Adaptive Inference for Cost-Efficient Deep Neural Networks

Thèse de doctorat – Université Gustave Eiffel

École Doctorale Mathématiques et STIC (MSTIC – ED 532)
Discipline : Mathématiques
Laboratoire d'Analyse et Mathématiques Appliquées (LAMA – UMR 8050)

Thèse présentée et soutenue à l'Université Gustave Eiffel, le 27 Janvier 2026, par :

Florian VALADE

Composition du Jury

Claire LACOUR Professeure, Université Gustave Eiffel	Présidente du jury
Yohann De CASTRO Professor, Institut Camille Jordan	Rapporteur
Massimiliano PONTIL Professor, Istituto italiano di tecnologica	Rapporteur
Katia MEZIANI Maîtresse de conférences, Université Paris Dauphine	Examinatrice
Romuald ELIE Professeur des universités, Google Deepmind	Examinateur
Paul GAY Assistant professor, CYTECH	Examinateur

Encadrement de la thèse

Mohamed HEBIRI Maître de conférences, Université Gustave Eiffel	Directeur de thèse

Acknowledgments

I would like to express my deepest gratitude to all the people who have contributed, directly or indirectly, to the completion of this thesis.

First and foremost, my sincere thanks go to Mohamed for his invaluable guidance, support, and encouragement throughout these years. I have learned tremendously from you, not only scientifically but also on a personal and professional level.

I would like to thank Yohann De Castro and Massimiliano Pontil for reviewing this work. My gratitude also goes to Claire Lacour, Katia Meziani, Romuald Elie, and Paul Gay for honoring me by being part of my jury.

I am grateful to Fujitsu for funding this PhD and for giving me the opportunity to work on exciting and meaningful projects. My special thanks go to Laurent and Axel, without whom this work would not have been possible. I am also thankful to Misu, Margarita, Adrien, Khan, Sebastien, Moïse, Thierry for the fruitful technical and scientific discussions, and to Thomas, Anaïs, Sofiane, Emma, Maud, Mariana, Bang-tam, Aubane, Edward, for all the great moments we shared. The entire Fujitsu team created a truly collaborative and inspiring environment—being part of such a supportive and curious company was a constant source of motivation.

During these years, I had the privilege to collaborate with outstanding researchers from whom I have learned a great deal. My gratitude goes to the IIT team: Massi, for welcoming me so warmly in Genova; Vladi, for your contagious enthusiasm for research; and Alek, Erfan, and Matia for the insightful discussions and the fun times.

I would also like to thank Université Gustave Eiffel and the LAMA laboratory for hosting me during these three years. Thank you Audrey for your help in navigating the administrative side of things, and to Gayane for bringing joy, drawings, and origami to our small two-person office, you made every day brighter. To Romain and Eddy, you made my first international conference unforgettable. To all the colleagues I met at LAMA, Felix, Claire, Nicolas, Christophe, from whom I have learned a lot. And to all the PhD students, thank you for your kindness and stimulating conversations.

My gratitude also extends to UPPA, and particularly to Sébastien and Paul, for welcoming me to Pau and for the fruitful scientific discussions we shared.

To all the passionate researchers I have met along the way—Katia, Karim, Evgeni, Daphney, and many others—thank you for inspiring me to keep pushing forward with your dedication and love for science.

Outside of the academic world, I owe so much to the people who personally supported me throughout this journey. To my family, thank you for your unconditional love and support. To Anaïs, thank you for believing in me from the very beginning—this thesis would never have started without you, and I wouldn't be the person I am today.

To my friends, from childhood to these PhD years: thank you for being there, for the music, the adventures, and all those moments that made life so much brighter. A huge thank you to Lea, Michael, Maxime, and Stefan for a friendship that has lasted since the very beginning.

A special and heartfelt thank you to Ludo, Paul, Clem, and Nico. You are incredible people and you made these years truly memorable. You might not realize how much you've supported me, or how much your presence meant during this journey, but it was more than I can put into words.

Finally, to all those whose names may not appear here but whose support and kindness have meant so much—thank you. Your presence, even in small ways, has been invaluable throughout this journey.

Abstract

The continuous rise of deep neural networks comes with an explosion in computational cost, latency, and energy consumption during inference. These constraints become critical in real-time computer vision scenarios, where multiple video streams must be processed in parallel on resource-limited *edge* platforms, as well as in the execution of large-scale models hosted in data centers.

This thesis proposes a systematic approach based on *early exits* to dynamically adapt the effective depth of a network according to the difficulty of each instance. The thesis explores three aspects:

1. **Architecture**: optimal design, placement, and sizing of exit branches that preserve feature hierarchy while minimizing parameter overhead;
2. **Training**: hierarchical loss strategies, multi-level distillation, and statistical calibration of outputs aimed at balancing overall accuracy and reliability of intermediate estimators;
3. **Inference**: policies based on statistically grounded confidence thresholds to decide on early stopping, and latency-energy measurement protocol on server GPU and Jetson platforms.

Major contributions include:

i) a systematic exploration of existing methods across the entire early exit deployment chain (architecture, training, and inference);

ii) a rejection-based inference approach enabling efficient use of early exits in resource-constrained environments;

iii) a specific training and inference methodology adapted to large language models (LLMs) allowing the exploitation of early exits to significantly reduce computational costs while preserving the quality of generated responses.

All code, profiling scripts, and trained models are published as open-source to promote research on energy efficiency in deep learning.

The results confirm that *early exit* networks constitute a pragmatic solution to reconcile performance and digital sobriety, paving the way for secure and privacy-respectful deployments on the *edge* and beyond.

Résumé

L'essor continu des réseaux de neurones profonds s'accompagne d'une explosion du coût computationnel, de la latence et de la consommation énergétique lors de l'inférence. Ces contraintes deviennent critiques dans les scénarios de vision par ordinateur temps réel, où plusieurs flux vidéo doivent être traités en parallèle sur des plateformes *edge* à ressources limitées, ainsi que dans l'exécution de modèles à grande échelle hébergés dans des centres de données.

Cette thèse propose une approche systématique fondée sur les *sorties anticipées* (**early exit**) pour adapter dynamiquement la profondeur effective d'un réseau en fonction de la difficulté de chaque instance. Le manuscrit explore trois aspects:

1. **Architecture** : conception, placement et dimensionnement optimaux des branches de sortie permettant de préserver la hiérarchie des caractéristiques tout en minimisant le surcoût paramétrique;

2. **Entraînement** : stratégies de pertes hiérarchiques, distillation multi-niveaux et calibration statistique des sorties visant à équilibrer précision globale et fiabilité des estimateurs intermédiaires;

3. **Inférence** : politiques basées sur des seuils de confiance statistiquement fondés pour décider de l'arrêt précoce, et protocole de mesure latence-énergie sur platesformes GPU serveur et Jetson.

Les contributions majeures incluent :

i) une exploration systématique des méthodes existantes à travers toute la chaîne de déploiement de l'early exit (architecture, entraînement et inférence);

ii) une approche d'inférence basée sur le rejet permettant l'utilisation efficace des sorties anticipées dans des environnements à ressources contraintes;

iii) une méthodologie spécifique d'entraînement et d'inférence adaptée aux modèles de langage de grande taille (LLMs) permettant d'exploiter les sorties anticipées pour réduire significativement les coûts computationnels tout en préservant la qualité des réponses générées.

L'ensemble du code, des scripts de profilage et des modèles entraînés est publié en open-source afin de favoriser la recherche sur l'efficacité énergétique du deep learning.

Les résultats obtenus confirment que les réseaux à *early exit* constituent une solution pragmatique pour concilier performances et sobriété numérique, ouvrant la voie à des déploiements sécurisés et respectueux de la vie privée sur l'*edge* et au-delà.

Contents

1 Introduction **9**

Introduction **9**
1.1 General Context 9
1.1.1 Problem Statement: Computational Cost, Latency, and Energy . . 10
1.1.2 Early Exit Methods: A Promising Direction 12
1.1.3 Roadmap 12
1.2 Architecture of Early Exit Networks 12
1.2.1 Mathematical Foundations 13
1.2.2 Historical Evolution of Multi-Exit Architectures 17
1.3 Training Methodologies for Early Exit Networks 23
1.3.1 Mathematical Formulation 23
1.3.2 Standard Multi-Exit Training 23
1.3.3 Limitations of Standard Training 24
1.3.4 Early Exit as a Pluggable Technique 24
1.4 Inference Strategies and Dynamic Execution 25
1.4.1 Dynamic Inference Mechanics 27
1.4.2 Exit Strategy Objectives and Approaches 28
1.4.3 Deployment Considerations 31
1.5 Contributions and Organization 33
1.5.1 Statistical Perspective on Early-Exit Learning 33
1.5.2 Statistical Perspective on Early-Exit Inference 33
1.5.3 Contribution 1: Budget-Constrained Inference with EERO (Chapter 3) 33
1.5.4 Contribution 2: Accuracy-Preserving Early Exits for LLMs (Chapter 4) 33
1.5.5 Broader Efficiency Ecosystem and System Insights (Chapter 2) . . . 33
1.5.6 Summary of Contributions 34

2 Literature Review **35**
2.1 Early Exit Architectures 35
2.1.1 Computer Vision Architectures 35
2.1.2 Natural Language Processing Architectures 38
2.2 Training Methodologies for Early Exit Networks 39
2.2.1 Multi-Exit Loss Functions: Balancing Competing Objectives 39
2.2.2 Training Strategies: Joint versus Progressive Optimization Paradigms 41
2.2.3 Knowledge Distillation: Leveraging Hierarchical Supervision 41
2.2.4 Gradient Flow Issues: Managing Optimization Conflicts 42
2.2.5 Self-Supervised Approaches: Eliminating Task-Specific Requirements 42
2.2.6 Calibration During Training: Ensuring Reliable Confidence Estimates 43
2.2.7 Critical Analysis: Training Paradigm Effectiveness 43
2.3 Inference Mechanisms and Adaptive Decision Policies 43
2.3.1 Confidence Estimation and Uncertainty Quantification 44

2.3.2 Decision Policy Frameworks . . . 45
2.3.3 Theoretical Foundations and Optimality . . . 45
2.3.4 Rigorous Guarantee Frameworks . . . 47
2.3.5 Policy Evaluation and Comparison . . . 47
2.4 Efficiency Methods, System Considerations, and Evaluation Frameworks . 48
2.4.1 Comparative Analysis with Other Efficiency Methods . . . 48
2.4.2 Domain-Specific Architectural Adaptations . . . 49
2.4.3 System-Level Deployment Considerations . . . 50
2.4.4 Evaluation Methodologies and Benchmarking . . . 51
2.5 Towards Practical Adaptive Inference . . . 52

3 Early Exit with Reject Option (EERO) 54
3.1 Introduction . . . 54
3.2 Related work . . . 55
3.3 Method . . . 56
3.3.1 Statistical framework – classification with reject option . . . 57
3.3.2 Data-driven procedure . . . 58
3.4 Experiments . . . 62
3.4.1 EERO with multiple heads . . . 62
3.4.2 Comparison with Existing Early Exit Strategies . . . 63
3.5 Conclusion . . . 64
3.6 Implementation details of experiments . . . 65
3.7 Analysis of aggregation weights . . . 65
3.8 EERO with only 1 auxiliary head . . . 66
3.9 Proofs . . . 67

4 Accelerating LLMs with Early Exits 76
4.1 Introduction . . . 76
4.2 Related Work . . . 77
4.3 Methodology . . . 78
4.3.1 Definitions and Notation . . . 78
4.3.2 Implementation Details . . . 79
4.3.3 Training data and objectives . . . 79
4.3.4 Calibration and Inference . . . 80
4.4 Experiments and Results . . . 82
4.4.1 Training . . . 83
4.4.2 Inference . . . 83
4.5 Limitations and potential impacts . . . 84
4.6 Conclusion . . . 85
4.7 Appendix . . . 87
4.7.1 Full benchmark results . . . 87
4.7.2 Post-training analysis . . . 87
4.7.3 Entropy distribution analysis . . . 87
4.7.4 Per-token accuracy analysis . . . 91

Conclusion and Perspectives 93

Bibliography 96

Chapter 1

Introduction

1.1 General Context

Deep Learning has revolutionized the field of artificial intelligence over the past few decades. This evolution is not the result of a sudden innovation, but rather the outcome of a gradual maturation of mathematical concepts, algorithms, and technological infrastructures.

Historical Foundations The theoretical foundations of deep learning date back to the 1940s with the first formal neuron models proposed by [92] and [106]. However, it was not until the 1980s that the first practical implementations emerged with the backpropagation algorithm [107], enabling efficient training of multilayer perceptrons (MLPs). These early models were already capable of solving classification problems on relatively simple tabular data but faced significant limitations when dealing with more complex data.

A major breakthrough occurred with the work of LeCun et al. [73] on convolutional neural networks (CNNs), which demonstrated their effectiveness in handwritten digit recognition with the famous LeNet-5 network and the MNIST database. This breakthrough marked a decisive turning point, as it established an architectural model adapted to computer vision, capable of automatically extracting hierarchical features from images.

The Evolution of Model Intelligence

The success of deep architectures can be attributed to their ability to learn hierarchical feature representations. This key property allows early layers to extract simple and global features (such as edges, textures, or basic shapes in the case of image processing), while subsequent layers progressively build upon these representations to capture increasingly complex and abstract patterns. This hierarchical organization mirrors the structure observed in biological visual systems [58] and provides these networks with a remarkable capacity for generalization. Visualization techniques like those developed in [153] have confirmed this intuition by revealing how features evolve from general to specific as information propagates through the network.

Despite these conceptual advances, deep learning remained relatively marginal until the early 2010s. Three converging factors then enabled its spectacular rise: the availability of vast annotated datasets like ImageNet [24], the exponential increase in computing power with GPUs [102], and several algorithmic innovations such as the use of new activation functions (ReLU) [96] and regularization techniques (Dropout) [121].

Diversification of Applications

The year 2012 marks a pivotal moment with the resounding victory of the AlexNet model [68] in the ImageNet competition, drastically reducing the error rate in image classification

compared to traditional methods. As models grew more powerful, numerous applications emerged across diverse domains. Computer vision projects began exploring real-time video processing for autonomous driving, surveillance systems, and augmented reality. In natural language processing, models enabled machine translation, sentiment analysis, and question answering at unprecedented levels of accuracy.

This diversification of use cases pushed existing infrastructures to their limits and created a demand for even more capable models that could handle increasingly complex tasks. Each application domain required specialized capabilities, driving research toward more sophisticated architectures tailored to specific problems.

The Scaling Paradigm

The empirical success of early deep models validated a fundamental paradigm: increasing model size, combined with more data and computing power, leads to enhanced performance. This "scaling" paradigm was further confirmed with the emergence of increasingly deeper networks such as VGG [120] (19 layers), followed by architectures with residual connections like ResNet [50], enabling hundreds of layers. Performance in visual recognition has continued to improve, reaching and sometimes exceeding human capabilities on specific tasks [49].

The last decade has seen a spectacular acceleration of this trend toward increasing model complexity. In natural language processing, the introduction of architectures based on attention mechanisms [133] led to Transformer-type language models, whose size has continually grown: from BERT [29] with its 340 million parameters, to GPT-3 [10] with 175 billion parameters, up to the most recent models exceeding a trillion parameters [35].

This pursuit of gigantism is also observed in other domains: in computer vision with models like ViT [31] or CLIP [99], and in multimodal models such as DALL-E [103] or Flamingo [1]. The impressive results obtained by these massive models continue to empirically validate the "scaling law" hypothesis [64], according to which performance follows a power law based on model size, the amount of training data, and the computing power used. Subsequent work has further refined these scaling relationships, with research on compute-optimal training [53] and continued scaling demonstrations with models like PaLM [21].

1.1.1 Problem Statement: Computational Cost, Latency, and Energy

This vertiginous rise in model size is not without consequences. Contemporary models require colossal computing infrastructures, with considerable energy and financial costs. Training a model like GPT-3 requires several million dollars in computing resources [122], while its carbon footprint becomes concerning in the context of a climate crisis [5].

The discovery of scaling laws has driven the research community to create increasingly larger models, but this approach introduces several critical challenges:

- **Unsustainable computational costs:** Training state-of-the-art models requires millions of dollars in computing resources and produces significant carbon emissions.[1]
- **Inference bottlenecks:** Deploying these massive models creates severe latency issues, making real-time applications impractical in many scenarios.

[1]Regarding these impacts, the work presented in this thesis only focus on first order effects. We consider that second and third order effects are out of the scope of our work, although we are aligned with the vision presented in [82] to acknowledge their critical importance.

- **Limited accessibility:** Only major technology companies can afford the infrastructure needed to serve these models, creating barriers to broader adoption and research.

- **Edge deployment constraints:** Resource limitations make it nearly impossible to run these models on peripheral devices or IoT systems where they could provide the most value.

The rush toward larger models has potentially caused us to miss significant efficiency improvements that could deliver comparable results with substantially smaller computing budgets. In today's world, most of our models are over-parameterized [53, 81] and therefore inherently computationally inefficient. By developing methods specifically adapted to these model sizes—focusing on smarter inference techniques and architectural optimizations—we might achieve similar performance while dramatically reducing resource requirements.

These challenges highlight a fundamental tension between model performance and computational efficiency. As models continue to grow, this tension only intensifies, creating an urgent need for optimization techniques that can reduce computational costs without significantly compromising performance. Among these approaches, early exit methods show particular promise by dynamically adapting computational complexity to the difficulty of each individual input.

To address these computational limitations, the research community has developed numerous techniques aimed at reducing the computational burden of large models while preserving their performance. Methods such as model pruning [72], knowledge distillation [52], and quantization [60] have shown significant success in compressing models and accelerating inference. Pruning techniques remove redundant connections or entire neurons, distillation transfers knowledge from large teacher models to smaller student networks, and quantization reduces the numerical precision of model parameters. However, these approaches share a common limitation: they treat all inputs uniformly, applying the same level of computational reduction regardless of the inherent difficulty or complexity of individual data samples. This one-size-fits-all approach fails to leverage a fundamental characteristic of real-world data—the significant variability in computational requirements across different inputs. Some samples can be correctly classified with minimal processing, while others require the full capacity of deep networks to achieve accurate predictions.



Figure 1.1: Examples illustrating the varying computational requirements across different ImageNet classes. Images of volcanoes (top) represent visually distinctive samples that can be classified early with high confidence due to their clear visual characteristics, while images of red wine (bottom) require deeper processing due to subtle visual features and potential confusion with similar classes. This fundamental variability motivates adaptive inference methods that can dynamically adjust computational complexity based on input-specific requirements [57].

This observation opens the door to a more nuanced approach: adaptive inference methods that can dynamically adjust computational complexity based on input-specific requirements.

1.1.2 Early Exit Methods: A Promising Direction

Among the various approaches to address computational efficiency challenges, early exit methods offer a particularly elegant solution. Rather than forcing every input to traverse the entire network depth, these methods enable models to make predictions at intermediate layers when sufficient confidence is reached.

The core insight is that not all inputs require the same computational complexity. Simple patterns can often be recognized early in the network, while only challenging cases truly benefit from the full model capacity. Early exit architectures exploit this by equipping networks with multiple intermediate prediction heads, allowing dynamic exit decisions based on confidence levels.

However, while this concept appears elegant in principle, the central challenge lies in the exit decision mechanism itself: *how do we accurately determine, for each individual input, at which layer the model should exit to achieve the optimal trade-off between computational cost and prediction accuracy?* This exit selection problem represents the core technical challenge that must be tackled by the field to realize faster yet accurate models.

1.1.3 Roadmap

Having established the foundational concepts and mathematical framework for early exit methods, this introduction will now delve deeper into the three critical aspects that define these architectures. The following sections will systematically examine the architectural design choices that enable dynamic inference, the training methodologies required to optimize multi-exit networks, and the inference strategies that determine when and where to exit during prediction.

Each of these components presents unique challenges and opportunities. The architectural considerations determine how intermediate predictions are integrated into existing networks, the training procedures must balance performance across multiple exit points, and the inference mechanisms must make real-time decisions about computational trade-offs. Understanding these interconnected aspects is essential for developing effective early exit systems that can deliver both accuracy and efficiency in practical applications.

In what follows, we build a specific formalism for early exit suitable for a better understanding of our subsequent contributions to the field. We introduce in Chapter 3 a budget-constrained methodology that explicitly accounts for computational limits when designing and deploying early exit policies. This framework provides a principled approach to optimizing the accuracy–efficiency trade-off under realistic resource constraints. Subsequently, the following Chapter 4 extends the analysis to large language models (LLMs), demonstrating how early exit mechanisms can be adapted to sequential generation tasks. Together, these contributions aim to provide a comprehensive perspective on early exit methods, ranging from theoretical foundations to practical applications in modern large-scale models.

1.2 Architecture of Early Exit Networks

The development of early exit architectures represents a paradigm shift from traditional monolithic neural networks to adaptive, dynamic inference systems. This section traces the historical evolution of these architectures, from the earliest convolutional networks to

the sophisticated multi-exit systems of today, highlighting the key innovations that have shaped this field.

1.2.1 Mathematical Foundations

Before examining the historical evolution and design principles of early exit architectures, we establish the mathematical framework that will be used throughout this analysis. This formalization provides the necessary notation and concepts for understanding how early exit networks extend traditional deep learning models.

Dataset Formalization

Let us define a dataset $\mathcal{D}$ as a collection of input-output pairs:

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N} \tag{1.1}$$

where $N \in \mathbb{N}$ represents the total number of samples in the dataset, $\mathbf{x}_i \in \mathcal{X}$ denotes an input sample from the input space $\mathcal{X}$, and $y_i \in \mathcal{Y}$ represents the corresponding target output from the output space $\mathcal{Y}$.

For experimentation and evaluation purposes, we partition the dataset $\mathcal{D}$ into three disjoint subsets:

- **Training set** $\mathcal{D}_{\text{train}} \subset \mathcal{D}$: Used to optimize the parameters of our models.
- **Validation set** $\mathcal{D}_{\text{val}} \subset \mathcal{D}$: Used for hyperparameter tuning and early stopping.
- **Test set** $\mathcal{D}_{\text{test}} \subset \mathcal{D}$: Reserved for the final unbiased evaluation of model performance.

These sets satisfy $\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{val}} \cup \mathcal{D}_{\text{test}} = \mathcal{D}$ and are mutually exclusive.

Input and Output Spaces

In this thesis, we focus on two primary application domains: computer vision and language modeling. The input and output spaces differ significantly between these domains:

Computer Vision: For image classification tasks, the input space $\mathcal{X}$ consists of 3D tensors:

$$\mathbf{x} \in \mathbb{R}^{H \times W \times C}$$

where H represents the height, W the width, and C the number of channels of the image (typically $C = 3$ for RGB images). The corresponding output space $\mathcal{Y}$ for our multi-class single-label classification setting is:

$$\mathcal{Y} = \{1, 2, \ldots, K\}$$

where K is the number of distinct classes. Alternatively, we can represent the output as a one-hot encoded vector $\mathbf{y} \in \{0, 1\}^K$ whose components are such that $\sum_{j=1}^{K} y_j = 1$.

Language Modeling: For language tasks, the input space $\mathcal{X}$ consists of sequences of tokens:

$$\mathbf{x} = (x^{(1)}, x^{(2)}, \ldots, x^{(L)}) \in \mathcal{V}^L$$

where $\mathcal{V}$ is the vocabulary (the set of all possible tokens), and L is the context length (the maximum sequence length). A token represents a basic unit of text, which can be a word, subword, or character depending on the tokenization scheme used, and is most of the time represented as an integer. In autoregressive language modeling, the output space $\mathcal{Y}$ also consists of token sequences:

$$\mathbf{y} = (y^{(1)}, y^{(2)}, \ldots, y^{(L)}) \in \mathcal{V}^L$$

where each $y^{(t)} = x^{(t+1)}$ for $t \in \{1, 2, \ldots, L-1\}$, representing the next-token prediction task. The final token $y^{(L)}$ typically corresponds to a special end-of-sequence token, the token following the sentence if the input sequence was truncated, or is ignored during training.

Model Architecture

In this thesis, we focus on deep neural network models for classification tasks. We formalize our model $f : \mathcal{X} \to \mathcal{Y}$ as a composition of two main components: a backbone network and a classification head. This decomposition is particularly useful for analyzing the model's behavior and for implementing early-exit techniques.

Model Formalization: A deep learning model f can be expressed as:

$$f(\mathbf{x}) = \psi(\phi(\mathbf{x}))$$

where:

- $\phi : \mathcal{X} \to \mathcal{Z}$ is the backbone network that maps the input to a latent representation.
- $\psi : \mathcal{Z} \to \mathcal{Y}$ is the classification head that maps the latent representation to the output space.
- $\mathcal{Z}$ is the final latent space, typically $\mathcal{Z} \subset \mathbb{R}^d$ for some dimension d. More generally, the backbone generates intermediate representations at different layers with varying dimensions d_i, creating a sequence of latent spaces $\mathcal{Z}^i \subset \mathbb{R}^{d_i}$ for $i = 1, 2, \ldots, E$, where $\mathcal{Z}^E \equiv \mathcal{Z}$. Here E represents the number of layer in the backbone.

Parameter Spaces: To ensure mathematical rigor, we define a comprehensive hierarchical parameter space framework that enables simplified notation throughout this thesis. Let $P \in \mathbb{N}$ denote the total number of parameters in the model. The complete parameter vector $\boldsymbol{\theta}$ lives in the parameter space:

$$\boldsymbol{\theta} \in \Theta \subset \mathbb{R}^P$$

where Θ represents the feasible parameter space, which may include constraints such as normalization conditions or architectural restrictions. The parameters decompose hierarchically as:

$$\boldsymbol{\theta} = (\boldsymbol{\theta}_\phi, \boldsymbol{\theta}_\psi) \in \Theta_\phi \times \Theta_\psi$$

where $\Theta = \Theta_\phi \times \Theta_\psi$.

Layer-wise Parameter Decomposition: The backbone parameters $\boldsymbol{\theta}_\phi$ further decompose into layer-specific tensors:

$$\boldsymbol{\theta}_\phi = (\theta_\phi^1, \theta_\phi^2, \ldots, \theta_\phi^E) \in \Theta_\phi^1 \times \Theta_\phi^2 \times \cdots \times \Theta_\phi^E$$

where each $\theta_\phi^i \in \Theta_\phi^i$ contains the parameters for the i-th backbone layer. Similarly, for early exit architectures, the head parameters $\boldsymbol{\theta}_\psi$ decompose as:

$$\boldsymbol{\theta}_\psi = (\theta_\psi^1, \theta_\psi^2, \ldots, \theta_\psi^M) \in \Theta_\psi^1 \times \Theta_\psi^2 \times \cdots \times \Theta_\psi^M$$

where each $\theta_\psi^i \in \Theta_\psi^i$ contains the parameters for the i-th classification head.

Remark 1.2.1. The parameters M and E denotes the number of layers and the number of exits respectively. In this setting, $M \leq E$ meaning that we do not put an exit at the level of each layer.

Implicit Parameter Usage Philosophy: With this comprehensive parameter space framework, we adopt the convention that function calls implicitly reference their associated parameters. Specifically:

- $\phi(\cdot)$ implicitly uses parameters $\boldsymbol{\theta}_\phi \in \Theta_\phi$;
- $\psi(\cdot)$ implicitly uses parameters $\boldsymbol{\theta}_\psi \in \Theta_\psi$.

This notation significantly simplifies mathematical expressions while maintaining full mathematical rigor through the explicit parameter space definitions provided above.

Backbone Architecture: The backbone ϕ can be further decomposed into a sequence of E layers or blocks:

$$\phi(\cdot) = (\phi^E \circ \phi^{E-1} \circ \cdots \circ \phi^1)(\cdot)$$

where for each $i \in \{1, \ldots E\}$, the mapping $\phi^i : \mathcal{Z}^{i-1} \to \mathcal{Z}^i$ represents the i-th layer or block of the network with parameters $\theta_\phi^i \in \Theta_\phi^i$. Here, $\mathcal{Z}^0 = \mathcal{X}$ is the input space and $\mathcal{Z}^E = \mathcal{Z}$ is the latent space.

Backbone Parameter Structure: The backbone parameters $\boldsymbol{\theta}_\phi$ encompass all learnable parameters across the network layers. The specific structure of Θ_ϕ^i depends on the layer type. For instance, for convolutional neural networks (CNNs) applied to image classification, the backbone typically consists of convolutional layers, pooling operations, and activation functions. For language models based on transformers, the backbone consists of self-attention mechanisms and feed-forward neural networks organized in blocks. Below is a formalization in these two popular architectures.

Convolutional Layer: For a convolutional layer i with C_{in}^i input channels, C_{out}^i output channels, and kernel size $K^i \times K^i$:

$$\theta_\phi^i = (W_{\text{conv}}^i, b_{\text{conv}}^i) \in \mathbb{R}^{c_{\text{out}}^i \times c_{\text{in}}^i \times K^i \times K^i} \times \mathbb{R}^{c_{\text{out}}^i}$$

where W_{conv}^i is the convolution kernel and b_{conv}^i is the bias vector for layer i.

Transformer Attention Layer: For a multi-head attention layer with model dimension d_{model}, key dimension d_k, and value dimension d_v:

$$\theta_\phi^i = (W_Q^i, W_K^i, W_V^i, W_O^i) \in \mathbb{R}^{d_{\text{model}} \times d_k} \times \mathbb{R}^{d_{\text{model}} \times d_k} \times \mathbb{R}^{d_{\text{model}} \times d_v} \times \mathbb{R}^{d_v \times d_{\text{model}}}$$

where W_Q^i, W_K^i, W_V^i are query, key, and value projection matrices, and W_O^i is the output projection matrix for layer i.

Classification Heads: Classification heads are classical Multi layer perceptron (MLP), thus each head component θ_ψ^i of $\boldsymbol{\theta}_\psi$ with $i \in \{1, \ldots, M\}$ can be decomposed as:

$$\theta_\psi^i = (W^i, b^i) \in \mathbb{R}^{K \times d_i} \times \mathbb{R}^K$$

where the weight matrix W^i has dimensions $K \times d_i$ to map from the d_i-dimensional latent space $\mathcal{Z}^i$ to the K-dimensional output space, and the bias vector b_i provides translation in the output space. Let us denote by Δ^{K-1} the $(K-1)$-simplex. The classification head $\psi^i : \mathcal{Z}^i \to \Delta^{K-1}$ maps the latent representation to a probability distribution over the K classes:

$$\psi^i(\boldsymbol{z}) = \text{softmax}(W^i \boldsymbol{z} + b^i)$$

for all $\boldsymbol{z} \in \mathcal{Z}^i$ where the softmax function defined by

$$\text{softmax}(\boldsymbol{v})_i = \frac{e^{v_i}}{\sum_{j=1}^K e^{v_j}} \quad \text{for} \quad i \in \{1, 2, \ldots, K\} \ .$$

The linear transformation followed by softmax normalization ensures that the output $\psi^i(\boldsymbol{z}) \in \Delta^{K-1}$ lies on the $(K-1)$-simplex, representing a valid probability distribution over the class labels.

Let $\{e_1, e_2, \ldots, e_M\}$ denote the set of M exit locations, where each $e_i \in \{1, \ldots, E\}$ indicates which backbone layer the i-th classification head is attached to. These attachment points are strictly ordered ($e_1 < e_2 < \ldots < e_M = E$), ensuring that successive heads have access to progressively deeper representations. The constraint $e_M = E$ guarantees that the final head utilizes the complete backbone network.

The above decomposition into backbone and classification head is particularly important for our work on early exit methods, as it allows us to attach additional classification heads at intermediate layers of the backbone to enable predictions at different computational depths.

Remark 1.2.2. Notice that the classical setting where there is no early exit corresponds to the case where $M = 1$.

Early Exit Architecture

Building on the backbone-head decomposition presented above, early exit models augment the standard architecture with intermediate classification heads to enable predictions at multiple computational depths. This architectural modification transforms a traditional deep network into a flexible system capable of dynamic inference.

An early exit model f_{exit} with M possible exit heads can be expressed as a collection of classifiers operating on partial backbone computations. Indeed, for each exit head $i \in \{1, 2, \ldots, M\}$, the model provides the probabilistic classifier:

$$\psi^i(\phi^{1:e_i}(\cdot)) : \mathcal{X} \to \Delta^{K-1} \quad \text{for } i = 1, 2, \ldots, M$$

where:

- $M \leq E$ is the number of exit heads distributed throughout the network;
- $\phi^{1:e_i}(\mathbf{x}) = \phi^{e_i} \circ \phi^{e_{i-1}} \circ \cdots \circ \phi^1(\mathbf{x})$ represents the partial backbone computation up to layer e_i, where the i-th head is located, for the instance $\mathbf{x}$, using parameters $\theta_\phi^1, \ldots, \theta_\phi^{e_i}$;
- $\psi^i : \mathcal{Z}^{e_i} \to \Delta^{K-1}$ is the i-th intermediate classification head, using parameters $\theta_\psi^{e_i}$.

The complete early exit model thus consists of M classifiers:

$$f_{\text{exit}}(\cdot) = \{\psi^i(\phi^{1:e_i}(\cdot))\}_{i=1}^{M} \ . \tag{1.2}$$

In particular for a classification task, the prediction for each instance x at the level of each head i is obtained by taking the argmax of the softmax output:

$$\hat{g}^i(\mathbf{x}) = \text{argmax}(\psi^i(\phi^{1:e_i}(\mathbf{x})) \tag{1.3}$$

where each head i uses parameters $\theta_\psi^i = (W^i, b^i) \in \mathbb{R}^{K \times d_i} \times \mathbb{R}^K$. The key difference lies in the dimensionality d_i of the latent representation, which might vary across layers as features become increasingly refined through the network depth.

Remark 1.2.3. The classification (1.3) based on argmax of probability scores is a classical approach to output the most likelihood class. However, other classification strategies might be used. In this thesis we will focus on the argmax selection given that it is a widely used method.

1.2.2 Historical Evolution of Multi-Exit Architectures

For a better comprehension of the framework, we will now briefly summarize the historical evolution of the field. A more specific and in-depth analysis of each paper will be performed in chapter 2.

