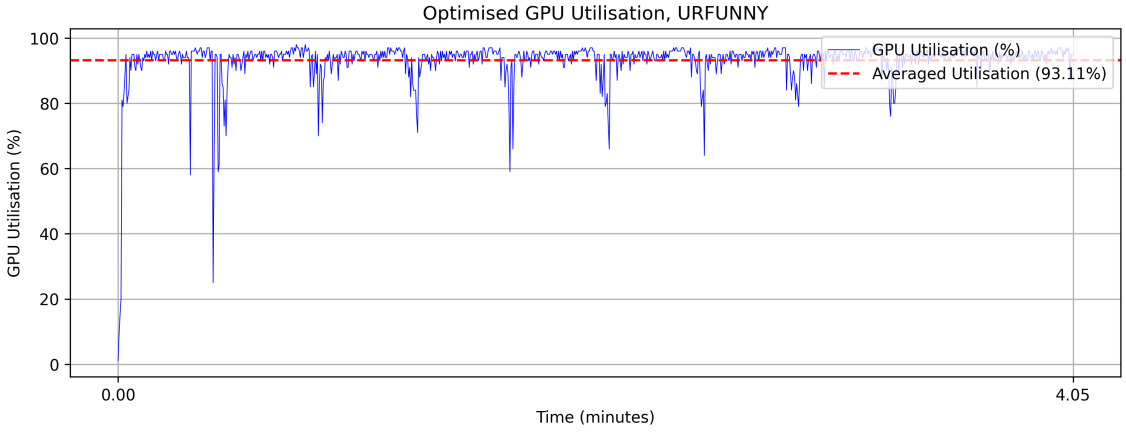



Figure A.8: GPU Utilisation on UR-FUNNY (With Optimisation)