Monolithic Classification Networks

The foundation of modern deep learning was established with the development of convolutional neural networks that processed every input through the entire network depth. The seminal work of LeCun et al. [73] with LeNet-5 demonstrated the effectiveness of hierarchical feature learning for handwritten digit recognition, establishing the basic paradigm of sequential layer processing. This approach was further validated by the breakthrough performance of AlexNet [68] in the ImageNet challenge, which showed that deeper networks with increased computational capacity could achieve unprecedented accuracy in image classification.

These early architectures shared two fundamental characteristics that, while contributing to their success, also introduced critical inefficiencies:

First, uniform computational processing: Every input, regardless of its complexity or the confidence of intermediate predictions, was required to traverse the complete network depth. This design choice, while ensuring maximum feature extraction capability, introduced a critical limitation—the computational cost scaled linearly with network depth for every single input. Simple patterns that could be recognized with basic features were forced to undergo the same extensive processing as complex, ambiguous cases that truly required deep feature analysis.

Second, hierarchical feature learning without adaptive utilization: Deep CNNs were specifically designed with a hierarchical architecture where early layers extract simple, low-level features (edges, textures, basic shapes) while deeper layers progressively build more complex, abstract representations [153]. This hierarchical design principle proved remarkably effective for achieving high overall accuracy, as it mirrors the structure of biological visual processing systems [58]. However, this hierarchical organization was not fully exploited for computational efficiency. Many inputs could be correctly classified using only the simple features extracted by early layers, yet the architecture forced all inputs to undergo the complete hierarchical processing chain. This represents a fundamental mismatch between the adaptive nature of the problem—where different inputs require

different levels of feature complexity—and the rigid, uniform processing imposed by traditional architectures.

The computational implications of this uniform processing became increasingly problematic as networks grew deeper. While VGG [120] demonstrated that depth could significantly improve performance, the computational cost per inference scaled proportionally. ResNet [50] enabled even deeper architectures through residual connections, but the fundamental issue remained: every input consumed the full computational budget, regardless of whether simple features were sufficient for accurate classification.

This uniform processing paradigm created several critical inefficiencies:

- **Computational waste on simple inputs:** Easy-to-classify samples received the same processing as challenging cases, leading to unnecessary computations.
- **Inflexible resource allocation:** The inability to adapt computational effort based on input complexity resulted in suboptimal resource utilization.
- **Latency bottlenecks:** Real-time applications were constrained by worst-case processing times, as all inputs required full network traversal.
- **Energy inefficiency:** Battery-constrained devices expended maximum energy on every prediction, regardless of the actual computational requirements.

Early Auxiliary Heads: Training Stabilization (2014)

The first departure from purely sequential processing emerged with the introduction of auxiliary classifiers in the GoogLeNet/Inception architecture [125]. This work introduced intermediate classification heads at specific points within the network, but with a fundamentally different motivation than modern early exit systems.

The auxiliary classifiers in GoogLeNet were designed to address the *vanishing gradient problem* in very deep networks. By providing additional gradient signals at intermediate layers, these auxiliary heads helped stabilize training and improve convergence. The network architecture included two auxiliary classifiers positioned at strategic points inside the network. During training, these auxiliary heads contributed to the total loss with reduced weights (typically 0.3), providing regularization and improved gradient flow.

These auxiliary heads were *not designed for dynamic inference*. During inference, only the final classifier was used for predictions, and the auxiliary heads were simply ignored. The computational cost remained identical to traditional monolithic networks, as every input still traversed the complete network depth. The auxiliary heads served purely as training aids, improving the optimization landscape but providing no inference-time benefits.

The key innovations introduced by GoogLeNet's auxiliary classifiers were:

- **Multi-point supervision:** Demonstration that networks could be trained with multiple loss signals at different depths.
- **Gradient stabilization:** Empirical evidence that intermediate supervision could improve training dynamics in deep networks.
- **Architectural modularity:** The concept that classification heads could be attached at arbitrary network depths without disrupting the main processing flow.

While not intended for dynamic inference, these auxiliary classifiers established technical foundations that would later enable true early exit systems. They demonstrated that intermediate representations could produce meaningful predictions and that multiple classification heads could coexist within a single architecture.

Birth of Dynamic Inference (2015-2016)

The transition from training-time auxiliary heads to inference-time dynamic processing began with pioneering work on cascade architectures and adaptive computation. The seminal contribution came from Teerapittayanon et al. [128] with BranchyNet, which introduced the first architecture explicitly designed for *dynamic inference cost reduction*.

BranchyNet represented a fundamental paradigm shift by enabling different inputs to exit at different network depths based on prediction confidence. The architecture featured multiple "branch points" throughout the network, each equipped with a classification head and a confidence estimation mechanism. During inference, if the prediction confidence at a given branch exceeded a predefined threshold, processing would terminate and return the prediction. Only inputs requiring additional processing would continue to deeper layers.

Concurrently, the concept of cascade architectures was formalized through works on multi-scale processing and hierarchical classification. The Big-Little Net approach [63] introduced the idea of using a small, fast network as a first-stage filter, followed by a larger, more accurate network for challenging cases. This two-stage cascade demonstrated that *most inputs could be correctly classified with minimal computation*, requiring the full network capacity only for a subset of difficult examples.

The period from 2015-2016 established several fundamental principles of dynamic inference:

- **Confidence-based early termination:** The use of prediction confidence as a proxy for processing adequacy became the dominant paradigm for exit decisions.
- **Threshold-based control:** Simple threshold mechanisms provided an intuitive way to balance accuracy and computational cost.
- **Input-adaptive processing:** Recognition that different inputs have fundamentally different computational requirements led to the development of adaptive algorithms.
- **Multi-stage architectures:** The demonstration that networks could be decomposed into sequential processing stages with intermediate decision points.

Emergence of Multiple Paradigms (2017-Present)

Following the initial breakthrough of dynamic inference, the field rapidly diversified into multiple architectural paradigms, each addressing different aspects of adaptive computation. This diversification was driven by the recognition that different application domains, hardware constraints, and accuracy requirements demanded specialized approaches to early exit design. This section provides a detailed examination of the three dominant architectural approaches that define the current state of early exit networks.

Cascade Architectures Cascade architectures evolved from the Big-Little Net[63] concept into sophisticated multi-stage systems. Notable developments include the work by Wang et al. [136] on multi-scale cascade networks and Li et al. [79] on hierarchical neural networks. These systems decompose the inference process into a sequence of independent models $\{f^1, f^2, \ldots, f^C\}$ where each stage can either provide a final prediction or defer to the next stage. Each f^c relies on parameters $\boldsymbol{\theta}^c$ that need to be trained. Before going in the details of the training strategies to build f^c, let us first describe the general cascade process

Mathematical Framework: The cascade decision process can be formalized as a sequential filtering mechanism. For $c \in \{1, \ldots, C\}$, let $\tau^c \in [0, 1]$ be the confidence

threshold for stage c given by the user. For an input $\mathbf{x} \in \mathcal{X}$, the cascade processes it through stages until a confident prediction is obtained:

$$\begin{aligned} \text{Stage } c: \quad & \hat{\mathbf{p}}^c = f^c(\mathbf{x}) \\ & \hat{g}^c(\mathbf{x}) = \text{argmax}_{y \in \{1,\ldots,K\}} \hat{p}^c(y|\mathbf{x}) \\ & \text{Confidence: } \gamma^c = \max_{y \in \{1,\ldots,K\}} \hat{p}^c(y|\mathbf{x}) \\ & \text{Decision: } \delta^c(\mathbf{x}) = \mathbf{1}[\gamma^c > \tau^c] \end{aligned}$$

where $\mathbf{1}[\cdot]$ stands for the indicator function, $\hat{\mathbf{p}}^c = (\hat{p}^c(1|\mathbf{x}), \ldots, \hat{p}^c(K|\mathbf{x})) \in \Delta^{K-1}$ with $\hat{p}^c(y|\mathbf{x})$ representing the estimated conditional probability of class $y \in \mathcal{Y}$ produced by stage c and $\hat{g}^c(\mathbf{x})$ is the predicted class label for instance $\mathbf{x}$ obtained by taking the argmax of the probability distribution. The confidence measure γ^c captures the maximum predicted probability, indicating how certain stage c is about its prediction. The final prediction is determined by the first stage that exceeds its confidence threshold:

$$\hat{g}_{\text{final}}(\mathbf{x}) = \hat{g}^c(\mathbf{x}) \text{ where } c = \min\{j \in \{1, \ldots C\} : \delta^j(\mathbf{x}) = 1\} \ .$$

Training Strategies: Cascade architectures support multiple training paradigms:
Independent Training: Each stage is optimized separately:

$$\hat{\boldsymbol{\theta}}^c = \text{argmin}_{\boldsymbol{\theta}^c} \frac{1}{N} \sum_{(\mathbf{x},y) \in \mathcal{D}_{\text{train}}} \mathcal{L}_{\text{CE}}(f^c(\mathbf{x}), \tilde{\mathbf{y}})$$

where $\tilde{\mathbf{y}} = (\tilde{y}_1, \ldots, \tilde{y}_K) \in \{0,1\}^K$ is the one-hot encoded true label y that is, $\tilde{y}_k = \mathbf{1}[y = k]$, and the cross-entropy loss is defined as $\mathcal{L}_{\text{CE}}(\hat{\mathbf{p}}, \tilde{\mathbf{y}}) = -\sum_{k=1}^K \tilde{y}_k \log(\hat{p}_k)$, with $\hat{\mathbf{p}} = (\hat{p}_1, \ldots, \hat{p}_K) \in \Delta^{K-1}$ being the predicted probability distribution.
Sequential Training: Later stages are trained on samples that earlier stages handle poorly:

$$\mathcal{D}^c = \{(\mathbf{x}, y) \in \mathcal{D}_{\text{train}} : \delta^{c-1}(\mathbf{x}) = 0\} \ .$$

End-to-end Training: Joint optimization with cascade-specific losses that encourage stage specialization.
Key characteristics:

- Stage specialization: Each cascade stage can be optimized for specific input characteristics or computational constraints
- Independent training: Later stages can be trained separately, allowing for heterogeneous architectures and optimization strategies
- Flexible deployment: Different stages can be deployed on different hardware or locations (e.g., edge-cloud architectures)
- Incremental updates: Individual stages can be updated without affecting others
- Fault tolerance: System can gracefully degrade if individual stages fail

Multi-Branch Architectures Multi-branch architectures explore parallel processing paths with different computational complexities, exemplified by the MobileNets family [55, 110] and EfficientNet [126]. This paradigm differs from sequential approaches by processing inputs through multiple branches concurrently before making exit decisions.
Architectural Design: A multi-branch network consists of:

- A shared feature extractor $\phi^0 : \mathcal{X} \to \mathcal{Z}$ that maps instances to some latent space;

- Multiple processing branches $\{\phi^j\}_{j=1}^M$, with $\phi^j : \mathcal{Z} \to \mathcal{Z}^j$, each ϕ^j having its own complexity.
- Branch-specific classification heads $\{\psi^j\}_{j=1}^M$ where for each $j \in \{1, \ldots, M\}$ the mapping $\psi^j : \mathcal{Z}^j \to \Delta^{K-1}$ is a classical MLP predictor.
- A fusion mechanism that combines or selects from branch outputs

The mathematical formulation is:

$$\begin{aligned} \mathbf{z}^0 &= \phi^0(\mathbf{x}) \\ \mathbf{z}^j &= \phi^j(\mathbf{z}_0) \quad \forall j \in \{1, \ldots, M\} \\ \hat{\mathbf{p}}^j &= \psi^j(\mathbf{z}_j) \quad \forall j \in \{1, \ldots, M\}. \end{aligned}$$

The classification for $\mathbf{x}$ is then deduced from the output probabilites $\hat{\mathbf{p}}^1, \ldots, \hat{\mathbf{p}}^M$. Each $\hat{\mathbf{p}}^j$ is a vector of the simplex and can be written as $\hat{\mathbf{p}}^j = (\hat{p}_1^j, \ldots, \hat{p}_K^j)$.

Branch Selection Strategies: Multi-branch architectures employ various strategies for selection:

Confidence-based Selection: This strategy chooses the branch with highest prediction confidence:

$$\hat{j} = \text{argmax}_{j \in \{1, \ldots M\}} \max_{k \in \{1, \ldots K\}} \hat{p}_k^j \ .$$

Learned Selection: Use a gating network g to predict optimal branch weights, where g is a neural network (typically a small MLP) trained to output probability weights. This can operate on either the shared representation or all branch outputs:

$$\mathbf{w} = h(\mathbf{z}^0) \quad \text{(shared-based)} \quad \text{or} \quad \mathbf{w} = h(\{\mathbf{z}^j\}_{j=1}^M) \quad \text{(branch-based)}$$

$$\hat{\mathbf{p}}_{\text{final}} = \sum_{j=1}^{M} w^j \hat{\mathbf{p}}^j,$$

and the classification is deduced from the probabilities in $\hat{\mathbf{p}}_{\text{final}}$.

Multi-branch architectures can achieve efficiency through different mechanisms depending on the selection strategy. Confidence-based selection enables early termination when a branch is sufficiently confident, while learned selection trades computational cost for improved accuracy by combining predictions from all branches. Additional efficiency can be gained through conditional execution based on shared feature analysis and resource sharing between branches.

Multi-Exit Networks Multi-exit networks represent the most direct evolution of BranchyNet, featuring multiple intermediate classification heads positioned throughout a single backbone architecture ϕ. Representative works include MSDNets [57] 1.2, which introduced the concept of multi-scale dense networks with multiple exit points, and SBERT [145], which adapted the approach to transformer-based language models. In this paradigm, a network is augmented with M exit points positioned at different depths, where each exit i consists of a classification head ψ^i with parameters θ_ψ^i.

Mathematical Formulation: Consider a backbone network ϕ with E layers. A multi-exit architecture augments this backbone with M intermediate classification heads positioned at different depths. The mathematical formulation is:

$$\begin{aligned} \text{Partial backbone up to exit } i: &\quad \phi^{1:e_i} : \mathcal{X} \to \mathcal{Z}^{e_i} \\ \text{Classification head at exit } i: &\quad \psi^i : \mathcal{Z}^{e_i} \to \Delta^{K-1} \\ \text{Prediction at exit } i: &\quad \hat{\mathbf{p}}^i = \psi^i(\phi^{1:e_i}(\mathbf{x})), \qquad \text{for all } \mathbf{x} \in \mathcal{X} \end{aligned}$$

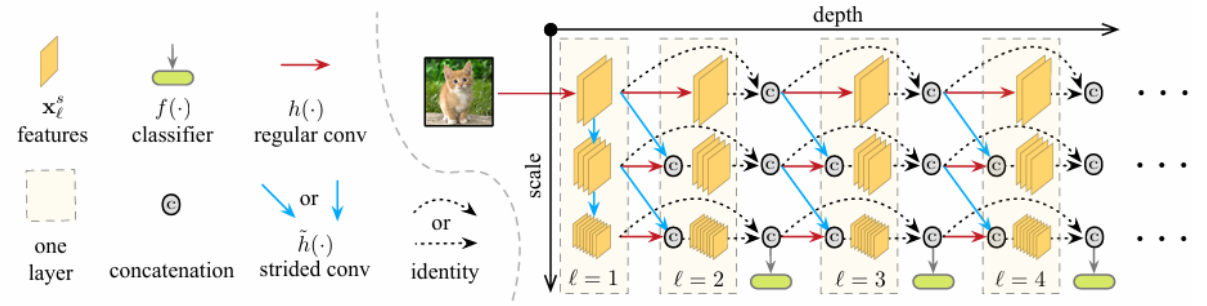


Figure 1.2: Figure from [57] which illustrate the first four layers of an MSDNet with three scales. The horizontal direction corresponds to the layer direction (depth) of the network. The vertical direction corresponds to the scale of the feature maps. Horizontal arrows indicate a regular convolution operation, whereas diagonal and vertical arrows indicate a strided convolution operation. Classifiers only operate on feature maps at the coarsest scale.

where $\phi^{1:e_i}(\mathbf{x}) = \phi^{e_i} \circ \phi^{e_{i-1}} \circ \cdots \circ \phi^1(\mathbf{x})$ represents the partial backbone computation through the first e_i layers to reach the location of head i, producing intermediate representations in the latent space $\mathcal{Z}^{e_i}$. For all $i \in \{1, \ldots, M\}$, the exit decision mechanism δ^i determines whether to terminate processing at exit i based on prediction confidence $\hat{\mathbf{p}}^i = (\hat{p}_1^i, \ldots, \hat{p}_K^i) \in \Delta^{K-1}$. For example, using maximum probability as confidence:

$$\delta^i(\mathbf{x}) = \begin{cases} 1 & \text{if } \max_k \hat{p}_k^i > \tau^i \\ 0 & \text{otherwise} \end{cases} \tag{1.4}$$

where $\tau^i \in [0, 1]$ is the confidence threshold for exit i determined by the practitioner.

Multi-exit networks offer several key advantages:

- **Fine-grained control**: Multiple exit points provide granular control over the accuracy-efficiency trade-off
- **Unified optimization**: All components are trained jointly, ensuring consistent feature representations across exits
- **Progressive refinement**: Each exit can leverage increasingly sophisticated features from deeper layers
- **Implementation simplicity**: Minimal modifications to existing architectures are required
- **Memory efficiency**: Shared backbone reduces overall parameter count compared to cascade approaches

Advanced and Hybrid Approaches Recent developments have explored sophisticated combinations and extensions of these core paradigms:

Hierarchical Architectures [25, 119] organize predictions in tree-like structures, exploiting the natural hierarchy of classification problems. The Adaptive Computation Time (ACT) framework [43] introduced differentiable dynamic inference, where computation steps are learned rather than predetermined.

Cascade-Multi-Exit Hybrids combine cascade and multi-exit concepts by employing multi-exit networks as individual cascade stages, providing both the specialization benefits of cascades and the fine-grained control of multi-exit systems.

Attention-Based Exit Selection incorporates attention mechanisms to make more informed exit decisions, considering not just confidence but also input characteristics and learned representations.

Neural Architecture Search for Exits uses automated approaches to optimize exit placement, branch design, and threshold selection simultaneously, leading to architectures specifically optimized for early exit performance.

Recent developments (2020-present) have extended early exit concepts to new domains: transformer-based architectures for large language models [145, 114], hardware-aware optimization for edge devices [77], multi-modal processing [16], and federated learning integration [138].

This comprehensive overview of modern early exit architectures demonstrates the richness and diversity of approaches available for implementing adaptive inference systems. Each paradigm offers distinct advantages and trade-offs, enabling practitioners to select the most appropriate architecture based on their specific requirements for accuracy, efficiency, and deployment constraints.

1.3 Training Methodologies for Early Exit Networks

The effectiveness of early exit networks depends critically on training methodologies that can simultaneously optimize multiple prediction points while maintaining the overall network performance. This section examines the fundamental challenges and opportunities in training early exit systems, from basic mathematical formulation to advanced inference-aware approaches.

1.3.1 Mathematical Formulation

As established in the Architecture Section 1.2, an early exit model f_{exit} consists of a backbone network with parameters $\boldsymbol{\theta}_\phi$ and multiple classification heads with parameters $\boldsymbol{\theta}_\psi = (\theta_\psi^1, \ldots, \theta_\psi^M)$. The complete parameter set is:

$$\boldsymbol{\theta}_{\text{exit}} = (\boldsymbol{\theta}_\phi, \{\theta_\psi^i\}_{i=1}^M) \in \Theta_\phi \times \prod_{i=1}^M \Theta_\psi^i \ .$$

The training objective is to optimize these parameters to maximize the accuracy of each classification head. For each exit i, we seek to minimize the prediction error on the training set $\mathcal{D}_{\text{train}}$:

$$(\hat{\theta}_\psi^i, \hat{\boldsymbol{\theta}}_\phi^{1:e_i}) = \text{argmin}_{\theta_\psi^i, \boldsymbol{\theta}_\phi^{1:e_i}} \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{train}}} \mathcal{L}(\psi^i(\phi^{1:e_i}(\mathbf{x})), y)$$

where $\boldsymbol{\theta}_\phi^{1:e_i} = (\theta_\phi^1, \ldots, \theta_\phi^{e_i})$ and the same convention is used for $\hat{\boldsymbol{\theta}}_\phi^{1:e_i}$. Moreover $\phi^{1:e_i}$ represents the partial backbone network up to layer e_i (parametrized by $\boldsymbol{\theta}_\phi^{1:e_i}$), ψ^i is the i-th classification head (parametrized by θ_ψ^i), and $\mathcal{L}$ represents an appropriate loss function for the task (*e.g.,* cross-entropy for classification, token-level cross-entropy for language modeling).

1.3.2 Standard Multi-Exit Training

This optimization objective is standard in machine learning and leads naturally to a multi-exit loss formulation. The most straightforward approach combines losses from all exits using a weighted sum: The multi-exit loss is:

$$\mathcal{L}_{\text{multi-exit}}(\mathbf{x}, y) = \sum_{i=1}^M \alpha_i \mathcal{L}(\psi^i(\phi^{1:e_i}(\mathbf{x})), y)$$

where α_i are weighting coefficients that allocate more emphasis to some heads than to others.

1.3.3 Limitations of Standard Training

While this standard approach provides a solid foundation, it introduces several challenges that can significantly impact the effectiveness of early exit systems:

Gradient Flow Issues: The interaction between gradients from multiple exit points can create complex optimization dynamics. Competing objectives from different exits may lead to conflicting gradient directions, potentially hindering the learning of shared backbone representations and causing training instability.

Inference-Unaware Training: The standard approach treats early exit networks as multiple independent classifiers that should all be optimized equally. However, this training paradigm does not reflect how the model is actually used during inference, where only one exit is selected per input based on confidence or other criteria. This mismatch between training and inference can lead to suboptimal performance.

Lack of Exit Coordination: The standard loss formulation does not explicitly encourage the different exits to work together as a cohesive system. Each exit is optimized independently, potentially missing opportunities for exits to complement each other's strengths and weaknesses.

These limitations motivate the need for more sophisticated training approaches that explicitly account for the dynamic nature of early exit inference. We will explore our work on inference-aware training methodologies that address these challenges throughout this manuscript.

1.3.4 Early Exit as a Pluggable Technique

One of the most powerful practical advantages of early exit networks is their ability to be seamlessly integrated into existing pre-trained models. Rather than requiring full model retraining, early exit heads can be added to any pre-trained backbone and trained independently while keeping the backbone parameters frozen:

$$\begin{aligned} \boldsymbol{\theta}_\phi &\leftarrow \text{frozen (no gradient updates)} \\ \{\theta_\psi^i\}_{i=1}^M &\leftarrow \operatorname{argmin}_{\{\theta_\psi^i\}} \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{train}}} \mathcal{L}_{\text{multi-exit}}(\mathbf{x}, y) \ . \end{aligned}$$

This **pluggable nature** transforms early exit methods into a highly practical technique that can be applied to any deep learning model without the computational burden of full retraining.

Significance for Large Language Models: This capability is particularly valuable in the era of large language models (LLMs), where foundation models are trained on billions of tokens using massive computational resources costing millions of dollars. The ability to add early exit capabilities by training only the lightweight classification heads—while preserving the rich representations learned during expensive pre-training—makes early exit methods highly practical and accessible.

Practical Advantages: The pluggable nature of early exit networks offers several key benefits:

- **Computational efficiency**: Only a small fraction of parameters need to be trained
- **Preservation of learned features**: The rich representations from pre-training remain intact
- **Rapid deployment**: Early exit capabilities can be added to existing models in hours rather than weeks
- **Broad applicability**: Can be applied to any pre-trained model across different domains

This pluggable characteristic positions early exit methods as a practical solution for bringing adaptive inference capabilities to the growing ecosystem of large pre-trained models, making efficient inference accessible without the prohibitive costs of full model development.

1.4 Inference Strategies and Dynamic Execution

Having established the architectural foundations and training methodologies for early exit networks, we now turn to the most critical phase of the early exit paradigm: inference. While the design of architectures and training strategies are essential components, the inference phase represents the ultimate test of early exit systems, as it is during inference that the primary objective of computational reduction must be realized.

The fundamental promise of early exit networks lies in their ability to dynamically reduce computational requirements during inference while maintaining acceptable performance levels. This computational efficiency is not achieved through static optimizations but rather through intelligent decision-making processes that determine when sufficient information has been gathered to make reliable predictions. Consequently, the success of any early exit system is ultimately determined by the quality and effectiveness of its inference strategy.

The inference phase introduces a unique challenge that distinguishes early exit networks from traditional deep learning systems: the need for dynamic decision-making based on intermediate outputs. Unlike conventional neural networks that process inputs through a fixed computational path, early exit systems must continuously evaluate whether to terminate computation early or continue processing through deeper layers. This decision-making process requires sophisticated strategies that can balance the competing objectives of computational efficiency and predictive accuracy.

To achieve the computational reduction objectives that motivate early exit research, we require robust exiting strategies that can reliably identify when sufficient processing has been completed for a given input. These strategies must operate effectively across diverse input distributions, varying computational constraints, and different performance requirements, making the development of effective inference mechanisms a critical research challenge in the early exit domain.

Mathematical Framework for Early Exit Inference Before examining specific inference strategies, it is essential to recall the mathematical framework established earlier for early exit networks. As defined in Equation (1.2), we work with an early exit model f_{exit} with M exit points, where each exit i produces predictions $\hat{g}^i(x)$ as defined in Equation (1.3) based on intermediate representations from the backbone network.

For the inference analysis, we introduce inference-specific notation. The accuracy of exit i on the test dataset $\mathcal{D}_{\text{test}}$ is:

$$A^i = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{test}}} \mathbf{1}[\hat{g}^i(\mathbf{x}) = y] \ .$$

The computational costs are represented by a fixed cost vector $\mathbf{c} = (C^1, C^2, \ldots, C^M)$, where each element C^i represents the cumulative computational resources required to reach exit i.

To formally characterize early exit model performance, we must first define the concept of an inference policy.

Definition 1.4.1. Let $i \in \{1, \ldots, M\}$. An inference policy π^i is a function that makes binary exit decisions at head i based on the intermediate predictions. Formally, given an input $\mathbf{x} \in \mathcal{X}$ and intermediate prediction at exit i, the inference policy returns

$$\pi^i(\mathbf{x}) \in \{0, 1\}$$

where $\pi^i(\mathbf{x}) = 1$ indicates the decision to exit at head i, and $\pi^i(\mathbf{x}) = 0$ indicates to continue processing to the next layer.

An example of policy has already been considered in Equation (1.4). Given an inference policy π^i, we can define the accuracy and cost of the early exit model $i \in \{1, \ldots, M\}$ respectively as follows

$$\text{Accuracy}(\pi^i) = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{test}}:\ \pi^i(\mathbf{x})=1} \mathbf{1}[\hat{g}^i(\mathbf{x}) = y] \tag{1.5}$$

where $\hat{g}^i$ is the classifier at head i and

$$\text{Cost}(\pi^i) = \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{test}}:\ \pi^i(\mathbf{x})=1} C^i \ . \tag{1.6}$$

To characterize the theoretical limits of the policies $\boldsymbol{\pi} = (\pi^1, \ldots, \pi^M)$ with respect to the above accuracy and cost, we define the correctness matrix $\mathbf{M} \in \{0, 1\}^{|\mathcal{D}_{\text{test}}|\times M}$ where $\mathbf{M}_{i,j} = \mathbf{1}[\hat{g}^j(\mathbf{x}_i) = y_i]$, and the classifiable subset[2]

$$\mathcal{S}_{\text{class}} = \{(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{test}} : \exists j \in \{1, \ldots, M\}, \mathbf{M}_{i,j} = 1\}$$

of samples for which at least one head produces the correct prediction. The theoretical bounds for early exit inference define the accuracy bound and cost bound:

$$\text{Accuracy}(\boldsymbol{\pi}) \leq \frac{|\mathcal{S}_{\text{class}}|}{|\mathcal{D}_{\text{test}}|} \tag{1.7}$$

$$\sum_{j=1}^{M} \min(C^j) \leq \text{Cost}(\boldsymbol{\pi}) \leq \sum_{j=1}^{M} \max(C^j) \ . \tag{1.8}$$

The objective of early exit inference strategies is to find an optimal inference policy that approaches these theoretical bounds. Since accuracy and cost are competing objectives, the inference problem is a multi-objective optimization that seeks to maximize accuracy while minimizing cost. This optimization can be constrained by user-defined requirements on accuracy $\text{Accuracy}(\boldsymbol{\pi}) \geq \alpha_{\min}$ or cost $\text{Cost}(\boldsymbol{\pi}) \leq \beta_{\max}$, where $\alpha_{\min}$ represents the

[2]For simplification and without loss of generality, we assume that the data in $\mathcal{D}_{\text{test}}$ are numbered from 1 to $T = |\mathcal{D}_{\text{test}}|$, that is, $\mathcal{D}_{\text{test}} = \{(\mathbf{x}_i, y_i),\ i = 1 \ldots, T\}$.

minimum acceptable accuracy threshold and $\beta_{\max}$ represents the maximum allowable computational budget. These constraints reflect practical deployment requirements where certain performance levels must be guaranteed while operating within resource limitations.

The model and data constraints encompass the theoretical bounds established by the classifiable subset and the inherent limitations of the network's predictive capabilities at different depths. This multi-objective optimization with user-defined constraints makes early exit inference a challenging problem that requires sophisticated strategies to balance competing objectives while respecting both theoretical and practical limitations.

Remark 1.4.2. It is important to observe that the accuracy of an early exit model with policy $\boldsymbol{\pi}$ is not bounded by the accuracy of individual heads or the full model. With a well-designed inference policy, it is possible to achieve $\text{Accuracy}(\boldsymbol{\pi}) > \max_i A^i$, where the early exit model outperforms any single head by intelligently selecting the most appropriate exit head for each input.

1.4.1 Dynamic Inference Mechanics

During inference, the model processes an input $\mathbf{x}$ sequentially through the backbone layers, computing intermediate predictions at each exit head. At each exit i, the inference policy π^i evaluates the intermediate prediction and decides whether to exit or continue. The sequential decision process can be formalized as follows:
For each layer $i = 1, 2, \ldots, M$:

$$\text{exit at layer } i \text{ if } \pi^i(\mathbf{x}) = 1 \ . \tag{1.9}$$

The model stops computation and returns prediction $\hat{g}^i(\mathbf{x}) = \text{argmax}_k \{\psi^i(\phi^{1:e_i}(\mathbf{x}))\}$ at the first layer i where $\pi^i(\mathbf{x}) = 1$. If no exit condition is satisfied at earlier layers, the model uses the final exit by default.

The core challenge in early exit inference lies in determining when to terminate computation at each intermediate layer. This exit decision can be based on different types of information available at each exit point:

- **Softmax probabilities**: Using the normalized probability distribution $\psi^i(\phi^{1:e_i}(\mathbf{x})$ to assess prediction confidence;
- **Raw logits**: Leveraging the pre-softmax activations $W^i\phi^{1:e_i}(\mathbf{x}) + b^i$ before normalization;
- **Hidden features**: Utilizing the intermediate representations $\phi^{1:e_i}(\mathbf{x})$ directly from the backbone network before the classification head.

Each approach offers different advantages: softmax probabilities provide intuitive confidence measures, raw logits preserve the full dynamic range of the model's outputs, and hidden features capture the rich representational content learned by the backbone network at each depth.

Typical example for dynamic inference. To illustrate how the inference policy works, consider a concrete example using *confidence thresholds* based on maximum softmax probability. Let $\tau_i \in (0, 1)$ be the confidence threshold for exit i, where higher values indicate higher confidence requirements. The inference policy π^i for layer i is then defined as:

$$\pi^i(\mathbf{x}) = \begin{cases} 1 & \text{if } \max_k\{\psi^i(\phi^{1:e_i}(\mathbf{x}))\} \geq \tau_i \\ 0 & \text{otherwise.} \end{cases} \tag{1.10}$$

Consider let say a 3-exit network with thresholds $\tau_1 = 0.9$, $\tau_2 = 0.8$, $\tau_3 = 0.7$, and an input $\mathbf{x}$ that produces:

$$\begin{aligned}
&\text{Exit 1: } \max_k\{\psi^1(\phi^{1:e_1}(\mathbf{x}))\} = 0.75 \to \pi^1(\mathbf{x}) = 0 \text{ (continue)};\\
&\text{Exit 2: } \max_k\{\psi^2(\phi^{1:e_2}(\mathbf{x}))\} = 0.85 \to \pi^2(\mathbf{x}) = 1 \text{ (exit)}.\\
&\text{Exit 3: not evaluated since } \pi^2(\mathbf{x}) = 1 \text{ at exit 2.}
\end{aligned}$$

The model exits at the second layer and returns prediction $\hat{g}^2(\mathbf{x}) = \text{argmax}_k\{\psi^2(\phi^{1:e_2}(\mathbf{x}))\}$.

Computational Cost Reduction: The computational savings achieved by early exit depend on the exit distribution across layers. For all $i \in \{1, \ldots, M\}$, if $p^i \in [0, 1]$ represents the probability of exiting at layer i, the expected computational cost relative to the full model becomes:

$$\text{Expected Cost} = \sum_{i=1}^{M} p^i \cdot C^i \quad .$$

This formalization captures the dynamic nature of early exit models, where computational cost adapts to input complexity while maintaining the flexibility to utilize the full network capacity when necessary.

Remark 1.4.3. In this section, confidence is estimated using the maximum predicted probability. However, any function of the model outputs can be employed, such as the margin between the two largest probabilities or the output entropy. One must ensure that the same confidence function is consistently applied during both calibration and inference to maintain methodological coherence.

1.4.2 Exit Strategy Objectives and Approaches

Early exit networks introduce a fundamental trade-off between two competing objectives: accuracy and computational cost. This dual-objective nature necessitates the development of sophisticated inference strategies that can effectively balance these competing demands while meeting specific deployment requirements. It is important to note that computational cost is directly related to inference speed, as higher computational cost typically corresponds to slower inference times. Throughout this thesis, we make the assumption that reducing computational cost directly improves inference speed, and we treat these two metrics as interchangeable. The policy π^i given in Equation (1.10) is an example of inference policies as stated in Definition 1.4.1. It is one central tool that must be designed to optimize one or both objectives – accuracy vs. computational cost – while respecting the constraints imposed by the application domain.

In the sequel, we highlight the two types of policies that we introduced in our thesis and that will be more developed in later chapters.

Accuracy-Constrained Strategies When the primary concern is maintaining a minimum level of predictive performance, accuracy-constrained strategies focus on ensuring that the early exit model achieves a specified accuracy threshold while minimizing computational cost. This approach is particularly relevant in applications where prediction quality cannot be compromised below a certain level.

To implement accuracy-constrained strategies, we utilize a calibration dataset $\mathcal{D}_{\text{cal}}$. Formally, let $\mathcal{P}_{\text{target}}$ denote the unknown target distribution from which deployment sample are drawn. We assume that the calibration dataset $\mathcal{D}_{\text{cal}} = \{(\tilde{\mathbf{x}}_i, \tilde{y}_i)\}_{i=1}^{|\mathcal{D}_{\text{cal}}|}$ consists of samples drawn independently and identically from the same distribution:

$$(\tilde{\mathbf{x}}_i, \tilde{y}_i) \sim \mathcal{P}_{\text{target}} \quad \forall i \in \{1, \ldots, |\mathcal{D}_{\text{cal}}|\} \quad .$$

The calibration process involves computing the outputs of all classification heads for each example in $\mathcal{D}_{\text{cal}}$, providing two critical pieces of information for strategy design:

1. **Correctness indicators**: For each sample $(\tilde{\mathbf{x}}_i, \tilde{y}_i) \in \mathcal{D}_{\text{cal}}$ and head j, we define the correctness indicator $\text{correct}_{i,j} = \mathbf{1}[\hat{g}^j(\tilde{\mathbf{x}}_i) = \tilde{y}_i]$, which captures whether head j produces the correct prediction for sample i.

2. **Confidence measures**: From the intermediate outputs, we derive confidence scores $\text{conf}_{i,j} \in \mathbb{R}^+$ that quantify the reliability of predictions at each exit point. These confidence measures can be computed from one of the three different types of tensors available at exit $j \in \{1, \ldots, M\}$ for sample $\tilde{\mathbf{x}}_i$: output of the softmax, raw logits or hidden features (see 1.4.1).

In our accuracy-constrained approach, we select probability-based confidence and specifically employ the entropy of the probability distribution as our confidence metric:

$$\text{conf}_{i,j} = -\sum_{k=1}^{K} p_{i,j,k} \log p_{i,j,k} \tag{1.11}$$

where lower entropy values indicate higher confidence, as they correspond to more peaked probability distributions. This choice is motivated by entropy's direct interpretation as uncertainty quantification in probability theory. Please note that other functions can be used, such as the maximum or the breaking-ties, which correspond to the difference of the two maxima of our probability distribution.

The threshold selection process for accuracy-constrained strategies operates as follows:

Algorithm 1: Accuracy-Constrained Threshold Selection

Input: Calibration dataset $\mathcal{D}_{\text{cal}} = \{(\tilde{\mathbf{x}}_i, y_i)\}_{i=1}^{|\mathcal{D}_{\text{cal}}|}$, target accuracy $\alpha_{\min} \in [0, 1]$
Output: Threshold vector $\boldsymbol{\epsilon} = [\epsilon_1, \epsilon_2, \ldots, \epsilon_M]$ where $\epsilon_j \in \mathbb{R}^+$
for *each head* $j = 1, 2, \ldots, M$ **do**
  Compute confidence scores $\{\text{conf}_{i,j}\}_{i=1}^{|\mathcal{D}_{\text{cal}}|}$ using entropy formula (1.11);
  Compute correctness indicators $\{\text{correct}_{i,j}\}_{i=1}^{|\mathcal{D}_{\text{cal}}|}$ where $\text{correct}_{i,j} = \mathbf{1}[\hat{g}^j(\tilde{\mathbf{x}}_i) = y_i]$;
  Sort calibration samples by confidence score in ascending order (lowest entropy first);
  Find $\min(\epsilon_j)$ such that the subset $\{i : \text{conf}_{i,j} \leq \epsilon_j\}$ achieves accuracy greater or equal to $\alpha_{\min}$ according to (1.12);
end

Formally, the threshold ϵ_j for head j is defined as:

$$\epsilon_j = \inf \left\{ \tau \in \mathbb{R}^+ : \frac{\sum_{i:\text{conf}_{i,j} \leq \tau} \text{correct}_{i,j}}{\sum_{i:\text{conf}_{i,j} \leq \tau} 1} \geq \alpha_{\min} \right\} . \tag{1.12}$$

This calibration process ensures that when the inference policy exits at head j for samples with confidence below ϵ_j, the resulting accuracy meets or exceeds the specified minimum threshold $\alpha_{\min}$. The strategy thus provides accuracy guarantees while enabling early exit opportunities when confidence is sufficiently high.

Budget-Constrained Strategies In resource-constrained environments such as edge computing deployments, the primary objective shifts from maintaining accuracy to operating within strict computational budgets. Budget-constrained strategies are designed for scenarios where degraded accuracy is acceptable if it enables the system to meet

stringent speed or energy requirements. This approach is particularly relevant for real-time applications where computational resources are limited and response time is critical.

Budget-constrained strategies utilize the same calibration framework as accuracy-constrained approaches but employ a fundamentally different threshold selection mechanism. Given the calibration dataset $\mathcal{D}_{\text{cal}}$ and confidence scores $\{\text{conf}_{i,j}\}_{i=1}^{|\mathcal{D}_{\text{cal}}|}$ computed using the same confidence function (entropy of probability distributions in our case), we construct the cumulative distribution function (CDF) of the confidence measure for each head. For head $j \in \{1, \ldots, M\}$, let $\hat{F}^j : \mathbb{R}^+ \to [0, 1]$ denote the empirical CDF of confidence scores:

$$\hat{F}^j(\tau) = \frac{1}{|\mathcal{D}_{\text{cal}}|} \sum_{i=1}^{|\mathcal{D}_{\text{cal}}|} \mathbf{1}[\text{conf}_{i,j} \leq \tau] \ .$$

The budget-constrained threshold selection operates by defining thresholds in the confidence measure space such that a predetermined fraction of samples will exit at each head. Specifically, for a desired exit rate $\rho^j \in [0, 1]$ at head j, we define the threshold as:

$$\epsilon^j_{\text{budget}} = \hat{F}^{-j}(\rho^j) =: \inf\{\tau \in \mathbb{R}^+ : \hat{F}^j(\tau) \geq \rho^j\}$$

where $\hat{F}^{-j}$ is the generalized inverse of $\hat{F}^j$. This approach ensures that (under mild conditions) approximately $\rho^j\%$ of the calibration dataset will exit at head j, and these will be the samples with the highest confidence (lowest entropy) among those not already exited at earlier heads.

To achieve fine-grained budget control across all exit points, we employ an aggregation method with exponential weights that optimally distributes the computational budget across heads. This method balances the trade-off between accuracy and computational cost by solving a constrained optimization problem.

In order to simplify the presentation, let us first discuss the imposed constraint on the computational cost. Let $\bar{C} > 0$ denote the maximum allowable expected computational cost. For an exit-rate vector $\hat{\boldsymbol{\varepsilon}} = (\hat{\varepsilon}^1, \ldots, \hat{\varepsilon}^M)$, the expected cost is $\sum_{j=1}^M \hat{\varepsilon}^j \hat{C}^j$. The budget constraint is then

$$\sum_{j=1}^{M} \hat{\varepsilon}^j \, \hat{C}^j \ \leq \ \bar{C} \ .$$

The exponential weights aggregation method operates by computing optimal rejection rates (equivalently, exit rates) for each head while respecting the overall computational budget constraint. For a held-out calibration set $\mathcal{D}_{\text{cal}}$ and a head $j \in \{1, \ldots, M\}$, let $\hat{R}^j$ be the empirical risk (error rate) defined as $\hat{R}^j = \frac{1}{|\mathcal{D}_{\text{cal}}|} \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{cal}}} \mathbf{1}[\hat{g}^j(\mathbf{x}) \neq y]$ and let $\hat{C}^j$ be the empirical computational cost. Let us now specify our approach to describe the exit-rate vector $\hat{\boldsymbol{\varepsilon}}$.

Exponential Weights Aggregation Formula: The budget-constrained strategy that we consider in the present thesis – see Chapter 3 – is based on exponential weights aggregation and requires the introduction of initial prior weights that we denote $\tilde{\epsilon}^1, \ldots, \tilde{\epsilon}^M \in \Delta^{M-1}$. For all $j \in \{1, 2, \ldots, M\}$, the j-th coordinate of the rejection rates vector $\hat{\boldsymbol{\varepsilon}}$ is given by

$$\hat{\varepsilon}^j = \frac{\tilde{\epsilon}^j \exp\left\{-\frac{\hat{R}^j + \hat{\mu}\hat{C}^j}{\beta}\right\}}{\sum_{\ell=1}^M \tilde{\epsilon}^\ell \exp\left\{-\frac{\hat{R}^\ell + \hat{\mu}\hat{C}^\ell}{\beta}\right\}}$$

where $\hat{\mu} = \max\{0, \bar{\mu}\}$, with $\bar{\mu}$ being the solution of

$$\sum_{j=1}^{M} (\bar{C} - \hat{C}^j)\tilde{\epsilon}^j \exp\left\{-\frac{\hat{R}^j + \bar{\mu}\hat{C}^j}{\beta}\right\} = 0$$

and $\bar{C}$ represents the target computational budget, $\beta > 0$ is the temperature parameter controlling the smoothness of the distribution.

This aggregation method provides a principled approach to distribute the computational budget across exit points by exponentially weighting the importance of each head based on its risk-cost trade-off. Heads with lower risk and cost receive higher rejection rates, leading to more samples exiting early at these efficient points. The Lagrange multiplier $\hat{\mu}$ ensures that the overall budget constraint $\bar{C}$ is satisfied.

1.4.3 Deployment Considerations

Early exit networks offer significant potential for improving inference efficiency, but their practical deployment involves both advantages and limitations that must be carefully considered. The choice of deployment architecture fundamentally affects the computational benefits that can be realized from early exit mechanisms.

Classical Single-Device Deployment In the most straightforward deployment scenario, the entire early exit network is executed on a single computational device with limited resources. This classical setting represents the primary use case for early exit networks, where the adaptive inference complexity directly translates to improved performance metrics.

In this configuration, the early exit mechanism operates by dynamically adjusting the computational burden according to input complexity. Simple inputs that can be classified with high confidence at early layers require minimal computation, while complex inputs automatically utilize the full network capacity. This adaptive behavior enables the system to achieve better average performance compared to a fixed-depth network operating under the same computational budget.

The computational savings in single-device deployment are directly proportional to the exit distribution achieved by the inference policy. If p_j represents the probability of exiting at layer j, the expected computational cost becomes $\sum_{i=1}^{M} p_i \cdot \text{Cost}(\phi^{1:e_i})$, providing measurable speed improvements when $\sum_{j=1}^{M-1} p_j > 0$.

Edge-Server Split Deployment A more sophisticated deployment architecture involves partitioning the early exit network between edge devices and backend servers, creating a distributed inference system. In this configuration, the early layers and corresponding exit points are deployed on resource-constrained edge devices, while the deeper layers remain on powerful backend servers.

This split deployment enables a two-tier inference system with distinct computational pathways:

- **Edge classification**: Simple inputs that achieve sufficient confidence at early exit points are classified entirely on the edge device, providing immediate responses with minimal latency and no network communication overhead.
- **Server completion**: Complex inputs that require deeper processing trigger partial computation on the edge device, followed by transmission of intermediate representations to the server for completion through the remaining network layers.

The edge-server split architecture offers several advantages for scalable deployment. First, it distributes computational load between edge and cloud resources, preventing bottlenecks that would occur if all processing were centralized. Second, it enables efficient handling of multiple input streams by allowing simple classifications to be processed locally while complex cases are batched for server processing. Third, it reduces bandwidth

requirements compared to transmitting raw inputs to the server, as intermediate features are typically more compact than original data.

Formally, if L_{edge} represents the number of layers deployed on the edge device, the system achieves local classification for inputs satisfying the exit condition at any layer $j \leq L_{\text{edge}}$, while inputs requiring layers $j > L_{\text{edge}}$ incur additional network latency and server processing time.

Limitations and Challenges Despite their computational advantages, early exit networks introduce several practical limitations that affect deployment efficiency, particularly in production environments where batched processing is commonly employed.

Batching Inefficiency: One of the most significant limitations of early exit networks concerns their interaction with batched inference, which is a cornerstone of efficient deep learning deployment. In traditional neural networks, batched processing enables efficient utilization of parallel computational resources by processing multiple inputs simultaneously through the same network layers.

Early exit networks fundamentally challenge this batching paradigm. When processing a batch of inputs, the network must continue computation until all samples in the batch have reached their respective exit points. In practice, this typically means that most batches require computation through the entire network, as at least one sample usually needs the full network depth. Consequently, the computational savings from early exit are significantly diminished in batched scenarios.

Mathematically, if a batch contains B samples with exit points $\{j_1, j_2, \ldots, j_B\}$, the actual computation required is $B \times \hat{C}^{\max\{j_1, j_2, \ldots, j_B\}}$, rather than $\sum_{i=1}^{B} \hat{C}^{j_i}$ that would be achieved with independent processing.

Dynamic Batching Solutions: To mitigate the batching inefficiency, one can implement dynamic batching strategies where inference proceeds in stages. In this approach, the system only processes a layer when the batch for that layer is sufficiently filled, allowing samples that exit early to be immediately returned while continuing computation for the remaining samples.

However, dynamic batching introduces its own complexities and trade-offs. First, it may delay the output of some samples as the system waits to accumulate sufficient samples for efficient batch processing at deeper layers. Second, it significantly increases system implementation complexity, requiring sophisticated scheduling and memory management to handle multiple concurrent batches at different network depths. Third, it may introduce variable latency patterns that are difficult to predict and optimize for real-time applications.

These limitations highlight the importance of carefully considering the deployment context when designing early exit systems, as the theoretical computational savings may not fully translate to practical performance improvements in all scenarios.

1.5 Contributions and Organization

This manuscript approaches early-exit neural networks through a statistical lens, treating the learning of auxiliary heads and the design of inference policies as two facets of the same adaptive system. The analysis keeps the focus on guarantees that remain valid under mild assumptions; in particular, Chapter 3 develops theoretical results that hold without resorting to restrictive hypotheses.

1.5.1 Statistical Perspective on Early-Exit Learning

The learning problem is revisited as a multi-objective optimisation task where several heads share a common backbone. The thesis examines how loss design, hierarchical distillation, and calibration-aware training jointly control the excess risk incurred by intermediate predictors. Special attention is paid to gradient conflicts and to the practical "plug-and-play" training of heads on frozen backbones, a setting especially relevant for large pre-trained models.

1.5.2 Statistical Perspective on Early-Exit Inference

Inference is framed as a sequential decision process. Confidence scores, reject-option rules, and computational budgets are combined to ensure that each exit decision balances expected risk against resource consumption. This perspective connects early exiting with reject option, emphasising calibrated decisions that can be certified empirically.

1.5.3 Contribution 1: Budget-Constrained Inference with EERO (Chapter 3)

Chapter 3 introduces the EERO framework, in which early exit is cast as classification with reject option. The method couples aggregation with exponential weights and data-driven calibration to deliver explicit control on computational costs while bounding the excess risk of predictions. The guarantees rely on mild assumptions and extend to standard computer-vision architectures such as ConvNeXt and MSDNet.

1.5.4 Contribution 2: Accuracy-Preserving Early Exits for LLMs (Chapter 4)

Chapter 4 studies early exits for autoregressive language models. It proposes self-supervised objectives that allow randomly initialized heads to inherit the behavior of the full model while retaining calibrated uncertainty. Entropy-based calibration leads to confidence thresholds that sustain baseline accuracy across established NLP benchmarks, delivering measurable speedups that grow with model size.

1.5.5 Broader Efficiency Ecosystem and System Insights (Chapter 2)

The literature review in Chapter 2 situates early exits among complementary efficiency techniques such as pruning, quantisation, mixture-of-experts, and speculative decoding. It also highlights system-level constraints—batching, memory footprint, and hardware support—that ground the statistical objectives pursued later in the thesis.

1.5.6 Summary of Contributions

This thesis is based on the following publications, which present the primary technical and theoretical contributions:

1. **Florian Valade**, Mohamed Hebiri, and Paul Gay. *EERO: Early Exit with Reject Option for Efficient Classification with limited budget.*

2. **Florian Valade**. *Accelerating Large Language Model Inference with Self-Supervised Early Exits.*

The specific contributions of each chapter are summarized below:

- Chapter 2 consolidates prior work and isolates the open problems addressed thereafter.

- Chapter 3 presents EERO, its theoretical analysis, and experiments under budget constraints, based on the research *EERO: Early Exit with Reject Option for Efficient Classification with limited budget.*

- Chapter 4 develops and evaluates accuracy-preserving early exits for large language models, building upon the work *Accelerating Large Language Model Inference with Self-Supervised Early Exits.*

- The conclusion synthesizes the findings, reports ongoing work on belief-state policies and speculative decoding, and outlines future research directions.

Together, these chapters show that it is possible to reconcile statistical guarantees and practical deployment needs for both the learning and inference stages of early-exit neural networks.

Chapter 2

Literature Review

Early exit methods represent a paradigm shift in neural network design, enabling **dynamic adaptation of network depth based on input difficulty** rather than processing all inputs through fixed computational paths. This capability addresses fundamental inefficiencies in traditional deep networks, where simple inputs unnecessarily consume the same computational resources as complex ones. The field has evolved from basic proof-of-concept approaches to sophisticated systems that achieve substantial efficiency gains while maintaining competitive accuracy across diverse domains.

2.1 Early Exit Architectures

Early exit architectures have evolved differently across domains, with distinct challenges and solutions emerging for computer vision and natural language processing applications. Vision tasks benefit from spatial locality and hierarchical feature extraction, where early exits can exploit coarse-to-fine representations at different network depths. In contrast, NLP tasks involve sequential dependencies and large vocabularies, requiring different approaches to intermediate prediction and confidence estimation. Both domains share fundamental architectural principles but diverge in their specific implementations and optimization strategies.

This section examines the architectural evolution in both domains, beginning with the foundational developments in computer vision that established core early exit principles, followed by the adaptation and extension of these concepts to natural language processing challenges.

2.1.1 Computer Vision Architectures

The evolution of computer vision architectures from foundational convolutional neural networks to modern deep learning systems provided the architectural foundation for early exit methods. The progression from LeNet [73] through AlexNet [68], which demonstrated the power of deep CNNs on large-scale datasets, to VGG [120] that showed the importance of network depth, and ResNet [50] that solved the vanishing gradient problem through residual connections, established the core CNN-based architectural principles. The subsequent introduction of Vision Transformers (ViT) [31] marked a paradigmatic shift, demonstrating that transformer architectures could achieve superior performance on image classification by treating image patches as sequences, effectively bridging computer vision and natural language processing approaches. This breakthrough spawned influential variants [130, 88, 140] that addressed data efficiency, computational complexity, and dense prediction tasks. These foundational architectures demonstrated that deeper networks could learn increasingly sophisticated hierarchical features, setting the stage for early exit

approaches that leverage these intermediate representations for adaptive computation across both CNN and transformer-based vision models.

Auxiliary classifiers: Training-only intermediate supervision

As mentioned in the introduction with **LeNet**, early intermediate heads were initially used to stabilize training rather than to enable adaptive inference. **GoogLeNet (2015)** [125] extended this idea by adding auxiliary classifiers at intermediate depths (after Inception modules 4a and 4d) to mitigate vanishing gradients in deep networks.

These auxiliary heads contributed additional loss terms during training but were **discarded at inference time**, serving purely as a mechanism to improve gradient flow. While the accuracy gains were minor, this approach established the architectural template of multi-exit networks later repurposed for dynamic inference in true early-exit methods.

BranchyNet: The foundational architecture

BranchyNet (2016) [128] established the foundational paradigm for early exit networks by introducing the concept of "side branch classifiers" attached to intermediate layers of existing networks. This seminal work demonstrated early exit feasibility on established CNN architectures including LeNet [73], AlexNet [68], and ResNet [50], showing that the hierarchical feature learning principles developed in these foundational architectures could be exploited for adaptive computation. The approach established the basic architectural template that underlies all subsequent developments.

The core architectural innovation consists of lightweight classification heads ψ^i attached at predetermined network locations, where each head processes intermediate representations $\phi^{1:i}(\mathbf{x})$ to produce predictions:

$$f_{\text{exit}}^{(i)}(\mathbf{x}) = \psi^i(\phi^{1:i}(x; \boldsymbol{\theta}_\phi); \theta_\psi^i) \ . \tag{2.1}$$

BranchyNet employed entropy-based exit policies where prediction entropy served as a confidence measure for early termination. However, the approach revealed **fundamental limitations** that would drive subsequent architectural innovations. Internal representation alteration emerged as the primary concern—early exit classifiers modified intermediate features, potentially degrading performance of subsequent layers. Additionally, early layers inherently lacked coarse-scale features necessary for robust classification, and predictions from non-exiting classifiers were simply discarded, representing wasted computation.

These limitations highlighted critical challenges: **feature interference** between early exits and deeper layers, **inadequate early representations** for classification tasks, and **computational waste** from unused predictions.

Dense connectivity solutions: MSDNet

Multi-Scale Dense Networks (MSDNet, 2017) [57] directly addressed BranchyNet's architectural limitations through two critical innovations that fundamentally changed early exit design philosophy.

Dense connectivity ensures every layer connects to all subsequent classifiers, preventing exit-induced feature degradation while enabling maximal computation reuse. This architectural change maintains feature quality throughout the network and eliminates the interference problem observed in BranchyNet.

Multi-scale architecture maintains features at multiple spatial resolutions throughout the network, providing discriminative coarse features early in the processing pipeline. This innovation directly addresses the poor early accuracy problem by ensuring that classification-relevant features are available at all exit points.

This combination achieved similar accuracy to 110-layer ResNet using only 1/10 computational budget, demonstrating up to 5× efficiency improvement over traditional architectures while maintaining the early exit paradigm.

Modern architectural sophistication

Building upon the dense connectivity foundations, recent developments have introduced increasingly sophisticated architectural approaches that push the boundaries of early exit design.

Zero Time Waste (ZTW) architectures (2023) [108] eliminate computation redundancy through direct connections between internal classifiers with ensemble-like prediction combination. Unlike MSDNet's dense connectivity, ZTW architectures specifically optimize for computational efficiency, achieving up to 5 percentage points improvement on low computational budgets through intelligent feature reuse.

Class-specific exit design (2021) [113] represents a departure from universal architectural approaches by targeting early exit placement for specific classes. This innovation recognizes that different classes may require different computational complexity, leading to specialized architectural patterns that optimize for class-dependent exit strategies.

Hierarchical exit strategies (2023) [83] with Bag-of-Features representations provide compact, discriminative representations that preserve spatial information and feature distributions. These approaches move beyond simple linear classification heads to sophisticated representation learning specifically designed for intermediate exit points.

Vision architectural design principles

The evolution from auxiliary classifiers (2015) through BranchyNet (2016) to modern architectures (2017-2023) has revealed fundamental **design principles** that distinguish effective early exit systems:

Training versus inference objectives: The paradigm shift from GoogLeNet's training-only auxiliary heads to BranchyNet's inference-time exits established the fundamental distinction between intermediate supervision for gradient flow versus adaptive computation for efficiency. Modern architectures maintain this dual consideration, ensuring that early exit heads serve both training stability and dynamic inference.

Progressive feature complexity: Unlike BranchyNet's uniform approach, modern architectures recognize that early exits require specialized feature representations that balance computational efficiency with discriminative power. This principle emerged from understanding that different network depths provide features of varying abstraction levels.

Complexity-decreasing branch patterns: The field has identified that more complex branches appearing earlier in the backbone network provide superior accuracy-cost trade-offs by maintaining knowledge consistency with backbone features and reducing disruption to the feature abstraction hierarchy—a finding that fundamentally challenges intuitive approaches to branch placement.

Feature preservation: Dense connectivity patterns, pioneered by MSDNet (2017) and refined in subsequent architectures, ensure that early exit decisions do not degrade the feature learning capabilities of deeper layers. This principle directly addresses the interference problems first observed in BranchyNet, enabling systems to maintain both early exit accuracy and final layer performance.

Architectural modularity: Modern approaches emphasize modular designs that can be added to existing pre-trained networks without requiring complete retraining, building on the conceptual foundation established by GoogLeNet's auxiliary structure while enabling practical deployment efficiency.

2.1.2 Natural Language Processing Architectures

Natural language processing early exit architectures face fundamentally different challenges compared to their computer vision counterparts due to sequential processing requirements and contextual dependencies. The evolution of NLP architectures from attention mechanisms [2] to the transformer architecture [133] and subsequently to models like BERT [29] and T5 [101] established the foundation for modern language understanding systems. Unlike vision networks that process spatial hierarchies with local pattern recognition, NLP networks handle sequential dependencies requiring global context preservation, fundamentally impacting architectural design decisions for early exit implementations.

Foundational transformer approaches

DeeBERT (2020) [146] established the foundation for early exit in transformer architectures by placing simple linear classifiers after each transformer layer, achieving up to 40% inference speedup through entropy-based thresholding. The approach demonstrated that transformer layers naturally develop increasingly sophisticated representations suitable for intermediate classification, but revealed **limitations** in handling attention mechanism complexities and global context requirements that distinguish NLP from vision tasks.

PABEE (2020) [157] introduced patience mechanisms where exits occur when predictions remain stable across multiple consecutive layers, providing more robust exit decisions than confidence-only approaches. This innovation addressed DeeBERT's overreliance on single-layer confidence by requiring consistency across layer sequences, better accommodating the contextual dependencies inherent in language processing.

FastBERT (2020) [87] employed self-distillation frameworks where early exit heads learn from the final model's predictions, similar to vision approaches but adapted for transformer architectures. However, **limitations** emerged in maintaining attention weight optimization when exits bypass subsequent transformer blocks, disrupting the global context mechanisms essential for language understanding.

Advanced transformer innovations

BERxiT (2021) [147] improved early exiting through alternating fine-tuning strategies and extended methods to regression tasks with Learning-To-Exit components. The approach addressed training instability issues observed in earlier methods by carefully orchestrating the training of exit heads with the main transformer backbone.

CALM (2022) [114] applied early exit concepts to T5 [101] encoder-decoder architectures, demonstrating confident adaptive language modeling for generation tasks. This marked a significant extension beyond BERT-style encoders to sequence-to-sequence models, revealing new challenges in maintaining generation coherence when computational paths vary across tokens.

Layer Skip (2024) [84] represents recent advances in self-speculative decoding, building upon foundational speculative decoding work [75, 13] that established the paradigm of using smaller draft models to accelerate larger target models. Unlike traditional speculative approaches that require separate draft and target models, Layer Skip enables early layers of the same model to generate draft predictions verified by remaining layers. This approach reduces memory footprint compared to traditional speculative decoding while benefiting from shared compute and activations, achieving up to 2.16× speedup on summarization tasks.

NLP architectural design principles

The evolution of NLP early exit architectures has revealed domain-specific **design principles** that distinguish them from vision approaches:

Sequential dependency preservation: Unlike vision networks that can accept spatial detail reduction, NLP tasks critically depend on sequential context. Exit placement must carefully consider attention weight optimization and global context requirements, typically positioning exits after complete transformer blocks rather than within attention mechanisms.

Vocabulary and scaling challenges: Large vocabulary sizes (>50K tokens) create computational bottlenecks in confidence estimation that do not exist in vision classification. Modern approaches must address the expense of softmax computation over extensive vocabularies, leading to specialized confidence metrics and calibration strategies.

Token-level versus sequence-level exits: NLP architectures must handle both token-level decisions (for generation) and sequence-level decisions (for classification), requiring more sophisticated exit policies than vision networks that typically operate at the sample level.

Attention mechanism compatibility: Early exits must maintain compatibility with self-attention and cross-attention mechanisms that depend on global context. This constraint limits exit placement options compared to vision networks and requires careful consideration of how exits affect subsequent attention computations.

Generation coherence: For sequence-to-sequence tasks, early exits must maintain generation coherence across varying computational paths, a challenge absent in vision classification tasks that produce single predictions per sample.

Modern generation acceleration: Layer Skip [84] represents the current state-of-the-art in early exit approaches for generation tasks, demonstrating that early hidden layers can effectively serve as draft generators for subsequent layers. This self-speculative approach achieves substantial generation speed improvements (up to 2.16× speedup) by leveraging the natural hierarchical processing in transformer architectures, where early layers generate draft predictions that are verified and refined by remaining layers, effectively combining the benefits of early exit with speculative decoding paradigms.

2.2 Training Methodologies for Early Exit Networks

Training early exit networks presents unique challenges that fundamentally distinguish them from traditional single-objective optimization problems. The introduction of multiple intermediate classifiers transforms the learning process into a complex multi-objective optimization task where competing objectives must be carefully balanced to achieve optimal performance across all exit points. This section examines the evolution of training methodologies from basic multi-exit loss formulations to sophisticated approaches that address gradient conflicts, leverage hierarchical supervision, and ensure reliable confidence estimation.

2.2.1 Multi-Exit Loss Functions: Balancing Competing Objectives

Training early exit networks fundamentally transforms single-objective optimization into a complex multi-objective problem where each intermediate classifier contributes its own loss term. The combined objective $\mathcal{L}_{\text{total}} = \sum_{i=1}^{M} w_i \mathcal{L}_i$ requires careful weight selection to balance competing goals across multiple exits, where w_i represents the importance weight for exit i and $\mathcal{L}_i$ denotes the loss at that exit point.

Static weighting approaches provided initial solutions but revealed significant limitations in practice. Linear weighting with predetermined constants offers implementation simplicity but requires extensive hyperparameter tuning that often fails to generalize across different architectures and datasets [128]. Depth-based weighting using coefficients inversely proportional to depth $(1/(M-i+1))$ addresses gradient imbalance by giving smaller coefficients to earlier layers, preventing vanishing gradients but potentially underemphasizing early exit performance [57].

Dynamic weighting strategies emerged as sophisticated solutions to multi-objective optimization challenges. The gradient equilibrium method dynamically adjusts loss weights to balance gradient magnitudes across exits, monitoring gradient alignment and scaling weights to prevent any single exit from dominating optimization [69]. Uncertainty-based weighting learns noise parameters σ_i for each exit, weighting losses by $1/(2\sigma_i^2)$ to automatically balance multiple loss functions based on predictive uncertainty [66].

Surprisingly, **Random Loss Weighting (RLW)** research demonstrated that sampling weights randomly from distributions can achieve comparable performance to sophisticated weighting schemes [76]. This finding suggests that avoiding local minima through stochasticity may be more important than precise weight optimization, challenging assumptions about the necessity of complex weighting strategies.

Harmonized Dense Knowledge Distillation (HDKD) represents the state-of-the-art approach, combining multi-exit classification losses with dense distillation losses through bilevel optimization [143]. The method updates loss weighting parameters based on validation performance while optimizing network parameters using weighted multi-objective functions, encouraging each exit to learn from all subsequent exits.

Formally, an early exit model f_{exit} with M heads provides the collection of classifiers

$$f_{\text{exit}}(\mathbf{x}) = \{\psi^i(\phi^{1:e_i}(\mathbf{x}))\}_{i=1}^M \ ,$$

where ψ^i denotes the i-th intermediate head applied to the partial backbone $\phi^{1:e_i}$. The classification loss at exit i is the cross-entropy:

$$\mathcal{L}_{\text{CE}}^{(i)}(\mathbf{x}, y) = -\sum_{c=1}^{K} \tilde{y}_c \log \left[\psi^i(\phi^{1:e_i}(\mathbf{x}))\right]_c \ , \tag{2.2}$$

where $\tilde{y}$ is the one-hot encoded ground-truth label. The distillation loss from a later exit i to an earlier exit j $(i > j)$ is defined as:

$$\mathcal{L}_{i\to j}^{KD}(\mathbf{x}, y) = -T^2 \sum_{c=1}^{K} s_c^j \log s_c^i,$$

with, for all $k \in \{1, \ldots, M\}$, we have $s_c^k = \text{softmax}\left(\frac{z^{(k)}(\mathbf{x})}{T}\right)_c$ is the softened probability of class c at head k, $z^{(k)}(\mathbf{x})$ is the corresponding logits, and $T > 0$ is the temperature.

The overall HDKD training objective is:

$$\theta^*(\boldsymbol{\gamma}) = \text{argmin}_{\boldsymbol{\theta}}\, \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}_{\text{train}}} \left[\sum_{i=1}^{M} \gamma^i\, \mathcal{L}_{\text{CE}}^{(i)}(\mathbf{x}, y) + \sum_{i=2}^{M} \sum_{j=1}^{i-1} \gamma^{ij}\, \mathcal{L}_{i\to j}^{KD}(\mathbf{x}, y) \right] \ ,$$

where $\gamma^i \geq 0$ controls the weight of the cross-entropy at exit i, and $\gamma^{ji} \geq 0$ is the weight of the distillation loss from exit i to exit j. The notation $\boldsymbol{\gamma}$ stands for the tensor $\left((\gamma^i)_{i=1}^M, (\gamma^{i,j})_{i,j=1}^M\right) \in \mathbb{R}^M \times \mathbb{R}^{M\times M}$. These weights $\boldsymbol{\gamma}$ are optimized dynamically in a bilevel scheme. At the upper level:

$$\boldsymbol{\gamma}^* = \text{argmin}_{\boldsymbol{\gamma}}\, \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}_{\text{val}}} \left[\mathcal{L}_{\text{CE}}(\mathbf{x}, y) \right] \ ,$$

while at the lower level, network parameters $\boldsymbol{\theta}$ are updated with the weighted training loss.

2.2.2 Training Strategies: Joint versus Progressive Optimization Paradigms

The fundamental question of how to train multi-exit architectures has evolved from simple two-stage approaches to sophisticated mixed-regime optimization that combines the benefits of both paradigms [69].

Joint (end-to-end) training optimizes the entire multi-exit network simultaneously from scratch, offering unified optimization with natural co-adaptation between backbone and exit heads [128, 57]. However, this approach suffers from gradient conflicts between competing objectives, risks suboptimal backbone pre-training, and often experiences convergence stability issues [70].

Disjoint (progressive) training separates optimization into distinct phases: first training the backbone network to convergence using only the final exit, then freezing backbone parameters while training internal classifiers [69]. This approach ensures high-quality backbone representations and eliminates gradient conflicts during backbone training, providing more predictable training dynamics. However, the increased computational overhead and potential suboptimality due to frozen backbone parameters limit its effectiveness.

Mixed regime training has emerged as the optimal approach through extensive empirical analysis [69, 57]. This strategy combines backbone pre-training until convergence with subsequent joint training of the entire network until convergence. Research demonstrates that mixed regime training achieves superior performance at medium and high computational budgets, while mode connectivity analysis shows mixed and disjoint regimes produce similar solution landscapes [69]. Recent comprehensive analysis [69] provides systematic evaluation of training strategy impacts, confirming the superiority of mixed regime approaches across diverse architectures and datasets while providing practical guidelines for training strategy selection.

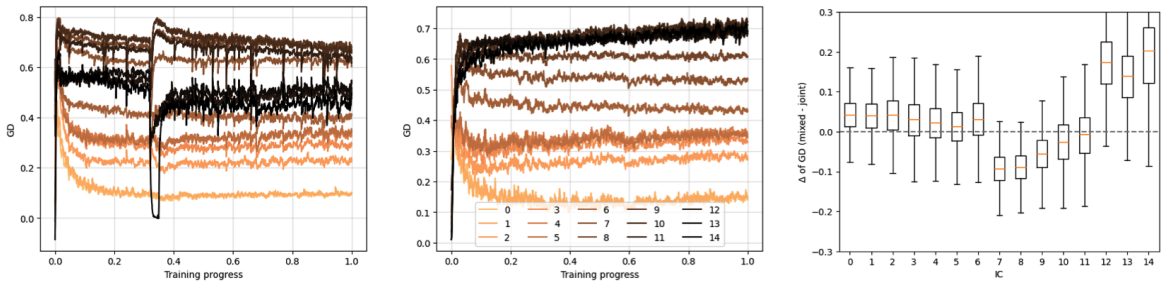
2.2.3 Knowledge Distillation: Leveraging Hierarchical Supervision

Early exit networks provide a natural framework for knowledge distillation, where later, more accurate exits serve as teachers for earlier exits with limited representational capacity. This hierarchical supervision enables sophisticated knowledge transfer strategies that significantly improve early exit performance.

Dense knowledge distillation extends traditional teacher-student paradigms by leveraging all possible supervision—each exit learns from all subsequent exits, creating a rich knowledge transfer network [143]. Unlike traditional approaches using only final layers as teachers, dense knowledge distillation (KD) incorporates distillation losses between all exit pairs, maximizing knowledge utilization across the architecture.

Self-distillation approaches demonstrate sophisticated knowledge transfer mechanisms. The FastBERT framework [87] employs self-distillation where early exit heads learn from the final model's predictions, adapting traditional distillation concepts for transformer architectures. However, limitations emerged in maintaining attention weight optimization when exits bypass subsequent transformer blocks, disrupting the global context mechanisms essential for language understanding.

One-for-All Knowledge Distillation uses single teacher models to improve all intermediate exits simultaneously, reducing computational complexity compared to training separate teachers for each exit while maintaining knowledge transfer effectiveness. This approach addresses the scalability challenges inherent in dense distillation methods for networks with many exit points.



(a) Joint regime (b) Mixed regime (c) Avg. difference between joint and mixed

Figure 2.1: Gradient dominance across exits for joint vs. mixed training. Deeper exits dominate under the mixed regime [69].

2.2.4 Gradient Flow Issues: Managing Optimization Conflicts

Training early-exit architectures involves competing gradient signals from multiple heads, which can conflict and slow convergence. Kubaty et al. [69] analyze these *gradient conflicts* and introduce a gradient-dominance metric to quantify alignment between exit-specific gradients and the overall update.

Gradient dominance. Let the objective be a weighted sum

$$\mathcal{L}_{\text{multi-exit}}(\mathbf{x}, y) = \sum_{i=1}^{M} \alpha^i \, \mathcal{L}_{\text{CE}}^{(i)}(\mathbf{x}, y) \ ,$$

where $\alpha^i \geq 0$ is the coefficient of exit i and $\mathcal{L}_{\text{CE}}$ denotes the cross-entropy (2.2). Denoting backbone parameters by $\boldsymbol{\theta}_\phi$ and head-i parameters by θ_ψ^i, the gradient of the backbone induced by exit i is

$$\mathbf{g}^{(i)}(\mathbf{x}, y) = \frac{\partial \mathcal{L}_{\text{CE}}^{(i)}(\mathbf{x}, y)}{\partial \boldsymbol{\theta}_\phi}, \qquad \mathbf{g}_{\text{tot}}(\mathbf{x}, y) = \sum_{j=1}^{M} \alpha^j \, \mathbf{g}^{(j)}(\mathbf{x}, y) \ .$$

The gradient dominance of exit i is then

$$\text{GD}_i(\mathbf{x}, y) \ = \ \frac{\langle \mathbf{g}^{(i)}(\mathbf{x}, y), \mathbf{g}_{\text{tot}}(\mathbf{x}, y) \rangle}{\|\mathbf{g}^{(i)}(\mathbf{x}, y)\| \, \|\mathbf{g}_{\text{tot}}(\mathbf{x}, y)\|} \ ,$$

where $\langle \cdot, \cdot \rangle$ and $\|\cdot\|$ denote the Euclidean inner product and norm respectively.

Findings. Kubaty et al. [69] show that in the mixed regime, later exits dominate the backbone updates (higher GD_i for large i), whereas in joint training the gradient contributions are more conflicting. Depth-dependent coefficients α^i or per-exit gradient scaling can mitigate conflicts but are less critical under mixed training.

2.2.5 Self-Supervised Approaches: Eliminating Task-Specific Requirements

DAISY (Data Adaptive Self-Supervised Early Exit) represents a breakthrough in eliminating task-specific training requirements through a three-stage process: attaching early exit branches and optimizing with self-supervised objectives, training downstream classifiers using weighted features from frozen branches, and freezing all models for inference [85].

This approach enables single training phases for early exit branches across all downstream tasks, eliminates fine-tuning requirements for self-supervised models, and bases exit decisions on data properties rather than task-specific requirements. **Confidence-based self-supervision** trains early exit heads using the model's own predictions as training data, while **class-exclusion based training** uses learned features to progressively eliminate irrelevant classes rather than making binary exit decisions.

2.2.6 Calibration During Training: Ensuring Reliable Confidence Estimates

Proper calibration ensures that confidence scores accurately reflect prediction correctness, which becomes critical for early exit networks where exit decisions depend on confidence thresholds. **Training-time calibration methods** integrate a *temperature parameter* into the softmax function during training rather than applying post-hoc calibration. Specifically, given the logits $\mathbf{z}_i$ for class i, the calibrated probability is defined as $p_i = \frac{\exp(\mathbf{z}_i/T)}{\sum_j \exp(\mathbf{z}_j/T)}$, where $T > 0$ is the temperature. When $T = 1$, this reduces to the standard softmax. Increasing $T > 1$ produces a smoother probability distribution (higher entropy), mitigating overconfidence, while decreasing $T < 1$ sharpens the distribution, amplifying confidence in the predicted class. By tuning T during training, the network learns probability estimates that are better aligned with true accuracy, thus improving the reliability of exit decisions in early-exit architectures [44].

Confidence weight methods explicitly penalize confident incorrect predictions during training by weighting losses based on prediction confidence, while **hierarchical differentiable mechanisms** include early exit mechanisms directly in training objectives to avoid train-test mismatch. Calibration-aware architectures incorporate calibration constraints during architecture design, using Expected Calibration Error (ECE) as design criteria to ensure reliable confidence estimation across all exit points.

2.2.7 Critical Analysis: Training Paradigm Effectiveness

Research reveals that **mixed regime training** can outperform alternatives for medium to high computational budgets, while joint training may suffice for low computational budgets and disjoint training provides maximum stability for critical applications [69]. **Loss weighting strategy** selection should begin with simple linear weighting for initial implementations, progress to dynamic weighting based on gradient equilibrium for advanced applications, and consider random loss weighting for research applications due to its surprising effectiveness [76].

Dense knowledge distillation should be considered essential for early exit networks, as it consistently improves performance without architectural modifications [143]. The additional computational cost during training is justified by significant inference-time benefits. **Training-time calibration methods** are preferable to post-hoc approaches for early exit networks due to multi-objective optimization requirements and the critical importance of reliable confidence estimates for exit decisions [44].

2.3 Inference Mechanisms and Adaptive Decision Policies

The transition from trained early exit networks to practical deployment requires sophisticated inference mechanisms that determine when to exit based on dynamic confidence assessments. This section examines the evolution from simple heuristic-based policies

to principled frameworks with formal guarantees, addressing the critical challenge of reliable confidence estimation under distribution shifts and the development of optimal decision-making strategies for adaptive computation.

2.3.1 Confidence Estimation and Uncertainty Quantification

The fundamental premise of early exit networks relies on the model's ability to accurately assess its own prediction confidence. However, modern deep neural networks suffer from severe miscalibration, where softmax outputs do not represent true posterior probabilities [44]. This overconfidence problem becomes catastrophic for early exit decisions, where **fake confidence** high probability scores assigned to incorrect predictions can trigger premature exits that lock in errors deeper layers might have corrected [65].

Confidence proxy mechanisms

Softmax-based metrics represent the most widely adopted confidence measures due to their computational simplicity and natural availability from classification heads. Maximum Softmax Probability (MSP) treats the highest probability in the output distribution as a confidence indicator, while entropy-based measures $H(p) = -\sum_{i=1}^{K} p_i \log p_i$ quantify prediction uncertainty through distribution peakedness [128]. However, these approaches suffer from fundamental limitations under distribution shifts, where calibration degrades severely.

Consistency-based confidence addresses single-prediction noise through temporal stability requirements. The patience mechanism introduced by Zhou et al. [157] requires predictions to remain stable across multiple consecutive exits before allowing termination, preventing spurious high-confidence predictions from triggering premature exits. This approach demonstrates particular effectiveness in transformer architectures where sequential processing naturally provides multiple decision points.

Bayesian uncertainty estimation provides more principled approaches through model uncertainty quantification. Monte Carlo Dropout [38] performs multiple forward passes with dropout enabled during inference, generating prediction distributions whose variance indicates epistemic uncertainty. While computationally more expensive, this approach provides richer uncertainty information for exit decisions.

Calibration challenges and solutions

The reliability of confidence-based exiting fundamentally depends on model calibration the agreement between predicted confidence and actual accuracy. Expected Calibration Error (ECE) quantifies miscalibration by partitioning predictions into confidence bins and measuring the absolute difference between average confidence and actual accuracy within each bin [44].

Distribution shift impacts create severe calibration failures that undermine early exit effectiveness. Under domain shifts, models exhibit two detrimental failure modes: **underthinking**, where corrupted samples receive high confidence scores leading to early incorrect exits, and **overthinking**, where correctly classified samples receive low confidence scores, preventing beneficial early exits and potentially causing errors at deeper layers [111].

Training-time calibration methods integrate calibration objectives directly into the training process. Temperature scaling learns per-exit temperature parameters during training: $p_i = \frac{\exp(\mathbf{z}_i/T)}{\sum_j \exp(\mathbf{z}_j/T)}$, while Platt scaling learns sigmoid transformations of logits to improve calibration [44]. These approaches prove more effective than post-hoc calibration for multi-exit settings due to the complex multi-objective optimization requirements.

2.3.2 Decision Policy Frameworks

The inference policy $\pi^i : \mathcal{X} \to \{0, 1\}$ determines exit decisions based on confidence estimates and system constraints. Policy design spans from simple heuristics to sophisticated learned mechanisms that optimize accuracy-efficiency trade-offs.

Heuristic-based policies

Static thresholding represents the foundational approach where exit decisions compare confidence scores against predetermined thresholds τ_i at each exit i. Despite implementation simplicity, static policies suffer from brittleness to distribution shifts and inability to adapt to runtime conditions [128]. Threshold optimization typically employs validation set sweeps to achieve desired accuracy-efficiency trade-offs.

Patience-based mechanisms enhance robustness by requiring prediction stability across multiple exits. The "BERT Loses Patience" strategy [157] waits for consistent predictions over k consecutive layers before exiting, addressing the noisy nature of single-layer confidence estimates in transformer architectures. This temporal consistency requirement proves particularly effective for sequential processing tasks where context accumulation occurs across layers.

Learned decision policies

Supervised gating networks treat exit decisions as classification problems, training dedicated models to predict the optimal exit point for each input. For instance, EENet learns a scheduler network that selects the earliest exit satisfying accuracy-efficiency constraints [59], while Han et al. [48] propose a sample-weighting mechanism that guides intermediate exits toward “easy” samples and later exits toward “hard” ones. Jointly-learned frameworks such as JEI-DNN further reduce train-test mismatch by optimizing gating and intermediate classifiers simultaneously [104].

Reinforcement learning approaches model early exit decisions as sequential decision processes where agents choose between `exit` and `continue` actions at each layer [70]. The state represents current hidden representations, while reward functions encode accuracy-efficiency trade-offs through positive rewards for correct early exits and negative costs for continued computation. Policy gradient methods like REINFORCE optimize expected cumulative rewards, enabling more sophisticated decision strategies than fixed heuristics.

Multi-armed bandit frameworks address dynamic environments where optimal trade-offs vary over time due to changing system conditions [98]. Each exit layer represents an arm, and bandit algorithms learn online to select optimal exit strategies based on observed rewards reflecting current objectives. This approach provides adaptation to runtime conditions while maintaining theoretical guarantees on cumulative performance.

2.3.3 Theoretical Foundations and Optimality

Early exit inference policies build upon established theoretical frameworks that provide principled approaches to accuracy-efficiency optimization.

Classification with reject option

The historical foundation traces to Chow’s optimal reject rule [19], which derives Bayes-optimal decisions for classifiers with rejection options. The rule prescribes prediction when conditional risk falls below rejection cost and abstention otherwise.

Formally, let $(\mathbf{X}, Y)$ be a sample data drawn from some distribution P on the product space $\mathcal{X} \times \mathcal{Y}$. The (population) risk of some classifier $g : \mathcal{X} \to \mathcal{Y}$ is defined as

$$R(g) = \mathbb{E}_P \left[\ell(Y \neq g(\mathbf{X})) \right] \quad ,$$

where ℓ is some loss function (typically $\ell(c, y) = \mathbb{1}_{\{c \neq y\}}$) and $\mathbb{E}_P$ is the expectation with respect to $\mathcal{P}$. Chow's rule is then

$$\delta(\mathbf{x}) = \begin{cases} \arg\max_{c \in \mathcal{Y}} P(Y = c \mid X = \mathbf{x}), & \text{if } 1 - \max_c P(Y = c \mid X = \mathbf{x}) \leq \lambda, \\ \text{reject}, & \text{otherwise}, \end{cases}$$

where $\lambda > 0$ is the rejection cost. Early exit networks implement cascaded reject classifiers where "rejection" at one layer passes samples to subsequent, more capable classifiers.

This statistical framework of rejection was popularized in the machine learning literature under the name *selective classification*, where risk represents error rate on non-abstained samples and coverage measures the fraction of samples receiving predictions [32, 40]. The Risk-Coverage curve characterizes selective classifier performance, with superior methods achieving lower risk at higher coverage levels.

Furthermore, the calibration of the rejection mechanism can be enhanced through semi-supervised techniques, where unlabeled data is used to achieve more precise control over the classification probability [26].

This formal treatment of uncertainty-based abstention aligns with the broader paradigm of conditional computation, which aims to improve model efficiency by activating only necessary network components for a given input [6]. This work also presaged the use of more advanced decision policies by proposing reinforcement learning as a natural method for learning the optimal, input-dependent rules for activating or deactivating parts of the network, a concept that directly informs the learned gating mechanisms discussed in Section 3.2.

Oracle analysis provides upper bounds on adaptive inference benefits by considering hypothetical agents with perfect knowledge of optimal exit points for each sample [54]. Oracle performance establishes the maximum possible efficiency gains for given datasets and model families, guiding realistic expectations for practical methods.

Formally, let $\{f^i\}_{i=1}^M$ be the classifiers at exit layers, each classifier f^i having a cost C^i with $C^1 \leq C^2 \leq \cdots \leq C^M$. For an input sample $\mathbf{x}$ with true label Y, we denote by

$$S(\mathbf{x}) = \left\{ i \in \{1, \ldots, M\} \mid f^i(\mathbf{x}) = Y \right\}$$

the set of heads producing a correct prediction. The *oracle rule* always selects the first correct head if any, otherwise the last one:

$$\delta^*(\mathbf{x}) = \begin{cases} \min S(\mathbf{x}), & \text{if } S(\mathbf{x}) \neq \emptyset, \\ M, & \text{otherwise.} \end{cases}$$

The cost per sample under the oracle decision is

$$\mathrm{C}^*(\mathbf{x}) = \begin{cases} \min\{\, C^i : i \in S(\mathbf{x}) \,\}, & \text{if } S(\mathbf{x}) \neq \emptyset, \\ C^M, & \text{otherwise,} \end{cases} \qquad \text{with} \qquad C^1 \leq \mathrm{C}^*(\mathbf{x}) \leq C^M.$$

The oracle is correct whenever at least one head predicts the true label. Hence, the oracle accuracy is defined as

$$\mathrm{Acc}_{\mathrm{oracle}} \;=\; \mathbb{P}\left(\bigcup_{i=1}^{M} \{\, f^i(\mathbf{X}) = Y \,\} \right) \quad .$$

This quantity satisfies

$$\underbrace{\mathbb{P}\big(f^M(\mathbf{X}) = Y\big)}_{\mathrm{Acc}_{\mathrm{last}}} \;\leq\; \mathrm{Acc}_{\mathrm{oracle}} \;\leq\; 1 \;\; .$$

The inequality can be strict when intermediate heads correctly classify samples that the final head misclassifies. As a consequence, a multi-exit network can theoretically achieve a **higher maximum accuracy** than the same backbone used with a single final head.

2.3.4 Rigorous Guarantee Frameworks

The brittleness of heuristic confidence measures under distribution shifts has motivated development of frameworks providing formal performance guarantees.

Reject option with guarantees

Learning with rejection formalizes the trade-off between prediction accuracy and abstention rates through PAC-Bayesian frameworks that provide generalization bounds for selective classifiers [32]. These theoretical foundations support principled design of early exit policies with performance guarantees.

Conformal prediction frameworks

Conformal prediction [135] provides distribution-free coverage guarantees for prediction sets. For significance level $1 - \alpha$, conformal predictors output sets guaranteed to contain true labels with probability at least $1 - \alpha$. Applied to early exit networks, conformal methods construct exit criteria with formal accuracy guarantees, crucial for safety-critical applications.

Adaptive Conformal Inference (ACI) [41] extends conformal prediction to non-exchangeable data sequences, addressing distribution shifts through online confidence level adjustment. This framework enables robust early exit policies that maintain coverage guarantees under evolving data distributions.

Anytime-valid confidence sequences [56] address consistency requirements in multi-exit conformal prediction. Standard conformal methods applied independently at each exit can create paradoxical situations where labels are excluded at early exits but included at later exits. AVCS constructs nested confidence sets ensuring monotonic behavior across exit depths.

2.3.5 Policy Evaluation and Comparison

Trade-off curve analysis provides comprehensive policy evaluation through accuracy versus computational cost relationships. Superior policies achieve Pareto dominance with higher accuracy at lower computational cost across operating points. Evaluation requires careful consideration of realistic computational metrics beyond theoretical FLOP counts, including memory access patterns and batching effects.

Baseline comparisons must include relevant alternatives: static models with similar average computational cost, pruned versions of full models, and other efficiency techniques. Rigorous evaluation protocols ensure fair comparisons across different optimization approaches and deployment scenarios.

The evolution from simple confidence thresholding to principled frameworks with formal guarantees reflects the field's maturation from proof-of-concept to reliable deployment solutions. Modern inference mechanisms must balance computational simplicity with theoretical rigor, providing both practical efficiency gains and reliability assurances for real-world applications.

2.4 Efficiency Methods, System Considerations, and Evaluation Frameworks

While early exit networks represent one paradigm for achieving computational efficiency, they exist within a broader ecosystem of optimization techniques, deployment architectures, and evaluation methodologies. This section examines the relationship between early exit approaches and other efficiency methods, domain-specific architectural considerations, system-level deployment challenges, and the specialized evaluation frameworks required for adaptive inference systems.

2.4.1 Comparative Analysis with Other Efficiency Methods

Early exit networks must be understood in context with other major efficiency paradigms, each offering distinct advantages and limitations that influence their applicability in different deployment scenarios.

Static optimization methods

Network pruning removes redundant parameters through magnitude-based [46], gradient-based [94], or lottery ticket hypothesis approaches [37]. Unlike early exit's dynamic depth adaptation, pruning creates uniformly smaller models that apply the same reduced computation to every input. However, these approaches synergize effectively: structural pruning can reduce per-layer costs in early exit networks, while early exit mechanisms can dynamically skip entire pruned layers, creating multiplicative efficiency gains [70].

Quantization reduces computational precision from 32-bit floating point to 16-bit, 8-bit, or even binary representations [60]. Post-training quantization applies to pre-trained models without retraining, while quantization-aware training incorporates precision constraints during optimization. Early exit networks benefit from quantization applied to both backbone and exit head computations, though careful calibration becomes crucial as reduced precision can degrade confidence estimation reliability.

Knowledge distillation transfers knowledge from large teacher models to compact student networks through soft target supervision [52]. While traditional distillation creates smaller static models, early exit networks employ self-distillation where later exits teach earlier exits within the same architecture. This hierarchical knowledge transfer proves particularly effective, with later exits providing rich supervision signals that significantly improve early exit performance [143].

Dynamic computation methods

Mixture-of-Experts (MoE) achieves conditional computation through sparse expert selection rather than depth adaptation [117]. While early exit networks dynamically vary depth (sequential layers executed), MoE models dynamically vary width (parallel experts activated). These approaches represent orthogonal optimization dimensions: early exit optimizes average computational cost, while MoE maintains fixed per-token cost but dramatically increases model capacity. The memory implications differ substantially—early exit networks add minimal parameters for exit heads, while MoE models require massive memory for expert storage.

Speculative decoding accelerates autoregressive generation through draft-verification paradigms where small models generate candidate sequences verified by large models [75]. This approach shares conceptual similarities with cascaded early exit systems: computationally cheap initial processing followed by expensive verification when needed. Layer

Skip [84] represents convergence of these paradigms, using early transformer layers as draft generators verified by remaining layers within the same model.

Adaptive computation time (ACT) [43] provides another dynamic computation paradigm where models learn the number of computation steps required for each input. Unlike early exit's discrete layer decisions, ACT enables continuous computation allocation through learned halting probabilities. However, ACT's continuous formulation creates training complexity that limits practical adoption compared to early exit's discrete, interpretable exit points.

2.4.2 Domain-Specific Architectural Adaptations

Different application domains present unique challenges that necessitate specialized architectural solutions and deployment considerations for early exit networks.

Large language models and autoregressive generation

Token-level exit decisions distinguish LLM applications from traditional classification tasks. During autoregressive generation, each token position may require different computational complexity, necessitating fine-grained exit policies that adapt dynamically within sequences [114]. In this context, autoregressive means that the model generates text one token at a time, with each new token conditioned on all previously generated tokens. This token-level granularity creates opportunities for substantial efficiency gains but introduces complex state management challenges. **Key-Value (KV) cache management** represents the primary technical challenge for early exit in autoregressive models. The self-attention mechanism requires Key and Value vectors from all previous tokens at every layer. When token $t-1$ exits at layer E, its KV states for layers $L+1$ to N remain uncomputed. Current solutions employ state copying—replicating layer L states to fill missing deeper layer slots—but this approximation can propagate errors throughout generation, degrading quality [17].

Generation coherence requires maintaining semantic consistency when computational paths vary across tokens within sequences. Unlike classification tasks with independent samples, generation tasks must preserve discourse coherence, reference resolution, and stylistic consistency despite varying exit patterns. Research addresses this through specialized training objectives that encourage consistent generation quality across different exit distributions.

Vision transformers and spatial computation

Patch-level adaptive processing extends early exit concepts to spatial dimensions, recognizing that different image regions may require varying computational complexity. Foreground objects, complex textures, and region boundaries may demand deeper processing, while uniform backgrounds can exit early [93]. This spatial adaptation enables more fine-grained efficiency optimization than sample-level decisions.

Attention mechanism integration in Vision Transformers presents opportunities for exit placement guided by attention saturation. Research demonstrates that attention patterns stabilize at different depths for different input complexities, providing natural indicators for exit decisions [33]. Exit placement after attention blocks maintains the global receptive field properties crucial for vision tasks.

Multi-scale processing architectures like hierarchical Vision Transformers create natural exit points at resolution transitions. Different scales capture features of varying complexity—early scales for global layout, later scales for fine details—enabling content-adaptive scale selection through early exit mechanisms.

Multimodal and cross-modal applications

Cross-modal exit decisions introduce complexity where exit choices must consider confidence across multiple modalities. Vision-language models may achieve high confidence in visual understanding but low confidence in textual reasoning, requiring sophisticated exit policies that balance multimodal uncertainty [127]. Exit decisions may vary by modality, with visual processing exiting early while textual processing continues to deeper layers.

Modality-specific computational requirements necessitate asymmetric exit strategies. Visual processing may require more computation for complex scenes, while language processing may need deeper reasoning for ambiguous contexts. Unified exit policies must account for these domain-specific complexity patterns.

2.4.3 System-Level Deployment Considerations

The transition from algorithmic early exit benefits to practical system-level efficiency requires addressing fundamental challenges in hardware utilization, runtime systems, and distributed deployment architectures.

The batching dilemma and hardware constraints

Dynamic execution versus static hardware creates the fundamental deployment challenge for early exit networks. Modern GPU architectures achieve high throughput through Single Instruction, Multiple Data (SIMD) execution, processing large batches with uniform computational graphs. Early exit introduces Multiple Instruction, Multiple Data (MIMD) patterns where different samples require different computational paths, creating synchronization bottlenecks [137].

Tail latency problems emerge when batched samples have varying exit points. GPU execution must wait for the slowest sample (deepest exit) to complete, eliminating latency benefits for early-exiting samples within the batch. This synchronization requirement can completely negate the computational savings that early exit algorithms provide.

Solutions and workarounds include dynamic batching with timeout-based dispatch, continuous batching for autoregressive models that decouples request-level and iteration-level batching [151], and specialized hardware designed for dynamic neural networks. However, these solutions often require significant systems engineering and may not be available in all deployment environments.

Edge-cloud architectures and split computing

Hierarchical deployment models naturally align with early exit architectures through split computing paradigms. Edge devices execute initial layers with early exit capabilities, while cloud resources handle complex samples requiring deeper processing [91]. This distribution minimizes communication overhead by processing simple samples locally and offloading only challenging cases.

Network-aware exit policies adapt exit decisions to communication constraints, network latency, and bandwidth availability. High-latency networks may favor more aggressive local exits to avoid communication costs, while high-bandwidth connections enable more conservative exit policies that prefer cloud processing for better accuracy.

Privacy and security implications of split computing require careful consideration. Early exit decisions potentially leak information about input complexity, while intermediate feature transmission may expose sensitive data. Privacy-preserving early exit mechanisms must balance efficiency gains with information disclosure risks.

Runtime systems and framework support

Dynamic graph execution requires runtime systems capable of managing conditional computation paths efficiently. Traditional deep learning frameworks optimized for static computational graphs struggle with the dynamic control flow inherent in early exit networks. Specialized frameworks like JAX enable more natural expression of dynamic computation through just-in-time compilation and dynamic shapes.

Memory management becomes complex when different samples follow different computational paths. Memory allocation strategies must account for varying activation sizes, intermediate buffer requirements, and garbage collection patterns that differ across exit points. Efficient memory pooling and reuse strategies become crucial for practical deployment.

Profiling and monitoring of early exit systems requires new metrics and tools. Traditional profiling focuses on static computational graphs, while early exit systems need dynamic performance analysis including exit distribution tracking, confidence calibration monitoring, and adaptive policy performance evaluation.

2.4.4 Evaluation Methodologies and Benchmarking

Early exit networks require specialized evaluation frameworks that capture the inherent trade-offs between accuracy and computational efficiency, necessitating departure from single-point metrics toward comprehensive trade-off analysis.

Trade-off curve analysis

Accuracy versus computational cost curves provide the primary evaluation methodology for early exit systems. These curves plot task-appropriate accuracy metrics (Top-1 accuracy, BLEU score—*Bilingual Evaluation Understudy*, a metric comparing machine-generated text against reference translations using n-gram overlap, F1-score) against computational cost measures (average FLOPs, latency, energy consumption) across different exit policy parameters [116]. Superior methods achieve Pareto dominance—higher accuracy at lower computational cost—across operating points.

Risk-Coverage curves from selective classification theory offer alternative evaluation perspectives, plotting selective risk (error rate on non-abstained samples) against coverage (fraction of samples receiving predictions) [40]. This framework explicitly captures the reliability-efficiency trade-off crucial for safety-critical applications where abstaining from prediction may be preferable to incorrect answers.

Operating point selection requires careful consideration of deployment constraints. Different applications prioritize different regions of trade-off curves: real-time systems emphasize low-latency points, energy-constrained devices focus on low-power regions, while accuracy-critical applications operate at high-accuracy points despite computational costs.

Benchmarking frameworks and standardized evaluation

ELUE (Efficient Language Understanding Evaluation) [148] provides standardized benchmarking for efficient NLP methods, including early exit approaches. ELUE encompasses multiple tasks, unified evaluation protocols, and efficiency scores that combine accuracy and computational metrics. This standardization enables fair comparison across different early exit methods and other efficiency techniques.

Baseline establishment requires comprehensive comparison sets including: full model performance (upper bound), computationally equivalent smaller models (fair comparison), statically pruned versions (alternative efficiency approach), and oracle performance bounds

(theoretical upper limit). These baselines provide context for evaluating early exit benefits and limitations.

Evaluation protocols must specify crucial details for reproducibility: exact hardware platforms for latency measurements, batching strategies during evaluation, warm-up procedures for stable timing, and statistical significance testing across multiple runs. These details significantly impact measured performance and comparison validity.

Computational cost metrics and realistic evaluation

Beyond FLOP counting: While theoretical FLOP reduction provides initial efficiency estimates, realistic evaluation requires measuring actual computational costs on target hardware. Memory access patterns, cache behavior, and parallelization efficiency significantly influence real-world performance, often diverging from theoretical predictions.

Energy consumption measurement becomes increasingly important for mobile and edge deployment scenarios. Energy costs include both computational processing and memory access, with different exit patterns creating varying energy profiles. Specialized tools for energy profiling enable accurate measurement of early exit energy benefits.

System-level metrics capture deployment reality: wall-clock latency includes all overheads, throughput measures requests processed per unit time under realistic load, and memory utilization tracks peak and average memory requirements. These metrics provide practical deployment guidance beyond algorithmic efficiency analysis.

2.5 Towards Practical Adaptive Inference

The literature review reveals that early exit networks have evolved from foundational proof-of-concepts to sophisticated systems capable of substantial efficiency gains across diverse domains. However, two critical research gaps emerge as particularly pressing in the current landscape of ubiquitous artificial intelligence deployment.

Budget-Constrained Classification in Computer Vision: While existing early exit methods provide accuracy-efficiency trade-offs, real-world deployment scenarios often require **strict computational budget adherence** rather than threshold-based heuristics. Edge computing environments, mobile devices, and real-time systems demand predictable resource consumption with formal guarantees that computational limits will never be exceeded. Current approaches lack principled frameworks for direct budget specification and enforcement, creating a fundamental mismatch between algorithmic capabilities and deployment requirements. This gap necessitates novel theoretical frameworks that can transform early exit optimization from confidence-based heuristics to rigorous budget-constrained classification with provable computational bounds.

Efficient Large Language Models in the Era of Ubiquitous AI: The explosion of large language model deployment across diverse applications—from mobile assistants to cloud services—has created unprecedented demands for inference efficiency. While traditional early exit approaches focus on encoder architectures, the autoregressive nature of modern LLMs presents unique challenges including token-level decision granularity, KV-cache state management, and generation coherence preservation. The field requires specialized frameworks that can maintain the high accuracy standards essential for language understanding while providing substantial computational savings. This challenge is particularly acute as LLMs become increasingly integrated into everyday applications where both efficiency and reliability are paramount.

These two research directions represent complementary approaches to adaptive inference: budget-constrained methods for resource-limited scenarios and accuracy-preserving

methods for quality-critical applications. Together, they address the fundamental tension between the computational demands of modern AI and the practical constraints of real-world deployment, paving the way for truly practical adaptive inference systems that can democratize access to advanced AI capabilities across diverse computing environments.

The following chapters present our contributions to addressing these critical gaps. Chapter 3 introduces EERO (Early Exit with Reject Option), a novel framework that transforms the budget-constrained classification challenge into a principled classification with reject option problem. By leveraging exponential weights aggregation and providing hard computational budget guarantees, EERO enables direct specification of computational constraints while maintaining competitive accuracy across diverse architectures and datasets. This approach represents a fundamental shift from confidence-based heuristics to rigorous budget-aware optimization with formal theoretical guarantees.

Chapter 4 then extends this adaptive inference perspective to large language models, delivering accuracy-preserving early exits tailored to autoregressive generation.

Chapter 3

Early Exit with Reject Option (EERO)

The increasing complexity of advanced machine learning models requires innovative approaches to manage computational resources effectively. One such method is the Early Exit strategy, which allows for adaptive computation by providing a mechanism to shorten the processing path for simpler data instances. In this chapter, we propose EERO, a new methodology to translate the problem of early exiting to a problem of using multiple classifiers with reject option in order to better select the exiting head for each instance. We calibrate the probabilities of exiting at the different heads using aggregation with exponential weights to guarantee a fixed budget. We consider factors such as Bayesian risk, budget constraints, and head-specific budget consumption. Experimental results demonstrate that our method achieves competitive compromise between budget allocation and accuracy.

3.1 Introduction

Nowadays, vision models are increasing in size rising the issue of their complexity and computation costs. There exist different strategies to train lighter deep learning networks, such as quantization and pruning [81], distillation [130], and dynamic inference where the network adapts its topology on the fly to the input data [47]. Among them, Early Exit [71] is an orthogonal approach which aims at adapting the amount of computation to each input data point, exploiting that most neural networks can be approximated as a stack of layers which process the data sequentially. The idea is to add auxiliary heads at regular intervals along the network (see Figure 3.1) which are able to produce a prediction with the current state of the features. The intuition is that easy cases can be processed with the first few layers. Those heads can also be used to analyze model's layers [15, 65] and improve training [128].

Despite its advantages, one of the main challenges in applying Early Exit during inference is to determine the appropriate moment to exit for a given input. This objective is intricately connected to assessing confidence in the predictions of neural networks. Importantly it requires to build a rule – that usually relies on a test that asks whether the score for the prediction is higher or not than a given threshold – at the level of each auxiliary head of the network and one possible ultimate goal may be to calibrate all thresholds appropriately in order to meet an overall objective. Specifically, in our contribution, we focus on *Budgeted Batch classification* [57], a strategy where a fixed computational budget is allocated for processing a batch of data. The strategy involves calculating thresholds to allocate these resources efficiently across different data points enhancing overall accuracy. In principle, this approach offers improved performance

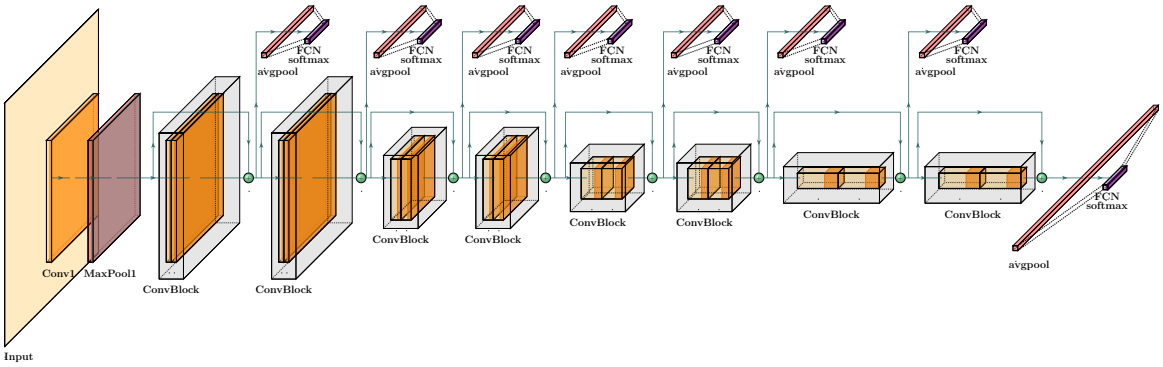


Figure 3.1: Illustration of the Early Exit principle in a convolutional architecture.

because it allows for the conservation of computational resources on simpler cases, which can then be reallocated to improve accuracy on more complex cases. However, practical investigations into this framework have been limited, with most studies concentrating on architectural design or the reliability of confidence scores. Those few that have addressed Budgeted Batch Classification often make additional assumptions for practicality [57] or utilize less-than-ideal algorithms without fully exploring the associated mathematical challenges [142].

Contributions In this work, we present EERO (Early Exit with Reject Option), a novel framework designed to augment Early Exit strategies in Budgeted Batch Classification for inference tasks. The EERO methodology progresses through a series of steps: Initially, we train auxiliary heads to refine decision-making. Subsequently, we translate the budget limitation to probabilities of classification at each head based on aggregation with exponential weights [23]. We then use a calibration set and those probabilities to calibrate head specific exit thresholds. We based this calibration of the thresholds on learning with rejection option arguments [18]. (Also named *learning with abstention* and later *selective learning* in the literature.) Finally, for each data point and at the level of each head we compute a score. If this score exceeds the threshold, we take this head output as the final prediction. Otherwise, we proceed with the following head in a similar way.

Our key contributions are: (i) Developing an optimal classification process within a specified GFlops[1] budget, ensuring efficient data processing while balancing accuracy; (ii) Adapting and generalizing EERO to various architectures like ResNet [50], ConvNext [89], and MSDNet [57], suitable for multiple auxiliary heads; (iii) Validating our approach through extensive benchmarks on CIFAR-100 and ImageNet datasets, proving its effectiveness in maintaining resource constraints, reducing computational load, and enhancing model accuracy.

This chapter is organized as follows: Section 3.2 reviews relevant literature. Section 3.3 introduces our proposed EERO method, including its statistical framework (Section 3.3.1) and detailed methodology (Section 3.3.2). Section 3.4 presents experimental results, applying EERO on ImageNet.

3.2 Related work

One of the first works on Early Exit [128] proposed to use auxiliary heads and a weighted sum of losses. The focus was essentially on the design and the positions of the auxiliary heads. MSDNet [57] is an architecture where the heads are carefully designed so that

[1]Flops: floating point operations per second.

early features which are assumed to be unsuitable for classification are refined. Although this leads to better performances, the complex resulting heads reduce their number, and therefore the flexibility of the model, and makes it more difficult to adapt to new architectures. Overall, the position and the number of the auxiliary heads remains an open discussion for convolutional networks [83] and transformers [4].

Determining the optimal timing for an early exit in neural network inference is a crucial aspect of efficient model design. A widely used method is the *Threshold-Based Approach*, where the exit decision is based on a predefined threshold on auxiliary heads output probabilities, focusing on different metrics of the probability distribution [57, 8]. Another strategy is the *Patience-Based Strategy*, requiring consecutive heads to agree on a prediction before exiting, thus leveraging sequential predictions for reliability [156]. Additionally, more sophisticated methods involve *Halting Scores*, where scores are accumulated from each layer's output, integrating model confidence into the exit process [36]. In complex scenarios, *Reinforcement Learning Techniques* are employed to finely balance computational efficiency and prediction accuracy, especially in dynamic inference settings [141, 144].

In the context of *Budgeted batch classification*, thresholds are not determined for individual images but for a batch as a whole, facilitating a more strategic deployment of computational resources. To date, the literature presents two notable algorithms addressing this problem in the realm of Early Exit strategies. The authors in [142] employ a genetic algorithm to optimize these thresholds. Conversely, the work [57] operates under the assumption that each exit point in the network has an equal and predetermined likelihood of accurately classifying an image. While this simplifies the optimization problem, it introduces an additional hyper-parameter that is not inherently related to computational complexity.

In addition, other contexts have been well studied in the literature, such as model cascading or learning-to-defer. Model cascading involves using separate, independent models of increasing complexity, where each model decides whether to classify an instance or defer to the next model in the sequence [61, 45]. Learning to defer, on the other hand, transfers decision-making to human operators when the model's confidence is low [97, 12]. Our work differs by presenting a comprehensive formulation that is grounded on the principles of reject option learning theory on a multi headed classification model with a focus on minimizing cost.

Learning with the reject option, a method to abstain from predictions under uncertainty, is crucial in our work. Initially explored in [18] and advanced through conformal prediction [134, 135], this concept has evolved significantly [51, 95, 42, 152, 74, 22, 26]. It generally encompasses strategies for predefined coverage, rejection rates, or a balance of both. In EERO, rejection is adapted at the head level, deciding if an instance $\mathbf{x}$ should exit early or continue processing. This application to deep learning for energy efficiency is novel, though the reject option has been previously applied to various learning problems [27, 28].

3.3 Method

In this section, we introduce the Early Exit with Reject Option (EERO) framework, applying reject option learning theory to enable efficient early exits in neural networks. We detail the statistical underpinnings and describe how EERO strategically manages resources to balance accuracy with computational expenditure. This approach not only optimizes performance but also ensures strict compliance with predefined computational constraints.

3.3.1 Statistical framework – classification with reject option

This section describes the mathematical framework for Early Exit. Let $(\mathbf{X}, Y)$ be a random couple distributed according to a distribution $\mathbb{P}$ on $\mathcal{X} \times [K]$, where $[K] := \{1, \dots, K\}$. Here, $\mathcal{X} \subset \mathbb{R}^d$ is the feature space, and Y is the label corresponding to the feature $\mathbf{X}$. We focus on the problem of K-class classification with $K \geq 2$ and one of our goals is to build a prediction rule $g : \mathcal{X} \to [K]$ that reduces the misclassification risk $\mathbb{P}(g(\mathbf{X}) \neq Y)$. This risk is minimized by the Bayes rule g^* that is given, in the multi-class setting, for all $\mathbf{x} \in \mathcal{X}$ by

$$g^*(\mathbf{x}) = \text{argmax}_{k=1,\dots,K} p_k(\mathbf{x}) \ , \tag{3.1}$$

where $p_k(\mathbf{x}) = \mathbb{P}(Y = k | \mathbf{X} = \mathbf{x})$ are the conditional probabilities. Because the distribution $\mathbb{P}$ is unknown, the Bayes rule itself is unknown, and we need to approximate it. In general, this approximation seeks to maximize the classification accuracy. In our case, the goal is also to reduce the energy consumption of the model, and thus, we assume a constraint on the computation budget available. Notably, we will show that the latter is strongly connected to classification with reject option (classification with abstention) that we describe in Section 3.3.2.

The framework of classification with reject option assumes that classifiers are allowed to abstain from classifying (on the empirical side, this means that the classifier abstains on a given proportion of the data). Let $\varepsilon \in (0, 1)$ be a parameter that denotes the probability of classifying an instance. The optimal rule that abstains with probability $1 - \varepsilon$ is given by:

Definition 3.3.1. Let $\varepsilon \in (0, 1)$. The optimal classifier with $1 - \varepsilon$ rejection rate is defined as

$$h_\varepsilon^* \in \text{argmin}_h \{\mathbb{P}(\{h(\mathbf{X}) \neq Y\} \cap \{h(\mathbf{X}) \neq \mathfrak{R}\}) \qquad \text{s.t. } \mathbb{P}(h(\mathbf{X}) = \mathfrak{R}) = 1 - \varepsilon\} \ , \tag{3.2}$$

where h is a classifier that is allowed to reject, that is, $h : \mathcal{X} \to [K] \cup \{\mathfrak{R}\}$ and $\mathfrak{R}$ is the output when the classifier rejects all elements from $[K]$.

The opportunity of using the reject option is important in applications where ambiguity occurs between classes, which is often the case when the total amount of classes K is large. There are several ways to handle this reject option. In this chapter, we constrain the rejection rate as it is in accordance with the resource limitation (see Sections 3.3.2 and 3.3.2).

Now, we aim at providing the explicit expression of the optimal rule given by Definition 3.3.1. Let us denote by s, the score function defined for each $\mathbf{x} \in \mathcal{X}$ by

$$s(\mathbf{x}) = \max_{k \in [K]} \{p_1(\mathbf{x}), \dots, p_K(\mathbf{x})\} \ . \tag{3.3}$$

From the definition of the Bayes rule (3.1), we can establish the following characterization of the optimal rule.

Proposition 3.3.2. *Assume the cumulative distribution function (CDF) F_s of $s(\mathbf{X})$ is continuous. Then, for all $\mathbf{x} \in \mathcal{X}$*

$$h_\varepsilon^*(\mathbf{x}) = \begin{cases} g^*(\mathbf{x}) & \text{if} \quad F_s(s(\mathbf{x})) \geq 1 - \varepsilon \\ \mathfrak{R} & \text{otherwise.} \end{cases}$$

In particular, we have $\mathbb{P}(h_\varepsilon^*(\mathbf{X}) = \mathfrak{R}) = 1 - \varepsilon$.

Proposition 3.3.2 extends the result established in [26] (*c.f.* Proposition 1) to multi-class setting. Its proofs can be found in the Sec. 3.9 of the Appendix. The main assumption used to build this result is the continuity assumption on the CDF of $s(\mathbf{X})$ and requires that the random variable has no atoms. It ensures that the classifier h_ε^* has a rejection rate exactly $1 - \varepsilon$. In the next section, we will see that from an empirical point of view, this condition can always be satisfied through randomization.

3.3.2 Data-driven procedure

Before considering the reject option arguments and since we deal with an Early Exit strategy, we train a neural network with M possible exits that correspond to $M-1$ auxiliary heads and the classical output of the last layer, as illustrated in Figure 3.1. To this end, we collect a *labeled* dataset $\mathcal{D}_{n_1}$, that consists of $n_1 \in \mathbb{N}$ *i.i.d.* copies of $(\mathbf{X}, Y)$, and train for each head ℓ an estimator $(\hat{p}_1^\ell, \ldots, \hat{p}_K^\ell)$ of the conditional probability vector $(p_1, \ldots, p_K)$. Since larger ℓ means that we go deeper in the network, it is reasonable to assume that for all layers indices $\ell, \ell' \in [M-1]$ with $\ell < \ell'$, the auxiliary head ℓ' consumes more resources than the auxiliary head ℓ and in general, yet this increase in consumption might come with a better accuracy.

Classification rule based on reject option

Our methodology highlights that early exiting can be efficiently performed, borrowing tools from learning with reject option. Let us then define, for each auxiliary head ℓ, a rejection rate $1-\varepsilon^\ell \in (0,1)$ – that will be specified later. In order to specify the prediction rule for each head, we will mimic the optimal rule provided by Proposition (3.3.2) using the plug-in principle. This step requires collecting a sample of *unlabeled* instances $\mathcal{D}_N$, that consists in N *i.i.d.* copies of $\mathbf{X}$. Since the process is the same at each auxiliary head, let us develop our methodology for a specific head ℓ.

The goal is to understand whether the head ℓ is a good early exit for a given instance $\mathbf{x} \in \mathcal{X}$. Translating this into the classification with reject option vocabulary, the question becomes whether classifier

$$\hat{g}^\ell(\cdot) = \operatorname{argmax}_{k\in[K]}\{\hat{p}_k^\ell(\cdot) + u_k\} \ , \tag{3.4}$$

should classify the instance $\mathbf{x}$ or reject it. The u_k variable is introduced to randomize the $\hat{p}_k^\ell$ estimations for a technical reason that we now explain. Let us denote by $(u_k)_{k\in[K]}$ *i.i.d.* variables which follow a uniform distribution on $[0,u]$ with u being a non-negative real number which is usually chosen very small. (In practice, we set $u = 10^{-5}$.) It ensures that the random variables $p_k^\ell(\mathbf{X}) + u_k$ has no atoms which is the key argument (see proof in appendix 3.9) to control suitably the rejection rate of the produced classifier with reject option. Formally, this classifier is the empirical counterpart of the classifier with reject option given by Proposition 3.3.2 and is based on the empirical score function given for all $\mathbf{x}$ by

$$\hat{s}^\ell(\mathbf{x}) = \max_{k\in[K]} \left\{\hat{p}_1^\ell(\mathbf{x}) + u_1, \ldots, \hat{p}_K^\ell(\mathbf{x}) + u_K\right\} \ . \tag{3.5}$$

Then we can derive the plug-in estimator of the classifier with reject option h_ε^* at the level of the ℓ-th auxiliary head.

Definition 3.3.3. For all $\mathbf{x} \in \mathcal{X}$

$$\hat{h}_{\varepsilon^\ell}^\ell(\mathbf{x}) = \begin{cases} \hat{g}^\ell(\mathbf{x}) & \text{if} \quad \hat{F}_{\hat{s}^\ell}(\hat{s}^\ell(\mathbf{x})) \geq 1-\varepsilon^\ell \\ \mathfrak{R} & \text{otherwise,} \end{cases}$$

where conditional on the dataset $\mathcal{D}_n$, we denote by $\hat{F}_{\hat{s}^\ell}(t) = \frac{1}{N}\sum_{\mathbf{X}\in\mathcal{D}_N} \mathbb{1}_{\left\{\hat{s}^\ell(\mathbf{X})\leq t\right\}}$ the empirical CDF of $\hat{s}^\ell(\mathbf{X})$ on the *unlabeled* sample $\mathcal{D}_N$.

Remark 3.3.4. The score function $\hat{s}^\ell$ in the above definition can be replaced by any suitable metrics providing an estimation of the prediction confidence. We tried entropy-based confidence and breaking ties [90] (*i.e.,* the difference between the two highest scores) in our experiments and selected the latter which performed slightly better.

According to the above definition, the classifier $\hat{h}^{\ell}_{\varepsilon^{\ell}}$ abstains when it is not confident about the classification. In our case, having $\hat{h}^{\ell}_{\varepsilon^{\ell}}$ assigns the output $\mathfrak{R}$ to a given instance $\mathbf{x}$ means that the observation should not be treated by $\hat{h}^{\ell}_{\varepsilon^{\ell}}$, but rather should be delayed to next auxiliary head treatment. In contrast, when $\hat{h}^{\ell}_{\varepsilon^{\ell}}(\mathbf{x}) = \hat{g}^{\ell}(\mathbf{x})$, then we use the early exit at the head ℓ and the observation $\mathbf{x}$ does not go through the rest of the network, thus reducing the computation. We can establish the following result whose proof can be found in Section 3.9 of the Appendix.

Proposition 3.3.5. *For all $\ell \in [M]$ and all $\varepsilon^{\ell} \in (0,1)$, there exists a constant $C > 0$ such that, whatever the distribution $\mathbb{P}$ of the data and whatever the estimators $\hat{s}^{\ell}$ of the score function s we consider, we have*

$$\left|\mathbb{P}\left(\hat{h}^{\ell}_{\varepsilon^{\ell}}(\mathbf{X}) = \mathfrak{R}\right) - (1-\varepsilon^{\ell})\right| \leq \frac{C}{\sqrt{N}} \ .$$

The above result confirms that the rejection rate of the classifier $\hat{h}^{\ell}_{\varepsilon^{\ell}}$ is indeed of the right order. However, this result suggests that the head ℓ might reject more data than it should (by a proportion of order $C/\sqrt{N}$ which is not suitable from the budget perspective. In order to solve this issue, it is sufficient to impose a smaller rate of rejection. More precisely, if we replace ε^{ℓ} by $\tilde{\varepsilon}^{\ell} = \varepsilon^{\ell} + C/\sqrt{N}$ when we run our algorithm, we force the rejection rate to be less than $1-\varepsilon^{\ell}$. From our numerical study, we observed that $C = \varepsilon^{\ell}$ leads to good results in order to ensure the budget limitation.

Remark 3.3.6. Our methodology, applicable for semi-supervised learning, capitalizes on both labeled and unlabeled data, making it ideal when acquiring labels is costly. If only labeled data is available, we advise splitting the dataset, facilitating budget control theoretically.

Remark 3.3.7. The result in Proposition 3.3.5 is valid when $M > 2$ only in the situations where the sets of classified instances by all heads are disjoint, meaning that each image is rejected by all heads but one. Since this is unlikely in practice, we need to adjust the rejection rate at the level of the head ℓ based on the rejection rates of the earlier heads. We will elaborate on this in Section 3.3.2.

Calibration of the rejection rates

To fully exploit the layered complexity of deep learning networks, our EERO methodology must handle multiple exits. We incorporate an aggregation with exponential weights [11, 23] to optimize the decision-making process at various network depths. This adaptation is crucial for leveraging the diverse representational capabilities of neural networks, ensuring computational efficiency, and maintaining high accuracy across multiple exit points. We recall that our main constraint here is the maximum allowed budget B in GFlops.

Let us then develop the process to build the vector $\hat{\boldsymbol{\varepsilon}} = (\hat{\varepsilon}^1, \ldots, \hat{\varepsilon}^M)$ of discrete probabilities that provides us the rates of classifying at each head (in other words, the $1-\hat{\varepsilon}^{\ell}$ are the rejection rates). As inputs, we assume

- we have already trained all heads classifiers that are called $\hat{g}^1 \ldots \hat{g}^M$ (based on the first labeled dataset $\mathcal{D}_{n_1}$);
- we have already computed for each of the classifiers $\hat{g}^{\ell}$ an evaluation of its risk $\hat{R}^{\ell} = \frac{1}{n_2}\sum_{(\mathbf{X},Y)\in\mathcal{D}_{n_2}} \mathbb{1}_{\{\hat{g}^{\ell}(\mathbf{X})\neq Y\}}$ based on a second labeled dataset $\mathcal{D}_{n_2}$ that consists in *i.i.d.* copies of $(\mathbf{X}, Y)$ – notice that these error rates are already computed during any classic deep learning training;

- we have a prior distribution $\boldsymbol{\pi} = (\pi^1, \ldots, \pi^M)$ on the simplex Λ_{M-1} defined as $\pi^\ell = \frac{\left(\hat{B}^\ell\right)^{-1}}{\sum_{j=1}^M \left(\hat{B}^\ell\right)^{-1}}$, where $\hat{B}^\ell$ is the budget required by the head classifier $\hat{g}^\ell$ to provide an inference for one instance,
- we have fixed the overall budget B we are allowed to use and the size T of the batch of new data points we need to predict.

Our proposal is based on aggregation with exponential weights. The vector $\hat{\boldsymbol{\varepsilon}} = (\hat{\varepsilon}^1, \ldots, \hat{\varepsilon}^M) \in \Lambda_{M-1}$ is solution in $\boldsymbol{\varepsilon} = (\varepsilon^1, \ldots, \varepsilon^M)$ of the following minimization problem:

$$\min_{\boldsymbol{\varepsilon} \in \mathbb{R}^M} \sum_{\ell=1}^M \varepsilon^\ell \hat{R}^\ell + \beta \sum_{\ell=1}^M \varepsilon^\ell \log\left(\frac{\varepsilon^\ell}{\pi^\ell}\right), \tag{3.6}$$

$$s.t. \qquad \varepsilon^\ell \geq 0, \qquad \sum_{\ell=1}^M \varepsilon^\ell = 1, \qquad \sum_{\ell=1}^M \varepsilon^\ell \hat{B}^\ell \leq \bar{B}, \tag{3.7}$$

where $\beta \geq 0$ is a tuning parameter that controls the strength of the Kullback-Leibler divergence between $\boldsymbol{\varepsilon}$ and $\boldsymbol{\pi}$ and $\bar{B} = B/T$ is the average budget we can spend to infer one instance. Notice that the constraint $\sum_{\ell=1}^M \varepsilon^\ell \hat{B}^\ell \leq \bar{B}$ reads as $\sum_{\ell=1}^M (T\varepsilon^\ell)\hat{B}^\ell \leq B$. In particular, $T\varepsilon^\ell$ interprets as the number of data points that should be classified by the head ℓ so that the total budget remains less (or equal) than the allocated budget B. Considering the Lagrangian of this problem, we can exhibit the following form of the probability vector $\hat{\boldsymbol{\varepsilon}}$.

Proposition 3.3.8. *For all $\ell \in [M]$, the ℓ-th coordinates of the rejection rates vector $\hat{\boldsymbol{\varepsilon}}$ is given by*

$$\hat{\varepsilon}^\ell = \frac{\pi^\ell \exp\left\{-\frac{\hat{R}^\ell + \hat{\mu}\hat{B}^\ell}{\beta}\right\}}{\sum_{j=1}^M \pi^j \exp\left\{-\frac{\hat{R}^j + \hat{\mu}\hat{B}^j}{\beta}\right\}} \ ,$$

where $\hat{\mu} = \max\{0, \bar{\mu}\}$, with $\bar{\mu}$ being solution of

$$\sum_{\ell=1}^M (\bar{B} - \hat{B}^\ell)\pi^\ell \exp\left\{-\frac{\hat{R}^\ell + \bar{\mu}\hat{B}^\ell}{\beta}\right\} = 0 \ .$$

The only tuning parameter in the above procedure is the temperature β. Higher values force the probability vector $\hat{\boldsymbol{\varepsilon}}$ to get closer to the prior distribution $\boldsymbol{\pi}$ that is created to take into account the budget required by each head to produce a prediction. Last but not least, The empirical risk factors $\hat{R}^\ell$ in Equation (3.7) allow the rejection rates $\hat{\boldsymbol{\varepsilon}}^\ell$ to depend on the head performances on the training data, *i.e.,* layers with good classifiers will be selected more often.

One main advantage of aggregation with exponential weights is that it is supported by strong theoretical performance. Let us focus on the misclassification risk $\mathcal{R}(g) := \mathbb{P}\left(g(\mathbf{X}) \neq Y\right)$. Following the analysis in [23, 105], we can show the following result.

Theorem 3.3.9. *Consider $\hat{g}^{EERO}$ the classifier resulting from our aggregation procedure (whose formal definition is given in*(3.16)*). Conditionally on $\mathcal{D}_{n_1}$, the data used to train the heads $\hat{g}^1 \ldots \hat{g}^M$, we have*

$$\mathcal{R}(\hat{g}^{EERO}) \leq \inf_{\ell \in [M]:\ \hat{B}^\ell \leq \bar{B}} \left\{\mathcal{R}(\hat{g}^\ell) + \beta \log(1/\pi^\ell)\right\} \ .$$

Notice that the above Oracle inequality shows that our aggregation procedure $\hat{g}^{\text{EERO}}$ is at least as good (up to a log factor) as the best head that satisfies the budget constraint. In addition, without *prior* knowledge on the budget, one may consider the uniform probabilities $\pi^\ell = 1/M$ and recover the classical bound of order $\log(M)$.

Remark 3.3.10. Our methodology fits well for the problem of Budgeted Batch Classification. However, the notion of batch is not important for our purpose. As expressed in the above algorithm (3.6)-(3.7) (see also Proposition 3.3.8), the only information that is required to calibrate the probability of rejections vector $\hat{\boldsymbol{\varepsilon}}$ is the average budget for classifying one instance.

Remark 3.3.11. EERO methodology takes advantage of aggregation with exponential weights to compromise between accuracy and budget consumption. However in the particular case of only one auxiliary head, there is no need for aggregation – the budget is split between the auxiliary head and the last layer. We have developed such particular case in Appendix 3.8 and run experiments in that case as well using the ResNet-18 architecture illustrated in Figure 3.1 on CIFAR-100.

Heads with reject option

In the case of multiple heads, it is difficult to ensure the validity of Proposition 3.3.5 for all heads (*c.f.* Remark 3.3.7). However, we can guarantee a weaker result that is sufficient to ensure the good control on the allocated budget. Based on the previous section, we built M head classifiers on one hand and a probability vector $\hat{\boldsymbol{\varepsilon}}$ whose ℓ-th components gives the rates of classification at the ℓ-th auxiliary head on the other hand. The probability vector $\hat{\boldsymbol{\varepsilon}}$ takes into account both the accuracy and the resources of each head and allows achieving the suitable control on the budget. However, in the case of multiple auxiliary heads, a careful analysis of our methodology imposes some modifications according to the sequential calibration of the probabilities of classification.

The calibration of the classification rates of all heads is based on the estimation of the CDF F_s of the score function. Since there are $M-1$ auxiliary heads, we estimate $M-1$ times the function F_s. Importantly, all these estimates are built on the same dataset $\mathcal{D}_N$. While the calibration at the level of the first head is perfectly valid, the calibration of the classification rate starting from the second head needs to be adjusted to get the $\hat{\varepsilon}^\ell$ classification rate. In particular, for all $\ell \in \{2, \dots, M-1\}$, we enforce a higher classification rate as

$$\hat{\varepsilon}^\ell_{\text{seq}} = \sum_{j=1}^{\ell} \hat{\varepsilon}^j \ . \tag{3.8}$$

This choice is motivated by the fact that we need, for each head ℓ, to calibrate the threshold for the rejection rule so that at most a proportion $1 - \sum_{j=1}^{\ell} \hat{\varepsilon}^j$ of the data in $\mathcal{D}_N$ is rejected. If we consider this adjustment together with the correction we have detailed right after Proposition 3.3.5, we can show that for all $\ell \in [M]$, if we replace the probability of classification $\hat{\varepsilon}^\ell$ in Definition 3.3.3 by

$$\tilde{\varepsilon}^\ell_{\text{seq}} = \hat{\varepsilon}^\ell_{\text{seq}} + C/\sqrt{N} \ . \tag{3.9}$$

We have that $\mathbb{P}\left(\hat{h}^\ell_{\tilde{\varepsilon}^\ell_{\text{seq}}}(\mathbf{X}) = \mathfrak{R}\right) \leq 1 - \sum_{j=1}^{\ell} \hat{\varepsilon}^j$ so that we can deduce the following result.

Theorem 3.3.12. *For all $\ell \in [L]$, the classification procedure at the head ℓ is such that*

$$\mathbb{P}\left(\hat{h}^\ell_{\tilde{\varepsilon}^\ell_{\text{seq}}}(\mathbf{X}) = \mathfrak{R}\right) \leq \sum_{j=\ell+1}^{M} \hat{\varepsilon}^j \ .$$

The above statement means that after using the head ℓ, there is at most a proportion $\sum_{j=\ell+1}^{M} \hat{\varepsilon}^j$ of the data that remains to classify. This result will be illustrated in the next section through numerical experiments.

Algorithm 2: EERO method: Calibration phase and classification of a batch of data points

Require: Batch of data points: $\mathbf{X}_1, \ldots, \mathbf{X}_T \in \mathcal{X}$;
Calibration sets $\mathcal{D}_N$; Model p with M heads,
Risk of heads: R, Allowed budget: B
Ensure: Prediction: $\mathcal{P} \in [K]^T$

$\boldsymbol{F_s} \leftarrow \text{ComputeCDF}(\mathcal{D}_N)$ {Eq. (3.5)}
$\tilde{\boldsymbol{\varepsilon}}_{\mathbf{seq}} \leftarrow \text{Aggregation}(M, R, B)$ {Prop 3.3.8, Eq (3.8)–(3.9)}
$\mathcal{P} \leftarrow$ empty list
for $i = 1$ to T **do**
  for $\ell = 1$ to M **do**
    $s^\ell, g^\ell \leftarrow \text{CompOutput}(\mathbf{X}_i, p_\ell)$ {Eq. (3.4)–(3.5)}
    if $F_{s^\ell}(s^\ell(\mathbf{x})) \geq 1 - \tilde{\varepsilon}^\ell_{\text{seq}}$ **then**
      $\mathcal{P} \leftarrow \left[\mathcal{P}, g^\ell\right]$
      break
    end if
  end for
end for
return $\mathcal{P}$

3.4 Experiments

Building upon the methodological foundations established for EERO, this section seeks to empirically substantiate our approach on ImageNet dataset with the ConvNext and MSDNet network architectures. We also compare our EERO approach with MSDNet strategy to compute the exit thresholds. Those experiments[2] validate our theory and demonstrate the scalability of EERO across diverse neural network configurations.

3.4.1 EERO with multiple heads

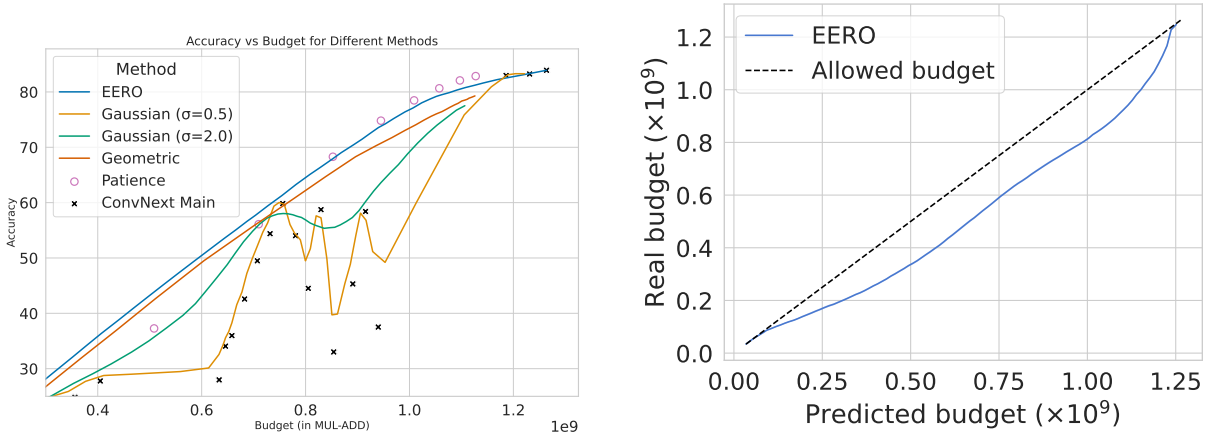
We recall that, after training the heads, the procedure consists of two steps. The first one focuses on determining the rejection rates at each head. The second one consists in specifying the rejection rule (that is, calibrating the exit thresholds) at the level of each given head, *c.f.,* Definition 3.3.3.

In particular, we exploit the methodology based on aggregation with exponential weights developed in Section 3.3.2 to specify the rejection rates.

Following the protocol from Section 3.3.2, EERO is applied to the ConvNext architecture using the ImageNet dataset. The model is adapted with auxiliary heads – See Appendix 3.6 for details on extra computational cost induced by training the auxiliary heads. The training is based on pretrained weights and a sum of cross-entropy losses from various exits. For complete hyperparameter details, refer to Appendix Tables 3.1 and 3.2.

We plot the results for the different heads and for different computation budgets in Figure 3.2a. Several observations can be made. First, we highlight an interesting phenomenon: for a given budget, our proposed multiple heads approach (Blue curve) improves the accuracy from the different exits used independently (black cross). Such observation confirms our statement in Theorem 3.3.9. Moreover, regarding the accuracy of

[2]All computations are run on a server with an Intel(R) Xeon(R) Gold 5120 CPU and a Tesla V100 GPU with 32GB of Vram and 64GB of RAM. Code associated with this chapter can be found in repository here : https://github.com/FlorianVal/Early-Exit-With-Reject-Option



(a) Accuracy vs. budget comparison

(b) Budget adherence verification

Figure 3.2: Evaluation of EERO on ConvNext architectures: (a) shows EERO's accuracy and budget against other known methods such as Patience [156, 155], Geometric distribution [57, 33] or Gaussian [78] distribution of weights on each head. ConvNext Main points correspond to the accuracy of each head alone. (b) Shows that our method accurately follows the budget given by never exceeding it.

the auxiliary heads alone, placing an exit further in the model does not necessarily result in a better accuracy when the model is not tuned for early exit.

We further check the budget used by our model in Figure 3.2b and show that our methodology respects the allowed budget by always being lower or equal to it, validating our theory from Proposition 3.3.5 and Theorem 3.3.12. On some budgets, the gap between the allowed budget and the actual consumption suggests that a better accuracy could be obtained. We also found that the rejection rates $\hat{\varepsilon}^\ell$ behave as expected as for example lower budgets favor early heads, and heads with higher risk $\hat{R}^\ell$ have lower values, thus decreasing their likelihood to be selected for the final classification score as depicted on Figure 3.3. (See also Appendix 3.7).

3.4.2 Comparison with Existing Early Exit Strategies

We extend our evaluation by comparing EERO with existing early exit strategies, demonstrating its general applicability and effectiveness across different models and scenarios.

Methods Using specific distributions of Exits

Some early exit strategies employ predefined distributions, such as Gaussian [78] or geometric [57, 33] distributions, to determine the probability of exiting at each head. We show on Figure 3.2a that these methods lack awareness of the individual risks associated with each head, which can significantly impact overall accuracy. While they may achieve comparable results on networks specifically tuned so that deeper exits yield better accuracy, they are less effective on general architectures. The lack of adaptation to the actual performance of each head can lead to suboptimal allocation of computational resources, potentially favoring less accurate exits. This is highlighted in Figure 3.2a where our method outperforms specific-distribution methods.

Patience-Based Strategies

Patience-based methods [156, 155] exit after a certain number of consecutive confident predictions, leveraging the model's consistency across layers. While these strategies can

outperform EERO in specific cases, they do not allow for explicit computational budget selection. Budget control is indirect and limited to discrete levels corresponding to different patience values, which restricts fine-grained resource allocation and may not align with strict budget constraints.

Comparison with MSDNet

In our experiments (see Appendix 3.4a, 3.4b), we compared EERO with the Multi-Scale Dense Network (MSDNet) [57], a well-known architecture specifically designed for early exiting that uses geometric distribution. The results show that EERO achieves similar accuracy and computational budget consumption as MSDNet (Figure 3.4a). However, there are key differences between the two approaches. MSDNet requires a specialized architecture and tuning to ensure that exits placed deeper in the network provide better accuracy. Moreover, MSDNet does not allow for the direct specification of a computational budget; instead, it requires iterative adjustments of exit probabilities to approximate the desired budget, which can be impractical. In contrast, EERO is model-agnostic and can be adapted to any network architecture with auxiliary heads. It allows for explicit selection of the computational budget beforehand, ensuring that the resource constraints are strictly met (Figure 3.4b). This flexibility makes EERO more suitable for applications where budget compliance is critical.

Advantages of EERO

EERO addresses the limitations of these methods by providing a general framework that is both model-agnostic and budget-aware. By incorporating the risks of each head through our aggregation method (Section 3.3.2), EERO optimally allocates computational resources to maximize accuracy under a given budget constraint. This ensures that more computational effort is devoted to heads with better performance, regardless of their position in the network. Our experimental results (Figure 3.2a) demonstrate that EERO matches or outperforms the performance of specialized methods without requiring architecture-specific tuning. It provides precise budget adherence (Figure 3.2b) and maintains competitive accuracy across various budgets and architectures.

3.5 Conclusion

In this chapter, we presented EERO, a novel mathematical framework devised to construct classification rules for Early Exit in deep learning networks. This framework targets the pivotal challenge of optimizing computational efficiency while enhancing model performance. EERO approaches the problem by modeling it as a classification with a reject option, providing a feasible solution for scenarios involving a single Early Exit. For multi-exit scenarios, our innovative approach hinges on an aggregation with exponential weights in order to tune the exiting probabilities. A key strength of EERO lies in its flexibility and generalizability, enabling it to be applied across various model architectures. This adaptability makes EERO a versatile tool, well-suited for diverse applications in the realm of efficient deep learning.

Our experiments on ImageNet and CIFAR-100 using the Resnet-18, MSDNet and the ConvNext architectures respectively demonstrated the efficiency of our framework. The results not only shed light on the intricate workings of multi-head deep neural networks but also offer practical strategies to enhance model accuracy while reducing computational budget. This has important implications for edge computing and making the field of deep learning more sustainable.

3.6 Implementation details of experiments

In Tables 3.1 and 3.2, we provide the key experimental details for our implementations, including model architectures, datasets, and training parameters. Our implementation is done by freezing the pretrained model and only training the added auxiliary heads. Involving the extra head in our process increases the computational cost by less than 4% and 2.5% in the case of ResNet-18 and ConvNext respectively. In all our experiments we set $\beta = 0.04$ and we mention that our method is not significantly affected by changes of this parameter ; notice that our theory suggests a value of order $1/\sqrt{N}$.

Table 3.1: Basic Setup of Experiments

Experiment	**Model**	**Dataset**	**Key Modifications**
ResNet with CIFAR-100	ResNet	CIFAR-100	Auxiliary heads at 7 positions, Flatten outputs
ConvNext with ImageNet	ConvNext	ImageNet	Auxiliary heads at 19 positions, Adaptive Average pooling, Flatten outputs

Table 3.2: Detailed Training Parameters

Experiment	**Training Details**	**Calibration Details**
ResNet with CIFAR-100	SGD, LR: 0.003, Batch size: 128 Train data : 49.000	Calibration set: 1,000 examples
ConvNext with ImageNet	Pretrained weights, Fine-tuned for 300 epochs, SGD, Momentum: 0.9, Weight decay: 10^{-4}, Train data : Full Imagenet	Calibration set: 5,000 examples

3.7 Analysis of aggregation weights

We examine the values of the obtained thresholds $\hat{\boldsymbol{\varepsilon}}$ as it offers interesting insights. We first display the values for all the layers and for different budgets in Table 3.3. The results highlights that: i) when higher budget are allocated, the weights produced by the aggregation put higher weights on heads with lower risks; ii) heads with high risks are assigned smaller weights by the aggregation, see for instance the weights corresponding to the heads 13 and 18 that have a higher risk and so a lower value for the ε^ℓ threshold.

This is also visible in Figure 3.5 where we show the $\hat{\boldsymbol{\varepsilon}}$ values for our EERO method together with the MSDNet like strategy.

First line shows a budget that does not allow usage of the computationally costly heads and so most data will be output on the two first heads. Second line corresponds a medium budget, where we can see that our EERO approach and the MSDNet strategy show different behaviors. Whereas the latter assumes that a sample that *reaches* a head classifier has the same constant probability to exit independently from its layer position, resulting in a monotonic behavior, our approach exploits the information of the empirical risks $\hat{R}^\ell$ (see Equation (3.6)) to modulate the importance of the heads. As we saw in Figure 3.5 and 3.6, a high risk is correlated with a low epsilon. Third line shows a permissive budget where most data will be output at the last heads as they have more accuracy.

			$6.55 * 10^{12}$ Flops		$3.21 * 10^{13}$ Flops		$5.64 * 10^{13}$ Flops	
ℓ	$\hat{R}^\ell$	π^ℓ	$\hat{\varepsilon}^\ell$	$\hat{\varepsilon}^\ell_{\text{out}}$	$\hat{\varepsilon}^\ell$	$\hat{\varepsilon}^\ell_{\text{out}}$	$\hat{\varepsilon}^\ell$	$\hat{\varepsilon}^\ell_{\text{out}}$
1	0.93	0.46	0.69	0.68	0.18	0.17	0.00	0.00
2	0.81	0.13	0.17	0.28	0.07	0.19	0.00	0.01
3	0.75	0.04	0.03	0.04	0.03	0.16	0.00	0.01
4	0.72	0.04	0.03	0.00	0.03	0.12	0.00	0.01
5	0.72	0.02	0.01	0.00	0.03	0.08	0.00	0.02
6	0.66	0.02	0.01	0.00	0.03	0.07	0.00	0.01
7	0.64	0.02	0.01	0.00	0.03	0.05	0.01	0.02
8	0.57	0.02	0.01	0.00	0.03	0.04	0.01	0.02
9	0.51	0.02	0.01	0.00	0.04	0.03	0.01	0.02
10	0.46	0.02	0.01	0.00	0.04	0.02	0.01	0.01
11	0.40	0.02	0.01	0.00	0.04	0.02	0.01	0.02
12	0.46	0.02	0.01	0.00	0.04	0.03	0.01	0.04
13	0.55	0.02	0.00	0.00	0.03	0.02	0.01	0.03
14	0.41	0.02	0.01	0.00	0.04	0.01	0.02	0.03
15	0.67	0.02	0.00	0.00	0.03	0.01	0.02	0.08
16	0.55	0.02	0.00	0.00	0.04	0.00	0.02	0.15
17	0.42	0.02	0.00	0.00	0.04	0.00	0.03	0.09
18	0.62	0.02	0.00	0.00	0.03	0.00	0.03	0.13
19	0.17	0.01	0.00	0.00	0.06	0.00	0.20	0.05
20	0.17	0.01	0.00	0.00	0.06	0.00	0.27	0.06
21	0.16	0.01	0.00	0.00	0.06	0.00	0.33	0.20

Table 3.3: Illustration of the evolution of the probability vector of classification for the different heads on ImageNET dataset with Convnext architecture and for three different budgets: each line represents an exit head, $\boldsymbol{\pi}$ is the prior distribution, $\hat{\boldsymbol{\varepsilon}}$ are the weights produced by the aggregation, and $\hat{\boldsymbol{\varepsilon}}_{\text{out}}$ are the proportion of the data that was actually classified by each head.

3.8 EERO with only 1 auxiliary head

In the case of one auxiliary head, computations of the rate of classifying $\hat{\varepsilon}^1$ (of this single head) can be simplified. In particular, we can express it analytically without the need for aggregation with exponential weights. Let $B > 0$ be the amount of *GFlops* we were allocated for the task of labeling T data points. Assume that the classifiers $\hat{h}^1_{\hat{\varepsilon}^1}$ and $\hat{g}^2$ burn off respectively $\hat{B}^1$ and $\hat{B}^2$ *GFlops* at each call with $\hat{B}^1 < \hat{B}^2$. (We assume that $B \geq T\hat{B}^1$ so that we are guaranteed to label all T points).

Proposition 3.8.1. *Let* $\hat{\varepsilon}^1 = \frac{\frac{B}{T} - \hat{B}^2}{\hat{B}^1 - \hat{B}^2}$. *Then the total amount of* GFlops *used to label* T *instances is not larger than* B.

This result highlights that our strategy succeeds to comply with the constraint of budget we imposed. The proof of the result lies in the fact that we use the early exit on a proportion $\hat{\varepsilon}^1$ of the data. Then we consume $T\hat{\varepsilon}^1\hat{B}^1$ *GFlops* (up to a $1/\sqrt{N}$ additive term – see the comment after Proposition 3.3.5 for a correction). The rest of the data is treated by the classifier $\hat{g}^2$ and then uses $T(1-\hat{\varepsilon}^1)\hat{B}^2$ *GFlops*. In total, we then get

$$T(\hat{\varepsilon}^1\hat{B}^1 + (1-\hat{\varepsilon}^1)\hat{B}^2) = B \ . \tag{3.10}$$

Remark 3.8.2. In this section, we enforce the rejection rate to exactly burn off the whole budget B – see Equation (3.10). We make this choice to explain some broader effects that illustrate the importance of using auxiliary heads (see below and Figure 3.7). On the other hand, we notify that it is extremely simple to modify the algorithm so that it consumes less than or equal to the total budget, that is, $(\hat{\varepsilon}^1 T)\hat{B}^1 + ((1-\hat{\varepsilon}^1)T)\hat{B}^2 \leq B$. In this case, the accuracy curve would always be increasing *w.r.t.* the budget.

We implement EERO using a ResNet [50] model on the CIFAR-100 [67] dataset, with auxiliary heads placed at seven different positions for early exit testing (see Figure 3.1). The model training involve a sum of cross-entropy losses from these exits. Detailed hyperparameters of this experiment are provided in Appendix 3.6 in Tables 3.1 and 3.2.

We plot the accuracy for the different exits in Figure 3.7 for different budgets. First, we can see a general trend where our procedure gradually improves the accuracy as the budget increases since images wrongly classified by the auxiliary head and correctly classified by the final layer are forwarded from the Early Exit to the latter. Then there is an accuracy decrease since augmenting the budget only affects images which are wrongly classified by both Early Exit and final model exit. Secondly, we notice that the oracle quickly outperforms the accuracy of the main model for only a fraction of the total budget required to process all images with the full model. This fact shows that Early Exits can have valid predictions, whereas the last model output is wrong. This phenomenon is known as over-thinking and motivates the potential of Early Exit methods, as a way of better understanding the flaws of deep learning models. Here, this over-thinking situation seems important, as some of the Early Exits have a better accuracy than the last model output, *c.f.,* Remark 3.8.2. A second reason explaining the oracle improvements is its capacity to save computation budget by choosing an Early Exit when both early and last exit predictions are wrong.

This advantage, only achievable in our specific context of Budgeted Batch Classification, aligns well with our primary objective of reducing computational power for our algorithms. Furthermore, it indirectly enhances the overall accuracy of our model.

3.9 Proofs

In this section, we provide the proofs of our main results.

Proof 1 (Proposition 3.3.2)**.** *Let us write*

$$\mathcal{R}(h) = \mathbb{P}\left(h(\mathbf{X}) \neq Y \ , \ h(\mathbf{X}) \neq \mathfrak{R}\right) \ ,$$

for short. We then need to solve the problem

$$h^*_\varepsilon = argmin_h \left\{\mathcal{R}(h) : \mathbb{P}(h(\mathbf{X}) = \mathfrak{R}) = 1-\varepsilon\right\},$$

where the minimum is taken over all measurable functions. Considering the Lagrangian of the above problem, we solve

$$\min_h \max_{\lambda \in \mathbb{R}^+} \underbrace{\left\{\mathcal{R}(h) + \lambda\left(\mathbb{P}(h(\mathbf{X}) = \mathfrak{R}) - (1-\varepsilon)\right)\right\}}_{:=\mathcal{L}(h,\lambda)} \ ,$$

where $\lambda \in \mathbb{R}$ is the dual variable. Observe that by weak duality we have

$$\min_h \max_{\lambda \in \mathbb{R}^+} \mathcal{L}(h,\lambda) \geq \max_{\lambda \in \mathbb{R}^+} \min_h \mathcal{L}(h,\lambda) \ ,$$

we then consider first the minimization problem over h of $\mathcal{R}$. We have

$$\mathcal{R}(h) = \mathbb{P}\left(h(\mathbf{X}) \neq Y \ , \ h(\mathbf{X}) \neq \mathfrak{R}\right) = \mathbb{E}\left[\sum_{k\in[K]} \mathbb{1}_{\{Y=k\}}\mathbb{1}_{\{h(\mathbf{X})\neq k\}}\mathbb{1}_{\{h(\mathbf{X})\neq\mathfrak{R}\}}\right]$$

$$= \mathbb{E}\left[\sum_{k\in[K]} p_k(\mathbf{X})\mathbb{1}_{\{h(\mathbf{X})\neq k\}}\mathbb{1}_{\{h(\mathbf{X})\neq\mathfrak{R}\}}\right] = \mathbb{P}\left(h(\mathbf{X}) \neq \mathfrak{R}\right) - \mathbb{E}\left[\sum_{k\in[K]} p_k(\mathbf{X})\mathbb{1}_{\{h(\mathbf{X})=k\}}\mathbb{1}_{\{h(\mathbf{X})\neq\mathfrak{R}\}}\right]$$

Moreover,

$$\mathbb{P}(h(\mathbf{X}) \neq \mathfrak{R}) = \mathbb{E}\left[\sum_{k\in[K]} \mathbb{1}_{\{h(\mathbf{X})=k\}}\mathbb{1}_{\{h(\mathbf{X})\neq\mathfrak{R}\}}\right].$$

Therefore, we can write

$$\mathcal{L}(h,\lambda) = \lambda - \lambda(1-\varepsilon) + (1-\lambda)\mathbb{P}(h(\mathbf{X}) \neq \mathfrak{R}) - \mathbb{E}\left[\sum_{k\in[K]} p_k(\mathbf{X})\mathbb{1}_{\{h(\mathbf{X})=k\}}\mathbb{1}_{\{h(\mathbf{X})\neq\mathfrak{R}\}}\right] \quad (3.11)$$

$$= \lambda\varepsilon - \mathbb{E}\left[\sum_{k\in[K]} \left(p_k(\mathbf{X}) - (1-\lambda)\right)\mathbb{1}_{\{h(\mathbf{X})=k\}}\mathbb{1}_{\{h(\mathbf{X})\neq\mathfrak{R}\}}\right] \ . \quad (3.12)$$

We need to maximize the expectation w.r.t. *h that first leads to the optimum h^*_λ is such that*

$$h^*_\lambda(\mathbf{x}) \neq \mathfrak{R} \iff \sum_{k\in[K]} \left(p_k(\mathbf{x}) - (1-\lambda)\right)\mathbb{1}_{\left\{h^*_\lambda(\mathbf{x})=k\right\}} > 0 \ , \quad (3.13)$$

for all $\mathbf{x} \in \mathcal{X}$. Moreover we have that on the event $\{h^*_\lambda(\mathbf{x}) \neq \mathfrak{R}\}$, the mapping h that maximizes $h \mapsto \sum_{k\in[K]} \left(p_k(\mathbf{x}) - (1-\lambda)\right)\mathbb{1}_{\{h(\mathbf{x})=k\}}$ is simply $h^*_\lambda(\mathbf{x}) = argmax k \in [K] p_k(\mathbf{x})$. At this level of the proof, we have shown that the problem $\min_h \mathcal{L}(h,\lambda)$ leads to the rule

$$h^*_\lambda(\mathbf{x}) = \begin{cases} argmax_{k\in[K]} p_k(\mathbf{x}) & \text{if} \quad \max_{k\in[K]} p_k(\mathbf{x}) \geq 1-\lambda \\ \mathfrak{R} & \text{otherwise} \ . \end{cases} \quad (3.14)$$

Now, we deal with the maximization of $\mathcal{L}(h^*_\lambda,\lambda)$ w.r.t. *λ. Substituting the above value of $h = h^*_\lambda$ in* (3.11)*, we can show that*

$$\mathcal{L}(h^*_\lambda,\lambda) = \lambda\varepsilon - \mathbb{E}\left[\left(\max_{k\in[K]}\left\{p_k(\mathbf{X}) - (1-\lambda)\right\}\right)_+\right] \ ,$$

where for all $a \in \mathbb{R}$, we write $(a)_+ = \max\{a, 0\}$. The above function is then concave in λ, therefore we can write the first order optimality condition as $0 \ \in \partial\mathcal{L}(h^*_{\lambda^*},\lambda^*)$. Observe that because we assumed that the r.v. $p_k(X)$ has no atom for all $\in [K]$, we have that $\mathbb{P}\left(\exists\, j \in [K] : p_k(X) = p_j(X)\right) = 0$ and then the subgradient reduces to the gradient. As a consequence, we obtain the following condition on λ^*

$$\mathbb{P}\left(\exists k \in [K] : p_k(\mathbf{X}) > \max\left\{\max_{j\in[K]}\left\{p_j(\mathbf{X})\right\}; 1-\lambda^*\right\}\right) = \varepsilon \ .$$

Notice that this last condition can rewrite as

$$\varepsilon = \mathbb{P}\left(s(\mathbf{X}) > 1-\lambda^*\right) = \mathbb{P}\left(h^*_{\lambda^*}(\mathbf{X}) \neq \mathfrak{R}\right) \ ,$$

where $s(\mathbf{X}) = \max_{k\in[K]} p_k(\mathbf{X})$, which guarantee that the optimal rule has indeed the correct rejection rate. Moreover, from the above relation, we can exhibit the value of λ. Indeed, using the continuity condition on the CDF F_s of $s(\mathbf{X})$, we have

$$\mathbb{P}\left(s(\mathbf{X}) > 1-\lambda^*\right) = \varepsilon \quad \iff \quad 1 - F_s\left(1-\lambda^*\right) = \varepsilon \quad \iff \quad \lambda^* = 1 - F_s^{-1}\left(1-\varepsilon\right),$$

where F_s^{-1} is the generalized inverse of F_s. We conclude the proof substituting this value into the expression of the optimal rule given by (3.14)*.*

Remark 3.9.1. If we replace the equality by an inequality, everything is the same except the part of λ^*. We need to consider the case where $\lambda^* = 0$ separately, and in this case, we would get $\mathbb{P}\left(h^*_{\lambda^*}(\mathbf{X}) \neq \mathfrak{R}\right) \geq \varepsilon$.

Proof 2 (Proposition 3.3.5)**.** *First, observe that conditionally to the training dataset $\mathcal{D}_n$ and due to the continuity of the CDF of $\hat{s}^\ell(\mathbf{X})$, the random variable $F_{\hat{s}^\ell}(\hat{s}^\ell(\mathbf{X}))$ is uniformly distributed. Therefore, for any $u \in [0,1]$, we have $\mathbb{P}\left(F_{\hat{s}^\ell}(\hat{s}^\ell(\mathbf{X})) \leq u\right) = u$. We then can write*

$$\begin{aligned}\left|\mathbb{P}\left(\hat{h}^\ell_{\varepsilon^\ell}(\mathbf{X}) = \mathfrak{R}\right) - (1-\varepsilon^\ell)\right| &= \left|\mathbb{E}\left[\mathbb{1}_{\left\{\hat{F}_{\hat{s}^\ell}(\hat{s}^\ell(\mathbf{X})) \leq 1-\varepsilon^\ell\right\}} - \mathbb{1}_{\left\{F_{\hat{s}^\ell}(\hat{s}^\ell(\mathbf{X})) \leq 1-\varepsilon^\ell\right\}}\right]\right| \\ &\leq \left|\mathbb{E}\left[\mathbb{1}_{\left\{\left|F_{\hat{s}^\ell}(\hat{s}^\ell(\mathbf{X}))-(1-\varepsilon^\ell)\right| \leq \left|\hat{F}_{\hat{s}^\ell}(\hat{s}^\ell(\mathbf{X}))-F_{\hat{s}^\ell}(\hat{s}^\ell(\mathbf{X}))\right|\right\}}\right]\right| \\ &\leq \left|\mathbb{E}\left[\mathbb{1}_{\left\{\left|F_{\hat{s}^\ell}(\hat{s}^\ell(\mathbf{X}))-(1-\varepsilon^\ell)\right| \leq \left\|\hat{F}_{\hat{s}^\ell}-F_{\hat{s}^\ell}\right\|_\infty\right\}}\right]\right| \leq 2\mathbb{E}\left\|\hat{F}_{\hat{s}^\ell} - F_{\hat{s}^\ell}\right\|_\infty \ ,\end{aligned}$$

where we used again in the last line the fact that $F_{\hat{s}^\ell}(\hat{s}^\ell(\mathbf{X}))$ is uniformly distributed. Moreover, we wrote $\|\hat{F}_{\hat{s}^\ell} - F_{\hat{s}^\ell}\|_\infty = \sup_{t\in\mathbb{R}} |\hat{F}_{\hat{s}^\ell}(t) - F_{\hat{s}^\ell}(t)|$. We conclude the proof using the Dvoretzky–Kiefer–Wolfowitz inequality, that states that

$$\mathbb{E}\left\|\hat{F}_{\hat{s}^\ell} - F_{\hat{s}^\ell}\right\|_\infty \leq \sqrt{\frac{\pi}{2N}} \ .$$

Proof 3 (Proposition 3.3.8)**.** *The Lagrangian of the minimization problem* (3.6)-(3.7) *is given by*

$$\mathcal{L}(\boldsymbol{\varepsilon}, \lambda, \mu) = \sum_{j=1}^M \varepsilon^j \hat{R}^j + \beta \sum_{j=1}^M \varepsilon^j \log\left(\frac{\varepsilon^j}{\pi^j}\right) + \lambda\left(\sum_{j=1}^M \varepsilon^j - 1\right) + \mu\left(\sum_{j=1}^M \varepsilon^j \hat{B}^j - \bar{B}\right) \ ,$$

with $(\lambda, \mu) \in \mathbb{R}\times\mathbb{R}^+$. Considering the KKT condition of the problem we get for all $j \in [M]$

$$\hat{R}^j + \beta\left(\log\left(\frac{\hat{\varepsilon}^j}{\pi^j}\right) + 1\right) + \hat{\lambda} + \hat{\mu}\hat{B}^j = 0,$$

that leads in turns to

$$\hat{\varepsilon}^j_{\hat{\lambda},\hat{\mu}} = \pi^j \exp\left(-\frac{\hat{R}^j + \hat{\lambda} + \hat{\mu}\hat{B}^j}{\beta} - 1\right) \ . \tag{3.15}$$

Plug-in these values into the equality constraint leads to

$$\begin{aligned}\sum_{j=1}^M \hat{\varepsilon}^j_{\hat{\lambda},\hat{\mu}} = 1 \quad &\iff \quad \sum_{j=1}^M \pi^j \exp\left(-\frac{\hat{R}^j + \mu\hat{B}^j}{\beta} - 1\right) = \exp\left(\frac{\hat{\lambda}}{\beta}\right) \\ &\iff \quad \hat{\lambda} = \beta \log\left(\sum_{j=1}^M \pi^j \exp\left(-\frac{\hat{R}^j + \mu\hat{B}^j}{\beta} - 1\right)\right) \ .\end{aligned}$$

Substituting back this value into (3.15)*, we get*

$$\hat{\varepsilon}^j_{\hat{\mu}} = \frac{\pi^j \exp\left(-\frac{\hat{R}^j + \hat{\mu}\hat{B}^j}{\beta}\right)}{\sum_{k=1}^M \pi^k \exp\left(-\frac{\hat{R}^k + \hat{\mu}\hat{B}^k}{\beta}\right)} \ .$$

According to the parameter $\hat{\mu}$, we need to consider the constraints $\hat{\mu} \geq 0$ and $\sum_{j=1}^M \hat{\varepsilon}^j_{\hat{\mu}} \hat{B}^j \leq \bar{B}$ together with the complementary condition $\hat{\mu}\left(\sum_{j=1}^M \hat{\varepsilon}^j_{\hat{\mu}}(\hat{B}^j - \bar{B})\right) = 0$. Therefore, when $\hat{\mu} \neq 0$, this parameter should be taken such that

$$\sum_{j=1}^M \hat{\varepsilon}^j_{\hat{\mu}}(\hat{B}^j - \bar{B}) = 0 \quad \iff \quad \sum_{j=1}^M \pi^j(\hat{B}^j - \bar{B}) \exp\left(-\frac{\hat{R}^j + \hat{\mu}\hat{B}^j}{\beta}\right) = 0 \ .$$

Otherwise $\hat{\mu} = 0$ and in this case, $\hat{\varepsilon}^j_{\hat{\mu}}$ becomes

$$\hat{\varepsilon}^j_{\hat{\mu}} = \frac{\pi^j \exp\left(-\hat{R}^j/\beta\right)}{\sum_{k=1}^M \pi^k \exp\left(-\hat{R}^k/\beta\right)} \ .$$

Proof 4 (Theorem 3.3.9)**.** *Let us first give a formal definition of the resulting classifier $\hat{g}^{EERO}$ from our procedure. To that end, let introduce for all $\mathbf{x} \in \mathcal{X}$, the index $\bar{\ell}(\mathbf{x}) = \min\left\{\ell \in [M] : \ \hat{F}_{\hat{s}^\ell}(\hat{s}^\ell(\mathbf{x})) \geq 1 - \hat{\varepsilon}^\ell\right\}$ as the first moment where we do not reject. Then, the classification by EERO is exactly provided by the head that corresponds to $\bar{\ell}$, that is*

$$\hat{g}^{EERO}(\mathbf{x}) = \hat{g}^{\bar{\ell}(\mathbf{x})}(\mathbf{x}) \ . \tag{3.16}$$

The goal now is here to demonstrate that the choice of $\hat{\boldsymbol{\varepsilon}}$ provides a good compromise in terms of risk and computations budget. Recall that for all $\ell \in [M]$, we have

$$\hat{\varepsilon}^\ell = \frac{\pi^\ell \exp\left\{-\frac{\hat{R}^\ell + \hat{\mu}\hat{B}^\ell}{\beta}\right\}}{\sum_{j=1}^M \pi^j \exp\left\{-\frac{\hat{R}^j + \hat{\mu}\hat{B}^j}{\beta}\right\}} = c \cdot \pi^\ell \exp\left\{-\frac{\hat{R}^\ell + \hat{\mu}\hat{B}^\ell}{\beta}\right\} \ ,$$

where $c = \sum_{j=1}^M \pi^j \exp\left\{-\frac{\hat{R}^j + \hat{\mu}\hat{B}^j}{\beta}\right\}$ is the normalization constant. Then

$$\log\left(\frac{\hat{\varepsilon}^\ell}{\pi^\ell}\right) = \log(c) - \frac{\hat{R}^\ell + \hat{\mu}\hat{B}^\ell}{\beta} \ .$$

Rearranging terms we get

$$\hat{R}^\ell = \beta \log(c) - \beta \log\left(\frac{\hat{\varepsilon}^\ell}{\pi^\ell}\right) - \hat{\mu}\hat{B}^\ell \ .$$

Notice that the same relation holds true for the optimal head ℓ^* – the feasible head that minimizes the risk. Therefore, subtracting the two equality gives

$$\hat{R}^\ell = \hat{R}^{\ell^*} + \beta\left(\log\left(\frac{\hat{\varepsilon}^{\ell^*}}{\pi^{\ell^*}}\right) - \log\left(\frac{\hat{\varepsilon}^\ell}{\pi^\ell}\right)\right) + \hat{\mu}\left(\hat{B}^{\ell^*} - \hat{B}^\ell\right) \ .$$

Multiplying by $\hat{\varepsilon}^\ell$ and summing out over ℓ we get

$$\begin{aligned}\frac{1}{n_2} \sum_{(\mathbf{X},Y)\in\mathcal{D}_{n_2}} \mathbb{1}_{\{\hat{g}^{EERO}(\mathbf{X}) \neq Y\}} &= \sum_{\ell=1}^M \hat{\varepsilon}^\ell \hat{R}^\ell \\ &= \hat{R}^{\ell^*} + \beta\left(\log\left(\frac{\hat{\varepsilon}^{\ell^*}}{\pi^{\ell^*}}\right) - \sum_{\ell=1}^M \hat{\varepsilon}^\ell \log\left(\frac{\hat{\varepsilon}^\ell}{\pi^\ell}\right)\right) + \hat{\mu}\left(\hat{B}^{\ell^*} - \sum_{\ell=1}^M \hat{\varepsilon}^\ell \hat{B}^\ell\right) \ . \end{aligned} \tag{3.17}$$

Let us first consider the term $\hat{\mu}\left(\hat{B}^{\ell^*} - \sum_{\ell=1}^M \hat{\varepsilon}^\ell \hat{B}^\ell\right)$. Recalling the previously stated complementary condition $\hat{\mu}\left(\sum_{\ell=1}^M \hat{\varepsilon}^\ell(\hat{B}^\ell - \bar{B})\right) = 0$, we either have $\hat{\mu} = 0$ or $\sum_{\ell=1}^M \hat{\varepsilon}^\ell \hat{B}^\ell = \bar{B}$. In the latter case, the term

$$\hat{B}^{\ell^*} - \sum_{\ell=1}^M \hat{\varepsilon}^\ell \hat{B}^\ell = \hat{B}^{\ell^*} - \bar{B} \leq 0 \ ,$$

since $\hat{B}^{\ell^*} \leq \bar{B}$ by the fact that it corresponds to the budget of the ℓ-th head with a weight $\hat{\varepsilon}^* = 1$ (all other weights are 0). Therefore, in both cases $\hat{\mu}\left(\hat{B}^{\ell^*} - \sum_{\ell=1}^M \hat{\varepsilon}^\ell \hat{B}^\ell\right) \leq 0$. On the other, we have by the properties of the KL divergence,

$$\sum_{\ell=1}^M \hat{\varepsilon}^\ell \log\left(\frac{\hat{\varepsilon}^\ell}{\pi^\ell}\right) \leq 0 \ ,$$

and then Equation (3.17) *becomes*

$$\frac{1}{n_2} \sum_{(\mathbf{X},Y)\in\mathcal{D}_{n_2}} \mathbb{1}_{\{\hat{g}^{EERO}(\mathbf{X})\neq Y\}} \leq \hat{R}^{\ell^*} + \beta \log(1/\pi^{\ell^*}) \ ,$$

since $\log(\hat{\varepsilon}^{\ell^*}) = 0$ *and this ends the proof by taking the expectation from both side.*

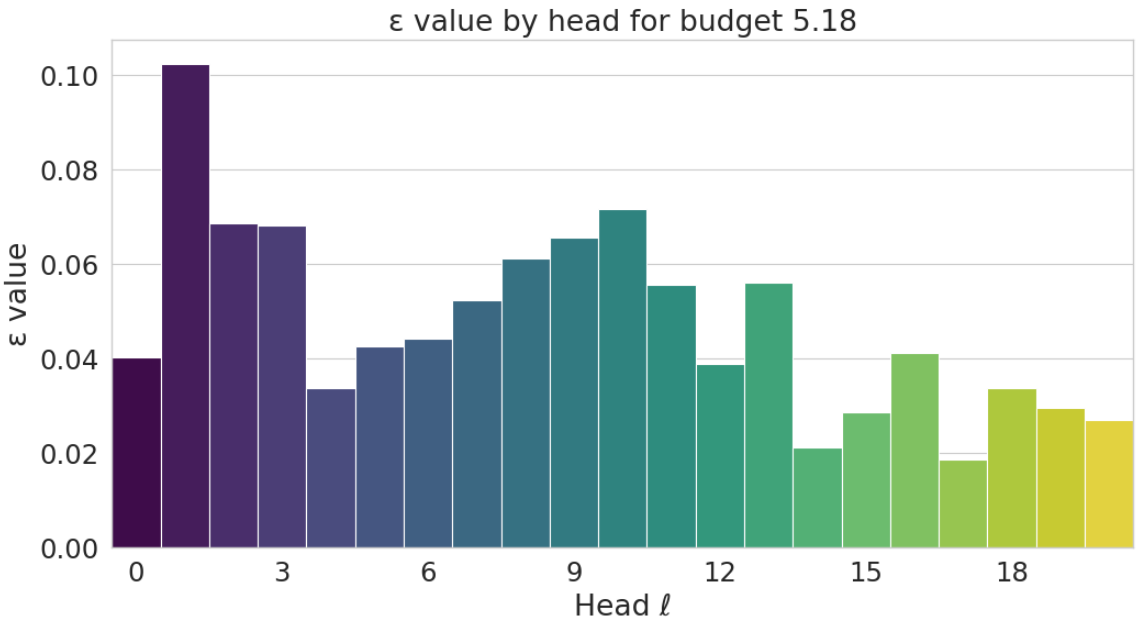
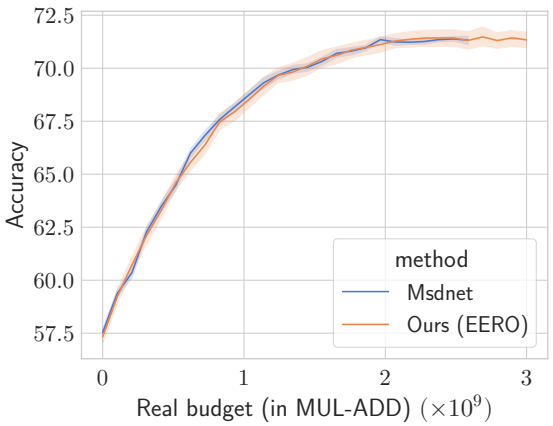
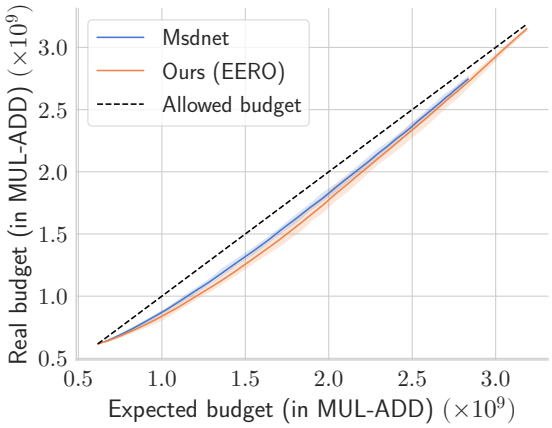


Figure 3.3: Value of the aggregation weights $\hat{\varepsilon}^{\ell}$ on an intermediate budget for the ConvNext model for EERO. Each bar represents an exit head ℓ.



(a) Accuracy *w.r.t.* the real budget based on MSDNet architecture – comparison of EERO methodology to the original MSDNet algorithm.



(b) Measured budget on MSDNet architecture – comparison of EERO methodology to the original MSDNet algorithm.

Figure 3.4: Subfigures (a) and (b) compare EERO with MSDNet's original method, focusing on accuracy and budget metrics. Error bars were made using bootstrap on data.

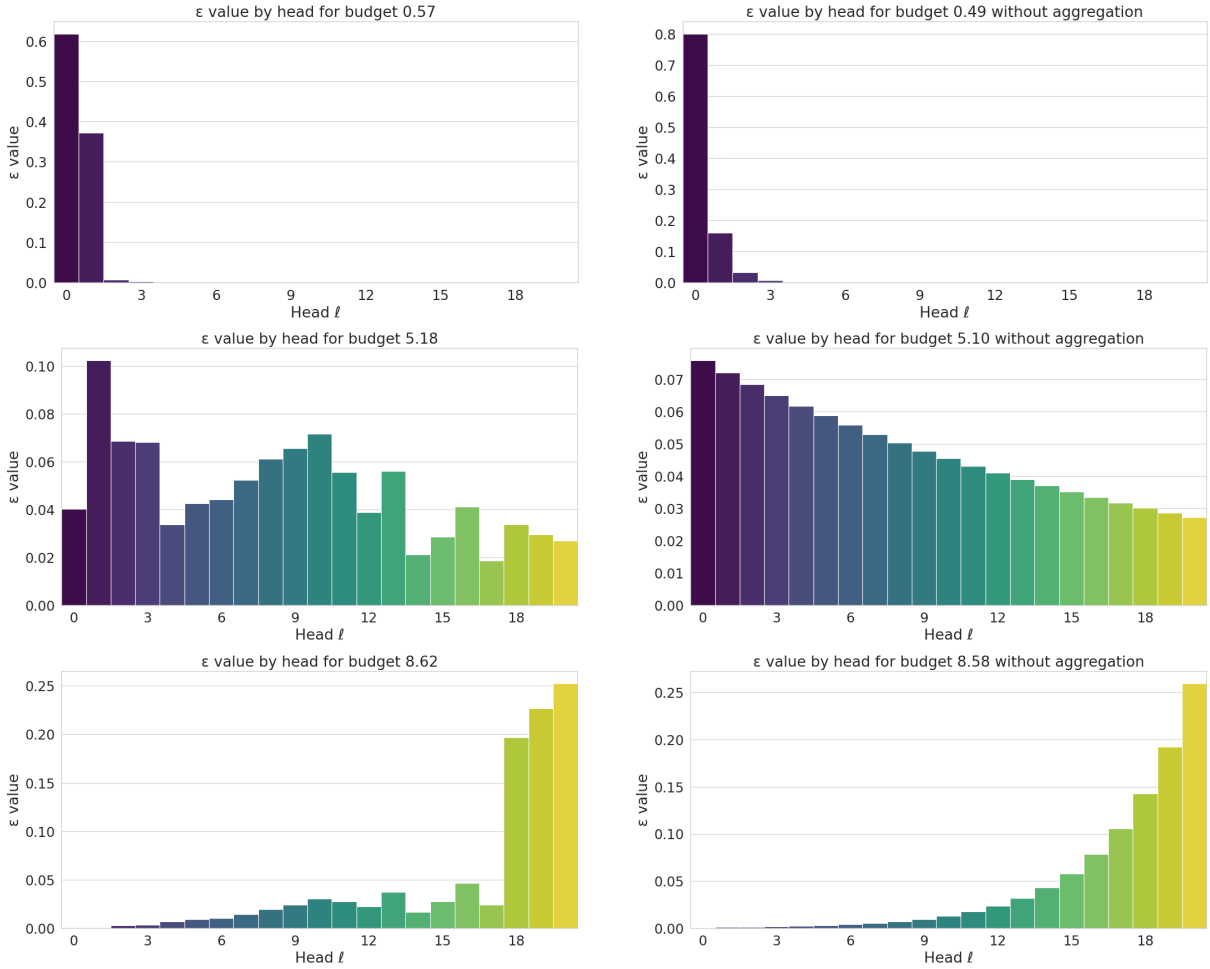


Figure 3.5: Value of the aggregation weights $\hat{\varepsilon}^{\ell}$ on three different budgets for the ConvNext model for EERO (left column) and with the MSDNet strategy (right column). Each bar represents an exit head ℓ. The lines corresponds to different budgets ranging from small (top), to medium (centre) and high (top).

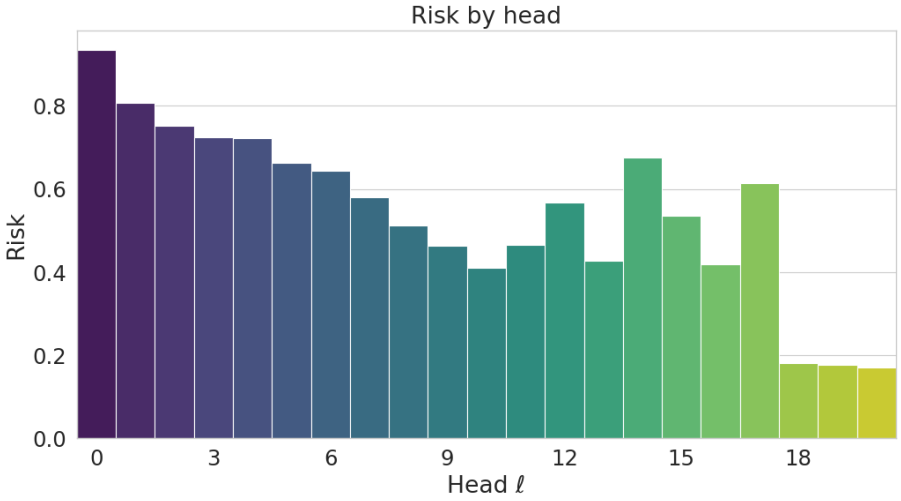


Figure 3.6: Risk $\hat{R}^{\ell}$ of each head on Convnext model

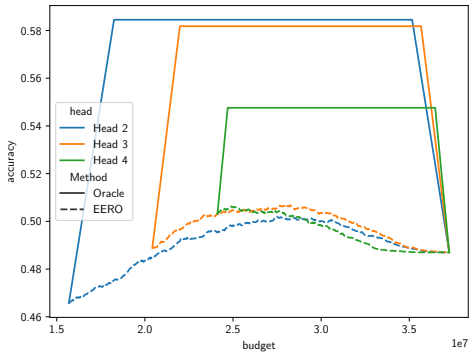


Figure 3.7: Accuracy versus computation budget for our multi-headed ResNet-18 (given in Figure 3.1). In this illustration, we consider only a two heads procedure by selecting a specific auxiliary head – 2, 3, or 4 – among the initial 7. Each curve shows how the accuracy evolves according to the probability of use of the auxiliary head. For each color, we display our algorithm as well as the oracle.

From Budget-Constrained Classification to Accuracy-Preserving LLM Inference

Chapter 3 presented EERO, a principled framework addressing budget-constrained early exit classification through the mathematical formalization of early exit as classification with reject option. This approach enables strict computational budget adherence with formal guarantees, making it particularly suitable for resource-limited environments such as edge computing, mobile devices, and real-time systems where computational violations are unacceptable.

The EERO framework demonstrated that exponential weights aggregation can provide provable bounds on computational resource consumption while maintaining competitive accuracy across computer vision tasks. This represents a significant advancement for deployment scenarios where **computational predictability takes precedence over achieving absolute peak performance**.

However, the landscape of early exit applications extends beyond resource-constrained environments. In many deployment scenarios—particularly those involving large language models—the primary concern shifts from strict budget adherence to **maintaining the highest possible accuracy while achieving computational savings**. This paradigm shift reflects the critical importance of accuracy in language understanding applications, where even small performance degradations can significantly impact user experience and system reliability.

Large language models present unique challenges that differentiate them from the computer vision scenarios addressed by EERO:

- **Sequential autoregressive generation**: Unlike image classification where early exit decisions affect single predictions, language models generate sequences token-by-token, where early exit decisions can impact the entire remaining generation.
- **High accuracy requirements**: Language understanding tasks often demand near-perfect accuracy, making accuracy preservation more critical than computational budget adherence.
- **Scaling benefits**: The computational costs of large language models are particularly severe, making efficiency improvements more valuable as model size increases.
- **Pre-trained model ecosystem**: The widespread adoption of pre-trained language models necessitates efficiency techniques that can be applied without expensive retraining.

These distinctive characteristics require a fundamentally different approach to early exit design—one that prioritizes accuracy preservation over strict budget constraints while leveraging the unique properties of transformer architectures and language understanding tasks.

The following chapter presents our second contribution: a specialized early exit framework for large language models that addresses these challenges through self-supervised training. This work complements EERO by providing accuracy-focused solutions for scenarios where maintaining baseline performance is paramount, demonstrating how early exit principles can be adapted to meet the diverse requirements of modern AI deployment scenarios.

Chapter 4

Accelerating LLMs with Early Exits

This chapter presents a simple and modular approach to accelerate inference in large language models (LLMs) by adding early exit heads at intermediate transformer layers. Each head is trained in a self-supervised manner to mimic the main model's predictions, allowing computation to stop early when a calibrated confidence threshold is reached. We evaluate several confidence metrics and show that entropy provides the most reliable separation between correct and incorrect predictions. Experiments on the Pythia model suite (70M to 2.8B parameters) demonstrate that our method significantly reduces inference cost while maintaining accuracy across multiple benchmarks. Notably, we observe that the benefits of early exits become more pronounced as model size increases, making this technique especially valuable for scaling LLMs to practical, resource-constrained, or real-time applications.

4.1 Introduction

Large language models (LLMs) have become central to advancing capabilities in natural language processing (NLP), delivering remarkable performance across a range of tasks. The trend towards scaling up these models correlates strongly with improved performance, understanding, and generality. This relationship has been formalized through empirical scaling laws, which demonstrate that model performance improves predictably with increased model size, dataset size, and compute budget [64, 53]. However, the computational cost associated with these larger models is substantial, often necessitating the use of powerful server infrastructure [109]. This not only limits local usability but also raises significant privacy concerns and requires considerable investment to scale in response to user demand. Solutions exist to reduce the computational demands of these models, but they often impact the model's performance by reducing its accuracy [158].

Despite their effectiveness, these models often operate inefficiently. The nature of language itself contributes to this inefficiency; namely, not all tokens generated during the inference process contribute equally to the overall meaning or require the same level of computational resources. Some tokens are inherently simpler and can be predicted with high confidence early in the computation process, while others, contributing more significantly to the context or meaning, may require deeper processing.

In response to these challenges, our objective is to develop a method that can be easily integrated into existing pre-trained models to enhance their inference speed without necessitating extensive retraining or modification. Our proposed solution focuses on the strategic placement of early exit "heads" [112, 128] within the transformer layers of an LLM. These heads are designed to terminate the inference process prematurely when a certain confidence threshold is met, based on the complexity and predictability of the token being processed.

Our contribution is threefold:

1. We conduct a comprehensive comparison of multiple training strategies for integrating early exits into pre-trained LLMs. By evaluating their impact on performance and efficiency, we aim to guide practitioners in selecting the most effective approach for their specific use case.

2. We analyze the behavior and scalability of early exit mechanisms as model size increases, providing insights into their viability and limitations in the context of very large language models.

3. We provide a lightweight and modular framework to facilitate the training, calibration, and evaluation of early exit models, making it easier to experiment with and deploy such techniques across different model architectures.

Having outlined our main contributions, we now situate our work within the broader context of related research.

4.2 Related Work

Transformers [133] scaled into today's LLM families such as BERT, GPT-3, PaLM, LaMDA, LLaMA and OPT [29, 9, 20, 129, 131, 154], powering vision [30], speech [100] and multimodal models. Their billion-parameter footprints, however, make every token generation costly. Static compression—quantisation, pruning and distillation [118, 124, 150, 34, 123, 3]—slashes model size but still expends identical compute on easy and hard inputs. Early-exit methods address this imbalance by attaching lightweight classifiers to intermediate layers and halting computation once a confidence criterion is met.

In computer vision, BranchyNet [128] and MSDNet [57] or the EERO methodology [132] established the two key components still used today: deeply supervised branches and an entropy-based exit rule. Transferring the idea to Transformers, DeeBERT and *The Right Tool* calibrated soft-max confidence to save up to 5× latency with negligible loss [146, 115]. FastBERT distilled the final head into earlier exits [87], PABEE demanded k consecutive agreeing predictions instead of a threshold [157], and BERxiT learned an explicit when-to-exit module while alternating fine-tune schedules [147]. Skip/SmartBERT added trainable gates that may bypass whole layers, combining skipping with exiting for 2–4× cheaper inference [80, 14].

Early exit for sequence-to-sequence generation is newer. Depth-Adaptive Transformers first applied exits to neural MT [33]; FREE and DEED refined token-level uncertainty estimates [139, 149]. Recent evidence shows exits remain effective inside 13 B-parameter LLaMA-2 and GPT-J [86], suggesting scalability to modern LLMs.

Our work inherits the plug-and-play nature of DeeBERT-style branches, borrows the self-supervised signal of FastBERT, and scale token-level exits to different architecture sizes. We train exit heads without extra data—using the model's own probabilities—then calibrate a single threshold on a held-out set à la conformal prediction [134]. Unlike layer-skipping approaches, the backbone weights remain frozen, guaranteeing monotonic accuracy with deeper computation.

Building on these foundations, we now detail our methodology for integrating early exits into large language models.

4.3 Methodology

This section details our methodology for integrating and utilizing early exits within large language models (LLMs) to enhance computational efficiency during inference. The approach is designed to be generalizable and, while we demonstrate its application using the Pythia suite, it is applicable to any multi-layered transformer model. This adaptability ensures that our methodology can be leveraged across a broad spectrum of modern LLMs, enhancing their usability without requiring significant modifications to their underlying architectures.

4.3.1 Definitions and Notation

We introduce here the main notations used throughout this chapter.

Vocabulary. Let $\mathcal{V}$ denote the vocabulary, a finite set of tokens:

$$\mathcal{V} = \{v_1, v_2, \ldots, v_{|\mathcal{V}|}\},$$

where $|\cdot|$ denotes cardinality of sets.

Dataset. We consider a dataset $\mathcal{D}$ of N examples, where each example is a sequence of tokens:

$$\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}, \quad \mathbf{x}^{(i)} = (x_1^{(i)}, \ldots, x_{L_i}^{(i)}), \quad x_j^{(i)} \in \mathcal{V}.$$

A subset $\mathcal{D}_{\text{cal}} \subset \mathcal{D}$ is reserved as a calibration set, the rest being used for training.

Each sequence $\mathbf{x}^{(i)}$ can have a variable length L_i. The total number of tokens in the dataset is $M = \sum_{i=1}^{N} L_i$, with $M > N$ in general. Similarly, the calibration set $\mathcal{D}_{\text{cal}}$ contains N_{cal} examples and a total of M_{cal} tokens.

Language model. A large language model (LLM) is a function f_θ parameterized by θ, mapping a sequence of tokens to a sequence of probability distributions over the vocabulary:

$$f_\theta : (x_1, \ldots, x_L) \mapsto (\mathbf{p}_1, \ldots, \mathbf{p}_L), \quad \mathbf{p}_t \in \Delta^{|\mathcal{V}|},$$

where $\Delta^{|\mathcal{V}|}$ is the probability simplex over $\mathcal{V}$.

Objective. Given an input sequence $\mathbf{x} = (x_1, \ldots, x_L) \in \mathcal{V}^L$, the target sequence is $\mathbf{y} = (y_1, \ldots, y_L) \in \mathcal{V}^L$ with $y_t = x_{t+1}$ (i.e., the next-token prediction task).

Early exit heads. We introduce K early exit heads, each defined as a function h_k (with its own parameters, but sharing the backbone with f_θ) that maps a sequence of L tokens to a sequence of probability distributions:

$$h_k : (x_1, \ldots, x_L) \mapsto (\mathbf{p}_{k,1}, \ldots, \mathbf{p}_{k,L}), \quad \mathbf{p}_{k,t} \in \Delta^{|\mathcal{V}|}.$$

All early exit heads share the same backbone, so most of their parameters are shared with the main model.

Confidence metric. A confidence metric is a function $c : \Delta^{|\mathcal{V}|} \to \mathbb{R}$ that assigns a real-valued confidence score to a probability vector (e.g., $c(\mathbf{p}) = \max_j p_j$).

Accuracy threshold ϵ. The parameter $\epsilon \in [0, 1]$ controls the minimal desired accuracy for early exit decisions. For each early exit head, a prediction is only output if its confidence metric exceeds a calibrated threshold corresponding to at least ϵ empirical accuracy on a held-out calibration set. Lowering ϵ increases speedup at the cost of potential accuracy loss, while higher ϵ enforces stricter correctness guarantees.

The above definitions give a precise, token-level view of the model outputs and their interaction with early-exit logic; they will serve as the foundation for the training objectives, calibration techniques, and inference algorithms described in the following sections.

4.3.2 Implementation Details

To enhance the inference efficiency of large language models, we incorporate early exit "heads" into a pre-existing model, in this instance, the Pythia suite [7]. These heads are implemented at regular intervals along the network. Structurally, each head is a simple multi-layer perceptron (MLP) identical to the final classification head of the model. Each of these head takes as input hidden features from a transformer block inside the model.

Placement of early–exit heads. Let the transformer backbone consist of L stacked blocks, indexed $\mathcal{I} = \{1, 2, \ldots, L\}$, and let K denote the desired number of early–exit heads. To distribute the heads uniformly while keeping the final classification layer untouched, we attach head $k \in \{1, \ldots, K\}$ to the block whose index is

$$\ell_k \;=\; \left\lfloor \frac{k}{K+1}\, L \right\rfloor, \qquad k = 1, \ldots, K. \tag{4.1}$$

Equation (4.1) simply divides the layer indices into $K + 1$ equal segments and selects the endpoint of each segment (rounded down to the nearest integer) as a branch location. In our experiments we set $K = 4$; hence the heads are inserted at $\ell_k \in \left\{ \lfloor \frac{L}{5} \rfloor,\ \lfloor \frac{2L}{5} \rfloor,\ \lfloor \frac{3L}{5} \rfloor,\ \lfloor \frac{4L}{5} \rfloor \right\}$.

For the implementation of these heads, we experimented with two initialization strategies: initializing the heads from scratch and copying the final classification head in order to fine-tune it. The difference between the two are analyzed in Section 4.4. With the architectural choices established, we next describe the data and objectives used to train the early exit heads.

4.3.3 Training data and objectives

Corpus. All auxiliary heads are trained on MiniPile [62], a 6-GB stratified subset of the original 825-GB ThePile-deduplicated [39] dataset that was employed for pre-training the Pythia backbone [7]. Using the same data distribution avoids the distribution-shift issues that often appear when auxiliary classifiers are fitted post-hoc.

Losses. For the whole predicted sequence $\hat{\mathbf{y}}$, we can compute different losses depending on the objectives. We define the cross-entropy loss and the Kullback–Leibler divergence between the predicted probability distribution $\hat{y}$ and the ground-truth labels y as follows:

$$\mathcal{L}_{\mathrm{CE}}(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{L} \sum_{t=1}^{L} \sum_{v \in \mathcal{V}} \mathbf{1}_{\{\mathbf{y}^{(t)}=v\}} \log(\hat{\mathbf{y}}_v^{(t)}),$$

$$\mathcal{L}_{\mathrm{KL}}(\hat{\mathbf{y}} || \mathbf{y}) = \frac{1}{L} \sum_{t=1}^{L} \sum_{v \in \mathcal{V}} \hat{\mathbf{y}}_v \log \frac{\hat{\mathbf{y}}_v}{\mathbf{y}_v}.$$

In the context of self-supervised training, we consider the Kullback-Leibler divergence where the target $\mathbf{y}$ is replaced by the output of the main model f_θ. Thus, the loss $\mathcal{L}_{\text{KL}}$ becomes:

$$\mathcal{L}_{\text{KL}}(\hat{\mathbf{y}}||f_\theta) = \frac{1}{L}\sum_{t=1}^{L}\sum_{v\in\mathcal{V}}\hat{\mathbf{y}}_v \log\frac{\hat{\mathbf{y}}_v}{f_\theta(v)}$$

The cross-entropy loss $\mathcal{L}_{\text{CE}}$ is used when considering a supervised training objective. It matches how the main model is trained and can be used to train the heads as well. The Kullback–Leibler divergence $\mathcal{L}_{\text{KL}}$ is used when considering a self-supervised training objective. It encourages the early exit heads to mimic the full probability distribution of the main model, which can be useful for improving the performance of the early exits. From this, we consider three training objectives obtained from the above building blocks:

- **Supervised** ($\mathcal{L}_{\text{sup}}$). Purely next-token cross-entropy against ground-truth labels, that is, $\mathcal{L}_{\text{sup}} = \mathcal{L}_{\text{CE}}$;
- **Self-supervised** ($\mathcal{L}_{\text{self}}$). KL divergence that encourages each head to mimic the teacher's full probability distribution, that is, $\mathcal{L}_{\text{self}} = \mathcal{L}_{\text{KL}}$;
- **Hybrid** ($\mathcal{L}_{\text{hyb}}$). The sum of the two losses above, weighted by coefficients $\alpha \in (0, 1)$, that is, $\mathcal{L}_{\text{hyb}} = \alpha\,\mathcal{L}_{\text{CE}} + (1-\alpha)\,\mathcal{L}_{\text{KL}}$.

As reported in Section 4.4, the self-supervised objective ($\mathcal{L}_{\text{self}}$) yields the most faithful approximation of the teacher's behavior while preserving calibration, and therefore constitutes our default choice for all subsequent experiments.

4.3.4 Calibration and Inference

After training the early exit heads, the next crucial step involves calibrating and using these heads during model inference. This process is divided into two main stages: calibration of the confidence thresholds and the application of these thresholds during inference.

Calibration of Confidence Thresholds

In our approach, we evaluated three different confidence metrics for calibrating early exit thresholds: (1) maximum probability, (2) entropy of the predicted distribution, and (3) the difference between the top two probabilities ("breaking ties"). To determine which metric best separates correct from incorrect predictions, we analyzed their ability to discriminate between right and wrong outputs using ROC (Receiver Operating Characteristic) curves.

After training, we computed the ROC curves for each metric by plotting the true positive rate against the false positive rate as the threshold varies, using the predictions from the early exit heads. This analysis was performed across six Pythia models, ranging from 70M to 2.8B parameters. The results are summarized in Figure 4.1, which displays the ROC curves for all metrics and models.

Our findings consistently show that entropy outperforms the other metrics in every case, achieving a higher area under the curve (AUC) regardless of model size. This indicates that entropy provides a more reliable separation between correct and incorrect predictions, making it the most effective metric for threshold calibration in our early exit framework. Furthermore, we observe that the AUC of the entropy metric tends to increase with model size, suggesting that early exit mechanisms become even more effective as the underlying model grows larger.

A detailed visualization of the entropy distribution for correct and incorrect outputs is also available in the appendix (see Figure 4.5), illustrating the ability of entropy to effectively separate reliable predictions from errors.

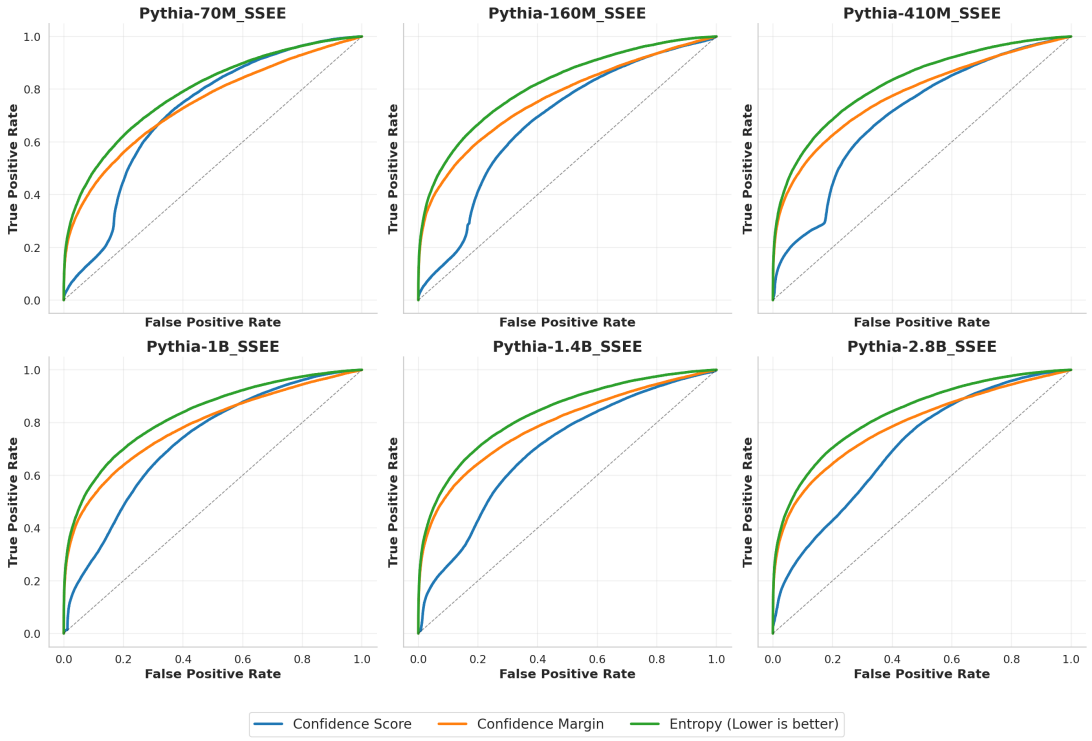


Figure 4.1: ROC curves for three confidence metrics (maximum probability, entropy, breaking ties) across six Pythia models (70M to 2.8B parameters). Entropy consistently achieves the highest AUC, indicating superior discrimination between correct and incorrect predictions.

To calibrate the confidence thresholds, we perform a full epoch over our calibration dataset. For each example in the calibration set and for each early exit head, we compute the head output for every token in the sequence, then apply the confidence metric c to each token output, and store all these scores in a global vector.

Formally, for each sequence $\mathbf{x}^{(i)}$ of length L_i in $\mathcal{D}_{\text{cal}}$, we obtain the outputs $h_k(\mathbf{x}^{(i)}) = (\mathbf{p}_{k,1}^{(i)}, \ldots, \mathbf{p}_{k,L_i}^{(i)})$. We then apply c to each $\mathbf{p}_{k,t}^{(i)}$ for $t = 1, \ldots, L_i$ and concatenate all results to obtain:

$$\mathbf{c}_k = \left(c(\mathbf{p}_{k,1}^{(1)}), \ldots, c(\mathbf{p}_{k,L_1}^{(1)}),\ c(\mathbf{p}_{k,1}^{(2)}), \ldots, c(\mathbf{p}_{k,L_2}^{(2)}),\ \ldots,\ c(\mathbf{p}_{k,1}^{(N_{\text{cal}})}), \ldots, c(\mathbf{p}_{k,L_{N_{\text{cal}}}}^{(N_{\text{cal}})})\right) \in \mathbb{R}^{M_{\text{cal}}} \tag{4.2}$$

Here, $M_{\text{cal}} = \sum_{i=1}^{N_{\text{cal}}} L_i$ is the total number of tokens in the calibration dataset.

Correspondingly, let $\mathbf{t}_k$ be a binary vector of length M_{cal} where each element corresponds to the correctness of the prediction associated with the respective element in $\mathbf{c}_k$. Specifically, $t_{k,j}$ is 1 if the j^{th} prediction at head k is the same as the j^{th} prediction of the underlying model $f_\theta(\mathbf{x})$, otherwise 0. More formally, we can write:

$$t_{k,j} = \mathbb{1}\arg\max \mathbf{p}_{k,j} = \arg\max \mathbf{p}_{\theta,j} = \begin{cases} 1 & \text{if } \arg\max(\mathbf{p}_{k,j}) = \arg\max(\mathbf{p}_{\theta,j}) \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

To determine the confidence threshold for each early exit head, we utilize the calibration set to empirically estimate the relationship between the confidence metric and prediction correctness. Specifically, for each head k, we sort the calibration metric values in ascending order. Given a user-specified confidence level $\epsilon \in [0, 1]$, which represents the minimum desired proportion of correct predictions above the threshold, we identify the smallest metric value such that the proportion of correct predictions among all samples with higher (or equal) metric values is at least ϵ. This procedure ensures that, during inference, predictions made with a confidence metric exceeding the threshold are correct with probability at least

ϵ, thereby providing a principled trade-off between computational efficiency and predictive accuracy. Formally, the threshold for each head is defined as follows.

Recall that $\mathbf{c}_k$ and $\mathbf{t}_k$ denote, respectively, the vectors of confidence metric values and correctness indicators for head k, as defined in equations (4.2) and (4.3). To proceed, we jointly sort these vectors in ascending order according to the values in $\mathbf{c}_k$, resulting in ordered sequences $\mathbf{c}_k = (c_{k,1}, c_{k,2}, \ldots, c_{k,M_{\text{cal}}})$ and $\mathbf{t}_k = (t_{k,1}, t_{k,2}, \ldots, t_{k,M_{\text{cal}}})$ such that $c_{k,1} \leq c_{k,2} \leq \ldots \leq c_{k,M_{\text{cal}}}$.

We then define $\hat{j}$ as the smallest index for which the proportion of correct predictions among all samples with confidence at least $c_{k,\hat{j}}$ meets or exceeds the target confidence level ϵ:

$$\frac{\sum_{i=\hat{j}}^{M_{\text{cal}}} t_{k,i}}{M_{\text{cal}} - \hat{j} + 1} \geq \epsilon.$$

Then, let the threshold τ_k be defined as:

$$\tau_k = c_{k,\hat{j}}.$$

In other words, τ_k is the value of the confidence metric at the index $\hat{j}$, where $\hat{j}$ is the smallest index such that the proportion of correct prediction in the remaining samples is at least ϵ.

Inference Process

After establishing the confidence thresholds for each early exit head through the calibration process, the model is then ready to utilize these thresholds during the inference phase to efficiently process new inputs.

During inference, each input $\mathbf{x}$ is sequentially processed through the model's layers, with the possibility of early termination at any of the early exit heads h_k. For each head $k \in \{1, 2, \ldots, K\}$, we compute:

$$\mathbf{p}_k = h_k(\mathbf{x}),$$
$$c_k = c(\mathbf{p}_k).$$

We then return $\mathbf{p}_k$ as output from the model if:

$$c_k \geq \tau_k.$$

If no head satisfies this inequality, we return $\mathbf{p}_\theta$.

This calibrated and threshold-driven early exit mechanism allows the model to balance efficiency with accuracy, ensuring that resource-intensive computations are only performed when necessary.

4.4 Experiments and Results

In this section, we present the results of our experiments[1] on training and evaluating the Pythia suite models with early exits placed at regular interval along the network. We first describe the training process and then present the results of our training experiments.

In the second part of this section, we focus on the inference phase, where we investigate the performance degradation of the model as we increase the speedup (or decrease epsilon)

[1]All computations are run on a server with an Intel(R) Xeon(R) Gold 5120 CPU and a Tesla V100 GPU with 32GB of Vram and 64GB of RAM. Code used in experiments to train and evaluate can be found under : https://anonymous.4open.science/r/BranchyLLM-B870

by exiting earlier in the network. We provide a comprehensive evaluation of our approach, including various evaluation scores and speedup metrics, to demonstrate the effectiveness of our method in achieving speedup while maintaining acceptable performance. Throughout our experiments, we use the Pythia suite as a representative example, as it allows us to scale the model size according to our hardware capabilities and to approximate the behavior of our method on larger models. However, our approach is theoretically applicable to any language model architecture.

4.4.1 Training

In the training phase, we systematically compared the three loss functions described in Section 4.3.2: supervised (cross-entropy), self-supervised (KL divergence to the main model's output), and a hybrid of both. Our primary goal was to select a training objective that enables early exit heads to best approximate the main model's predictions while also providing meaningful uncertainty estimates.

We found that the self-supervised loss, which encourages each early exit head to mimic the output distribution of the main model, consistently led to the best alignment with the main model's predictions. This loss not only matches the output probabilities but also preserves the calibration properties necessary for reliable early exits.

Additionally, we experimented with two initialization strategies for the early exit heads: (1) copying the weights from the main model's final classification head (lm_head), and (2) random initialization. When the early exit heads are initialized by copying the lm_head, they start with a lower loss and initially perform better. However, as training progresses, the randomly initialized heads quickly overtake the copied ones in both loss and performance.

A key observation is that heads copied from the lm_head tend to be overconfident, even when making incorrect predictions. This results in low entropy outputs and a lack of meaningful uncertainty, which is detrimental for early exit decisions. In contrast, randomly initialized heads, when trained with the self-supervised loss, develop a better notion of uncertainty, producing higher entropy outputs when unsure. This property is crucial for effective early exit mechanisms, as it allows the confidence metric to reliably distinguish between correct and incorrect predictions.

For our training experiments, we launch training with the self-supervised loss described above on all Pythia models from 70M to 2.8B parameters. We use a learning rate of 5×10^{-5} and train on 500,000 examples sampled from the MiniPile dataset. Four early exit heads are added at the locations described in the Methodology section.

Finally, a detailed analysis of the top-k accuracy of each intermediate head after training is provided in the appendix (see Section 4.7, Figure 4.4), confirming that intermediate heads reach competitive accuracy levels and make early exits reliable in practice.

4.4.2 Inference

For the inference evaluation, each model is evaluated with different values of the confidence threshold parameter ϵ, ranging from 0.5 to 1.0 in increments of 0.05. This systematic sweep allows us to analyze the trade-off between computational savings and predictive accuracy.

We follow the same set of benchmarks as the original Pythia paper: WSC, Winogrande, SciQ, PIQA, LogiQA, LAMBADA_OpenAI, ARC Easy, and ARC Challenge. For each benchmark and each value of ϵ, we report both the benchmark score and the mean output layer, translated to a speedup percentage.

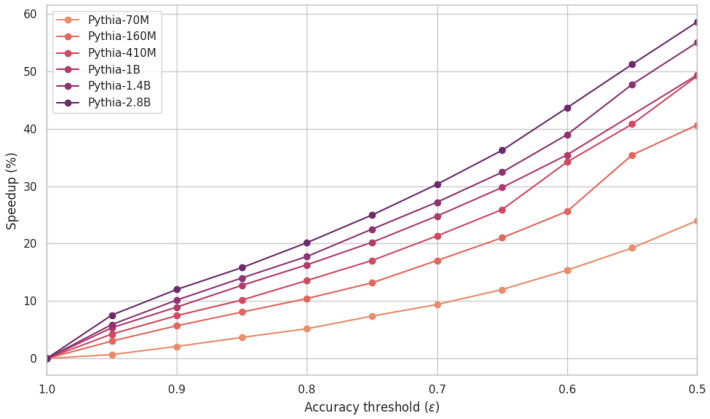


Figure 4.2: Speedup as a function of the confidence threshold ϵ for different Pythia model sizes. Larger models achieve greater speedup for the same threshold.

The results of the benchmark on Pythia-2.8B are shown in Tables 4.1 and 4.13. The results for the other model sizes will be provided in the appendix. The evolution of benchmark scores as a function of ϵ for all model sizes is illustrated in Figure 4.3.

Our results show that for some benchmarks, such as Winogrande and WSC, the reduction in computation has little to no effect on accuracy, even with aggressive early exiting. However, most benchmarks experience a drop in performance when using very aggressive thresholds (lower ϵ). Despite this, for moderate values of ϵ, the benchmark scores remain close to those of the main model, while achieving speedups between 10% and 20% depending on the model size.

Interestingly, we observe that smaller models in the Pythia suite, such as the 70M and 160M parameter variants, can even show improvements on certain benchmarks when early exits are used aggressively. This suggests that early exit mechanisms may help regularize smaller models or mitigate overfitting in some cases.

Beyond benchmark results, we also observe that the speedup achieved by early exit increases with the size of the model. Larger models benefit more from early exit, with greater computational savings for the same threshold values. This effect may be explained by several factors: (1) the relative size of the early exit heads becomes less significant as model size increases, and (2) the intermediate features in larger models may provide better representations, enabling more accurate early predictions. This trend is illustrated in Figure 4.2, which shows the speedup as a function of the threshold for all model sizes.

Finally, a fine-grained analysis of per-token top-1 accuracy for each intermediate head is provided in the appendix (see Figure 4.6), showing that some tokens are predicted very early with high reliability, which further supports the benefit of early exits for 'easy' tokens.

All detailed tables of benchmark scores and speedup for each model and threshold are provided in the appendix. While our results demonstrate the effectiveness of early exits, it is important to consider the limitations and broader impacts of this approach.

4.5 Limitations and potential impacts

While our early exit approach enables acceleration of large language models, the speedup gains can remain modest when aiming to preserve the original LLM's accuracy. In other words, if the goal is to match the baseline model's performance, the computational savings are limited. However, we observe that for certain benchmarks, early exits do not lead to any noticeable degradation in accuracy, even with significant acceleration. This suggests

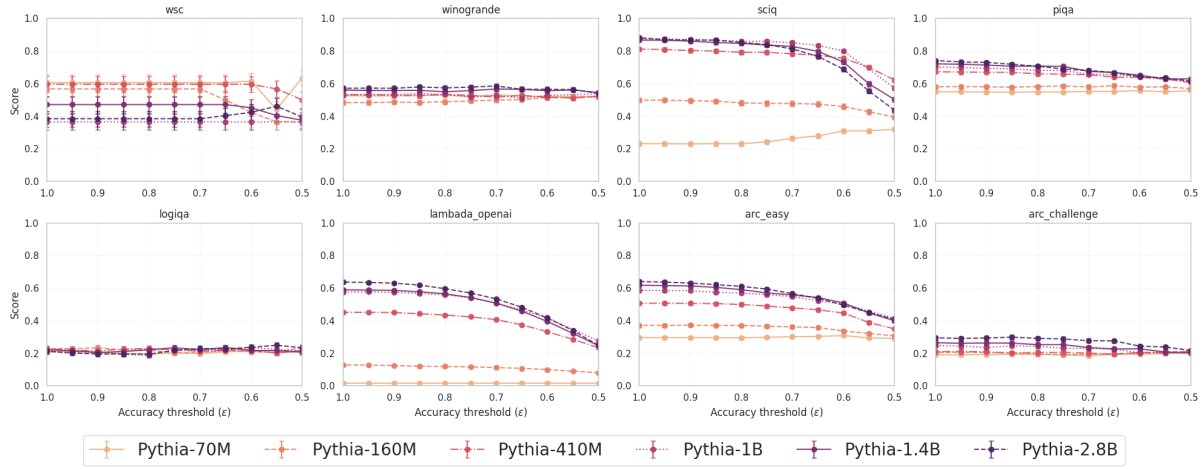


Figure 4.3: Benchmark scores for the Pythia suite of models (70M to 2.8B) as a function of the confidence threshold ϵ.

Table 4.1: Benchmarks results for Pythia-2.8B. Each line corresponds to a value of ϵ ranging from 0.5 to 1.0 in steps of 0.05 (top to bottom), with the bottom line being the baseline ($\epsilon = 1$). More benchmark available in the appendix 4.13.

ϵ	Speedup	wsc	winogrande	sciq	piqa	logiqa
0.50	58.6%	0.394 ± 0.048	0.541 ± 0.014	0.436 ± 0.016	0.615 ± 0.011	0.233 ± 0.017
0.55	51.2%	0.462 ± 0.049	0.564 ± 0.014	0.554 ± 0.016	0.635 ± 0.011	0.249 ± 0.017
0.60	43.7%	0.423 ± 0.049	0.565 ± 0.014	0.688 ± 0.015	0.643 ± 0.011	0.237 ± 0.017
0.65	36.2%	0.404 ± 0.048	0.565 ± 0.014	0.767 ± 0.013	0.668 ± 0.011	0.226 ± 0.016
0.70	30.3%	0.385 ± 0.048	0.586 ± 0.014	0.813 ± 0.012	0.681 ± 0.011	0.229 ± 0.016
0.75	25.0%	0.385 ± 0.048	0.579 ± 0.014	0.839 ± 0.012	0.690 ± 0.011	0.217 ± 0.016
0.80	20.2%	0.385 ± 0.048	0.574 ± 0.014	0.853 ± 0.011	0.707 ± 0.011	0.195 ± 0.016
0.85	15.8%	0.385 ± 0.048	0.579 ± 0.014	0.867 ± 0.011	0.719 ± 0.010	0.195 ± 0.016
0.90	12.0%	0.385 ± 0.048	0.571 ± 0.014	0.867 ± 0.011	0.730 ± 0.010	0.195 ± 0.016
0.95	7.6%	0.385 ± 0.048	0.572 ± 0.014	0.870 ± 0.011	0.733 ± 0.010	0.201 ± 0.016
1.00	0.0%	0.385 ± 0.048	0.571 ± 0.014	0.882 ± 0.010	0.742 ± 0.010	0.212 ± 0.016

that, for specific tasks or use cases, it is possible to benefit from substantial speedups without sacrificing performance, opening the door to targeted applications where efficiency is critical.

Another important trend highlighted by our experiments is that the speedup obtained through early exits tends to increase with model size. Larger models appear to benefit more from this mechanism, both in terms of computational savings and in the stability of their predictions under early exit. Nevertheless, due to our limited computational resources, we were unable to train and evaluate early exit models on the largest architectures available. Confirming and further exploring this trend would require access to greater computing power and remains an important direction for future research.

4.6 Conclusion

In this chapter, we introduced a simple and modular early exit mechanism for large language models, allowing inference to terminate early based on calibrated confidence metrics. Our approach is easy to integrate into existing transformer architectures and does not require retraining the backbone model. Through extensive experiments on the Pythia suite, we demonstrated that early exits can provide significant inference speedups, especially for larger models, while maintaining high accuracy on several benchmarks.

Notably, for some tasks, acceleration can be achieved with virtually no loss in performance, highlighting the practical value of this method for targeted applications.

However, our results also show that when strict accuracy preservation is required, the achievable speedup may remain modest. The benefits of early exit are not uniform across all tasks, and the effectiveness of the approach depends on the nature of the benchmark and the model size. Furthermore, due to limited computational resources, we were unable to train and evaluate early exit mechanisms on the largest available models. Confirming the observed trends and fully exploiting the potential of early exits will require further research with greater computational capacity.

Overall, our findings suggest that exploiting the natural variability in token difficulty is a promising direction for accelerating large language models. We encourage the community to build upon this work, exploring new strategies and applications to enable efficient and scalable deployment of LLMs in real-world, resource-constrained environments.

4.7 Appendix

4.7.1 Full benchmark results

This appendix presents the complete benchmark results for all Pythia models evaluated in this work. The following tables report the benchmark scores for each model size (from 70M to 2.8B parameters) on a range of tasks (WSC, Winogrande, SciQ, PIQA, LogiQA, LAMBADA_OpenAI, ARC Easy, ARC Challenge) for different values of the confidence threshold ϵ. For each configuration, we also indicate the corresponding speedup achieved by the early exit mechanism. These results allow a detailed comparison of the trade-off between computational savings and accuracy across models and tasks.

Table 4.2: Results for Pythia-70M (Part 1/2)

ϵ	Speedup	wsc	winogrande	sciq	piqa	logiqa
0.50	24.0%	0.635 ± 0.047	0.522 ± 0.014	0.320 ± 0.015	0.557 ± 0.012	0.206 ± 0.016
0.55	19.2%	0.442 ± 0.049	0.522 ± 0.014	0.310 ± 0.015	0.550 ± 0.012	0.204 ± 0.016
0.60	15.4%	0.615 ± 0.048	0.520 ± 0.014	0.310 ± 0.015	0.557 ± 0.012	0.210 ± 0.016
0.65	12.0%	0.606 ± 0.048	0.525 ± 0.014	0.279 ± 0.014	0.552 ± 0.012	0.209 ± 0.016
0.70	9.4%	0.606 ± 0.048	0.530 ± 0.014	0.264 ± 0.014	0.552 ± 0.012	0.198 ± 0.016
0.75	7.4%	0.606 ± 0.048	0.521 ± 0.014	0.242 ± 0.014	0.548 ± 0.012	0.201 ± 0.016
0.80	5.2%	0.606 ± 0.048	0.528 ± 0.014	0.231 ± 0.013	0.548 ± 0.012	0.195 ± 0.016
0.85	3.7%	0.606 ± 0.048	0.529 ± 0.014	0.231 ± 0.013	0.548 ± 0.012	0.198 ± 0.016
0.90	2.1%	0.606 ± 0.048	0.524 ± 0.014	0.230 ± 0.013	0.548 ± 0.012	0.203 ± 0.016
0.95	0.7%	0.606 ± 0.048	0.525 ± 0.014	0.231 ± 0.013	0.550 ± 0.012	0.212 ± 0.016
1.00	0.0%	0.606 ± 0.048	0.526 ± 0.014	0.231 ± 0.013	0.550 ± 0.012	0.210 ± 0.016

Table 4.3: Results for Pythia-70M (Part 2/2)

ϵ	Speedup	lambada_openai	arc_easy	arc_challenge
0.50	24.0%	0.015 ± 0.002	0.291 ± 0.009	0.203 ± 0.012
0.55	19.2%	0.016 ± 0.002	0.294 ± 0.009	0.197 ± 0.012
0.60	15.4%	0.016 ± 0.002	0.308 ± 0.009	0.194 ± 0.012
0.65	12.0%	0.016 ± 0.002	0.303 ± 0.009	0.193 ± 0.012
0.70	9.4%	0.016 ± 0.002	0.302 ± 0.009	0.183 ± 0.011
0.75	7.4%	0.016 ± 0.002	0.298 ± 0.009	0.191 ± 0.011
0.80	5.2%	0.016 ± 0.002	0.295 ± 0.009	0.193 ± 0.012
0.85	3.7%	0.016 ± 0.002	0.295 ± 0.009	0.193 ± 0.012
0.90	2.1%	0.016 ± 0.002	0.296 ± 0.009	0.191 ± 0.011
0.95	0.7%	0.016 ± 0.002	0.296 ± 0.009	0.190 ± 0.011
1.00	0.0%	0.016 ± 0.002	0.296 ± 0.009	0.190 ± 0.011

4.7.2 Post-training analysis

To further analyze the quality of the early exit heads after training, we report the top-1, top-3, top-5, and top-10 accuracy of each head on the Pythia-2.8B model. Figure 4.4 shows that the intermediate heads, post-training, achieve competitive accuracy compared to the final head. This demonstrates that our post-training procedure enables early exit heads to provide strong predictions, making early exits practical and reliable in this setting.

4.7.3 Entropy distribution analysis

To further illustrate the discriminative power of the entropy metric, Figure 4.5 presents a violin plot of the entropy values computed on the calibration set. This plot shows the

Table 4.4: Results for Pythia-160M (Part 1/2)

ϵ	Speedup	wsc	winogrande	sciq	piqa	logiqa
0.50	40.7%	0.365 ± 0.047	0.519 ± 0.014	0.394 ± 0.015	0.569 ± 0.012	0.237 ± 0.017
0.55	35.4%	0.365 ± 0.047	0.513 ± 0.014	0.427 ± 0.016	0.581 ± 0.012	0.204 ± 0.016
0.60	25.6%	0.423 ± 0.049	0.515 ± 0.014	0.459 ± 0.016	0.578 ± 0.012	0.210 ± 0.016
0.65	21.0%	0.500 ± 0.049	0.501 ± 0.014	0.474 ± 0.016	0.587 ± 0.011	0.217 ± 0.016
0.70	17.1%	0.567 ± 0.049	0.499 ± 0.014	0.478 ± 0.016	0.579 ± 0.012	0.204 ± 0.016
0.75	13.2%	0.567 ± 0.049	0.493 ± 0.014	0.479 ± 0.016	0.585 ± 0.011	0.203 ± 0.016
0.80	10.4%	0.567 ± 0.049	0.489 ± 0.014	0.482 ± 0.016	0.582 ± 0.012	0.218 ± 0.016
0.85	8.1%	0.567 ± 0.049	0.485 ± 0.014	0.492 ± 0.016	0.578 ± 0.012	0.220 ± 0.016
0.90	5.7%	0.567 ± 0.049	0.486 ± 0.014	0.495 ± 0.016	0.581 ± 0.012	0.233 ± 0.017
0.95	3.0%	0.567 ± 0.049	0.483 ± 0.014	0.499 ± 0.016	0.582 ± 0.012	0.227 ± 0.016
1.00	0.0%	0.567 ± 0.049	0.483 ± 0.014	0.498 ± 0.016	0.581 ± 0.012	0.229 ± 0.016

Table 4.5: Results for Pythia-160M (Part 2/2)

ϵ	Speedup	lambada_openai	arc_easy	arc_challenge
0.50	40.7%	0.081 ± 0.004	0.307 ± 0.009	0.209 ± 0.012
0.55	35.4%	0.090 ± 0.004	0.321 ± 0.010	0.209 ± 0.012
0.60	25.6%	0.100 ± 0.004	0.337 ± 0.010	0.201 ± 0.012
0.65	21.0%	0.107 ± 0.004	0.357 ± 0.010	0.197 ± 0.012
0.70	17.1%	0.112 ± 0.004	0.362 ± 0.010	0.194 ± 0.012
0.75	13.2%	0.116 ± 0.004	0.366 ± 0.010	0.190 ± 0.011
0.80	10.4%	0.118 ± 0.004	0.371 ± 0.010	0.195 ± 0.012
0.85	8.1%	0.120 ± 0.005	0.371 ± 0.010	0.203 ± 0.012
0.90	5.7%	0.124 ± 0.005	0.372 ± 0.010	0.204 ± 0.012
0.95	3.0%	0.127 ± 0.005	0.371 ± 0.010	0.206 ± 0.012
1.00	0.0%	0.127 ± 0.005	0.371 ± 0.010	0.206 ± 0.012

Table 4.6: Results for Pythia-410M (Part 1/2)

ϵ	Speedup	wsc	winogrande	sciq	piqa	logiqa
0.50	49.2%	0.500 ± 0.049	0.522 ± 0.014	0.622 ± 0.015	0.606 ± 0.011	0.217 ± 0.016
0.55	40.8%	0.567 ± 0.049	0.511 ± 0.014	0.699 ± 0.015	0.628 ± 0.011	0.201 ± 0.016
0.60	34.2%	0.596 ± 0.048	0.516 ± 0.014	0.756 ± 0.014	0.639 ± 0.011	0.220 ± 0.016
0.65	25.9%	0.596 ± 0.048	0.527 ± 0.014	0.774 ± 0.013	0.639 ± 0.011	0.218 ± 0.016
0.70	21.3%	0.596 ± 0.048	0.517 ± 0.014	0.783 ± 0.013	0.655 ± 0.011	0.212 ± 0.016
0.75	17.1%	0.596 ± 0.048	0.528 ± 0.014	0.792 ± 0.013	0.658 ± 0.011	0.221 ± 0.016
0.80	13.6%	0.596 ± 0.048	0.534 ± 0.014	0.792 ± 0.013	0.660 ± 0.011	0.229 ± 0.016
0.85	10.2%	0.596 ± 0.048	0.526 ± 0.014	0.800 ± 0.013	0.669 ± 0.011	0.226 ± 0.016
0.90	7.5%	0.596 ± 0.048	0.529 ± 0.014	0.803 ± 0.013	0.669 ± 0.011	0.214 ± 0.016
0.95	4.3%	0.596 ± 0.048	0.532 ± 0.014	0.809 ± 0.012	0.671 ± 0.011	0.220 ± 0.016
1.00	0.0%	0.596 ± 0.048	0.530 ± 0.014	0.812 ± 0.012	0.672 ± 0.011	0.226 ± 0.016

Table 4.7: Results for Pythia-410M (Part 2/2)

ϵ	Speedup	lambada_openai	arc_easy	arc_challenge
0.50	49.2%	0.234 ± 0.006	0.349 ± 0.010	0.212 ± 0.012
0.55	40.8%	0.285 ± 0.006	0.388 ± 0.010	0.202 ± 0.012
0.60	34.2%	0.332 ± 0.007	0.446 ± 0.010	0.205 ± 0.012
0.65	25.9%	0.375 ± 0.007	0.467 ± 0.010	0.194 ± 0.012
0.70	21.3%	0.406 ± 0.007	0.478 ± 0.010	0.201 ± 0.012
0.75	17.1%	0.423 ± 0.007	0.490 ± 0.010	0.205 ± 0.012
0.80	13.6%	0.434 ± 0.007	0.499 ± 0.010	0.205 ± 0.012
0.85	10.2%	0.442 ± 0.007	0.505 ± 0.010	0.201 ± 0.012
0.90	7.5%	0.449 ± 0.007	0.506 ± 0.010	0.208 ± 0.012
0.95	4.3%	0.451 ± 0.007	0.507 ± 0.010	0.211 ± 0.012
1.00	0.0%	0.451 ± 0.007	0.507 ± 0.010	0.209 ± 0.012

Table 4.8: Results for Pythia-1B (Part 1/2)

ϵ	Speedup	wsc	winogrande	sciq	piqa	logiqa
0.50	49.4%	0.365 ± 0.047	0.537 ± 0.014	0.573 ± 0.016	0.613 ± 0.011	0.214 ± 0.016
0.60	35.5%	0.365 ± 0.047	0.527 ± 0.014	0.800 ± 0.013	0.643 ± 0.011	0.235 ± 0.017
0.65	29.8%	0.365 ± 0.047	0.511 ± 0.014	0.834 ± 0.012	0.655 ± 0.011	0.221 ± 0.016
0.70	24.8%	0.365 ± 0.047	0.527 ± 0.014	0.851 ± 0.011	0.659 ± 0.011	0.220 ± 0.016
0.75	20.2%	0.365 ± 0.047	0.518 ± 0.014	0.859 ± 0.011	0.681 ± 0.011	0.223 ± 0.016
0.80	16.3%	0.365 ± 0.047	0.534 ± 0.014	0.859 ± 0.011	0.681 ± 0.011	0.186 ± 0.015
0.85	12.7%	0.365 ± 0.047	0.538 ± 0.014	0.867 ± 0.011	0.691 ± 0.011	0.194 ± 0.015
0.90	8.9%	0.365 ± 0.047	0.534 ± 0.014	0.871 ± 0.011	0.694 ± 0.011	0.210 ± 0.016
0.95	5.4%	0.365 ± 0.047	0.534 ± 0.014	0.874 ± 0.010	0.699 ± 0.011	0.206 ± 0.016
1.00	0.0%	0.365 ± 0.047	0.532 ± 0.014	0.876 ± 0.010	0.702 ± 0.011	0.209 ± 0.016

Table 4.9: Results for Pythia-1B (Part 2/2)

ϵ	Speedup	lambada_openai	arc_easy	arc_challenge
0.50	49.4%	0.276 ± 0.006	0.414 ± 0.010	0.206 ± 0.012
0.60	35.5%	0.417 ± 0.007	0.503 ± 0.010	0.202 ± 0.012
0.65	29.8%	0.465 ± 0.007	0.522 ± 0.010	0.224 ± 0.012
0.70	24.8%	0.508 ± 0.007	0.547 ± 0.010	0.230 ± 0.012
0.75	20.2%	0.541 ± 0.007	0.559 ± 0.010	0.230 ± 0.012
0.80	16.3%	0.558 ± 0.007	0.569 ± 0.010	0.242 ± 0.013
0.85	12.7%	0.567 ± 0.007	0.574 ± 0.010	0.245 ± 0.013
0.90	8.9%	0.574 ± 0.007	0.584 ± 0.010	0.235 ± 0.012
0.95	5.4%	0.576 ± 0.007	0.585 ± 0.010	0.244 ± 0.013
1.00	0.0%	0.575 ± 0.007	0.586 ± 0.010	0.247 ± 0.013

Table 4.10: Results for Pythia-1.4B (Part 1/2)

ϵ	Speedup	wsc	winogrande	sciq	piqa	logiqa
0.50	55.0%	0.375 ± 0.048	0.545 ± 0.014	0.502 ± 0.016	0.629 ± 0.011	0.207 ± 0.016
0.55	47.7%	0.404 ± 0.048	0.561 ± 0.014	0.598 ± 0.016	0.628 ± 0.011	0.217 ± 0.016
0.60	39.0%	0.452 ± 0.049	0.556 ± 0.014	0.732 ± 0.014	0.651 ± 0.011	0.217 ± 0.016
0.65	32.4%	0.471 ± 0.049	0.565 ± 0.014	0.798 ± 0.013	0.668 ± 0.011	0.235 ± 0.017
0.70	27.2%	0.471 ± 0.049	0.567 ± 0.014	0.828 ± 0.012	0.675 ± 0.011	0.221 ± 0.016
0.75	22.5%	0.471 ± 0.049	0.557 ± 0.014	0.837 ± 0.012	0.706 ± 0.011	0.233 ± 0.017
0.80	17.8%	0.471 ± 0.049	0.550 ± 0.014	0.847 ± 0.011	0.708 ± 0.011	0.221 ± 0.016
0.85	14.0%	0.471 ± 0.049	0.559 ± 0.014	0.852 ± 0.011	0.707 ± 0.011	0.210 ± 0.016
0.90	10.2%	0.471 ± 0.049	0.557 ± 0.014	0.861 ± 0.011	0.714 ± 0.011	0.204 ± 0.016
0.95	5.9%	0.471 ± 0.049	0.558 ± 0.014	0.866 ± 0.011	0.719 ± 0.010	0.215 ± 0.016
1.00	0.0%	0.471 ± 0.049	0.560 ± 0.014	0.867 ± 0.011	0.721 ± 0.010	0.221 ± 0.016

Table 4.11: Results for Pythia-1.4B (Part 2/2)

ϵ	Speedup	lambada_openai	arc_easy	arc_challenge
0.50	55.0%	0.246 ± 0.006	0.399 ± 0.010	0.201 ± 0.012
0.55	47.7%	0.322 ± 0.007	0.447 ± 0.010	0.207 ± 0.012
0.60	39.0%	0.395 ± 0.007	0.508 ± 0.010	0.227 ± 0.012
0.65	32.4%	0.458 ± 0.007	0.542 ± 0.010	0.226 ± 0.012
0.70	27.2%	0.507 ± 0.007	0.558 ± 0.010	0.235 ± 0.012
0.75	22.5%	0.542 ± 0.007	0.570 ± 0.010	0.253 ± 0.013
0.80	17.8%	0.565 ± 0.007	0.590 ± 0.010	0.253 ± 0.013
0.85	14.0%	0.577 ± 0.007	0.604 ± 0.010	0.262 ± 0.013
0.90	10.2%	0.585 ± 0.007	0.613 ± 0.010	0.263 ± 0.013
0.95	5.9%	0.588 ± 0.007	0.615 ± 0.010	0.261 ± 0.013
1.00	0.0%	0.589 ± 0.007	0.617 ± 0.010	0.265 ± 0.013

Table 4.12: Results for Pythia-2.8B (Part 1/2)

ϵ	Speedup	wsc	winogrande	sciq	piqa	logiqa
0.50	58.6%	0.394 ± 0.048	0.541 ± 0.014	0.436 ± 0.016	0.615 ± 0.011	0.233 ± 0.017
0.55	51.2%	0.462 ± 0.049	0.564 ± 0.014	0.554 ± 0.016	0.635 ± 0.011	0.249 ± 0.017
0.60	43.7%	0.423 ± 0.049	0.565 ± 0.014	0.688 ± 0.015	0.643 ± 0.011	0.237 ± 0.017
0.65	36.2%	0.404 ± 0.048	0.565 ± 0.014	0.767 ± 0.013	0.668 ± 0.011	0.226 ± 0.016
0.70	30.3%	0.385 ± 0.048	0.586 ± 0.014	0.813 ± 0.012	0.681 ± 0.011	0.229 ± 0.016
0.75	25.0%	0.385 ± 0.048	0.579 ± 0.014	0.839 ± 0.012	0.690 ± 0.011	0.217 ± 0.016
0.80	20.2%	0.385 ± 0.048	0.574 ± 0.014	0.853 ± 0.011	0.707 ± 0.011	0.195 ± 0.016
0.85	15.8%	0.385 ± 0.048	0.579 ± 0.014	0.867 ± 0.011	0.719 ± 0.010	0.195 ± 0.016
0.90	12.0%	0.385 ± 0.048	0.571 ± 0.014	0.867 ± 0.011	0.730 ± 0.010	0.195 ± 0.016
0.95	7.6%	0.385 ± 0.048	0.572 ± 0.014	0.870 ± 0.011	0.733 ± 0.010	0.201 ± 0.016
1.00	0.0%	0.385 ± 0.048	0.571 ± 0.014	0.882 ± 0.010	0.742 ± 0.010	0.212 ± 0.016

Table 4.13: Results for Pythia-2.8B (Part 2/2)

ϵ	Speedup	lambada_openai	arc_easy	arc_challenge
0.50	58.6%	0.249 ± 0.006	0.407 ± 0.010	0.217 ± 0.012
0.55	51.2%	0.340 ± 0.007	0.450 ± 0.010	0.238 ± 0.012
0.60	43.7%	0.416 ± 0.007	0.495 ± 0.010	0.243 ± 0.013
0.65	36.2%	0.482 ± 0.007	0.537 ± 0.010	0.276 ± 0.013
0.70	30.3%	0.533 ± 0.007	0.565 ± 0.010	0.276 ± 0.013
0.75	25.0%	0.570 ± 0.007	0.594 ± 0.010	0.288 ± 0.013
0.80	20.2%	0.596 ± 0.007	0.610 ± 0.010	0.291 ± 0.013
0.85	15.8%	0.618 ± 0.007	0.621 ± 0.010	0.299 ± 0.013
0.90	12.0%	0.630 ± 0.007	0.630 ± 0.010	0.294 ± 0.013
0.95	7.6%	0.635 ± 0.007	0.636 ± 0.010	0.291 ± 0.013
1.00	0.0%	0.637 ± 0.007	0.640 ± 0.010	0.294 ± 0.013

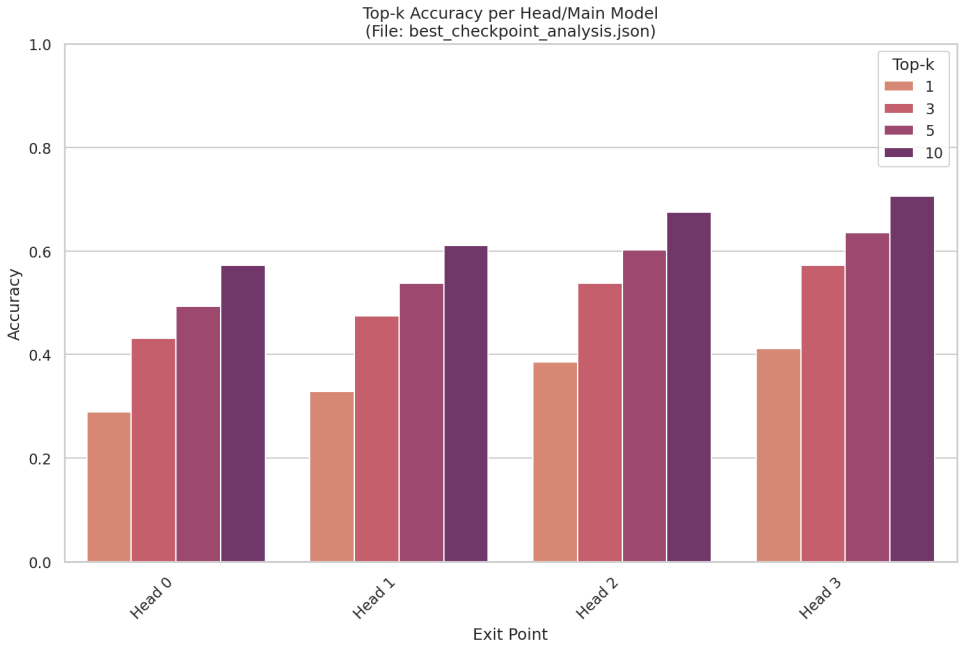


Figure 4.4: Top-1, top-3, top-5, and top-10 accuracy of each early exit head on Pythia-2.8B after post-training. Intermediate heads achieve competitive accuracy relative to the final head.

distribution of entropy for correct and incorrect outputs, demonstrating that the entropy metric can effectively separate right and wrong predictions in our model.

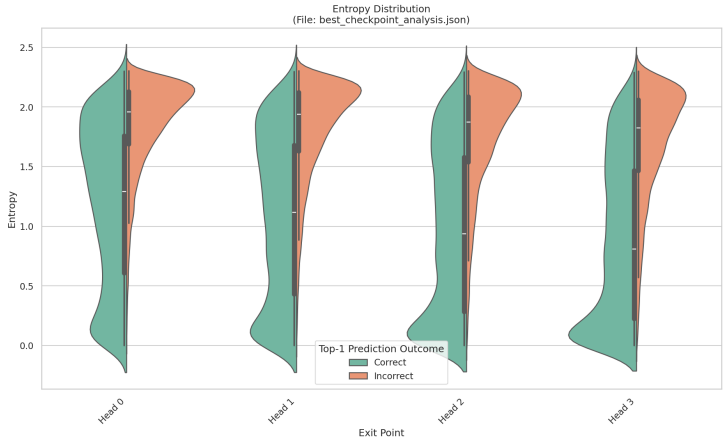


Figure 4.5: Violin plot of the entropy values for correct and incorrect outputs on the calibration set. The entropy metric provides good separation between right and wrong predictions.

4.7.4 Per-token accuracy analysis

To further support the idea that early exit heads are particularly effective on "easy" tokens, Figure 4.6 presents a heatmap of the top-1 accuracy per token for each head. This plot highlights that certain tokens achieve very high accuracy at intermediate heads, and these tokens often correspond to common or simple tokens in the dataset. This observation confirms that our early exit mechanism is able to leverage the inherent variability in token difficulty: easy tokens can be predicted confidently and accurately by earlier heads, enabling faster inference without sacrificing accuracy on these cases.

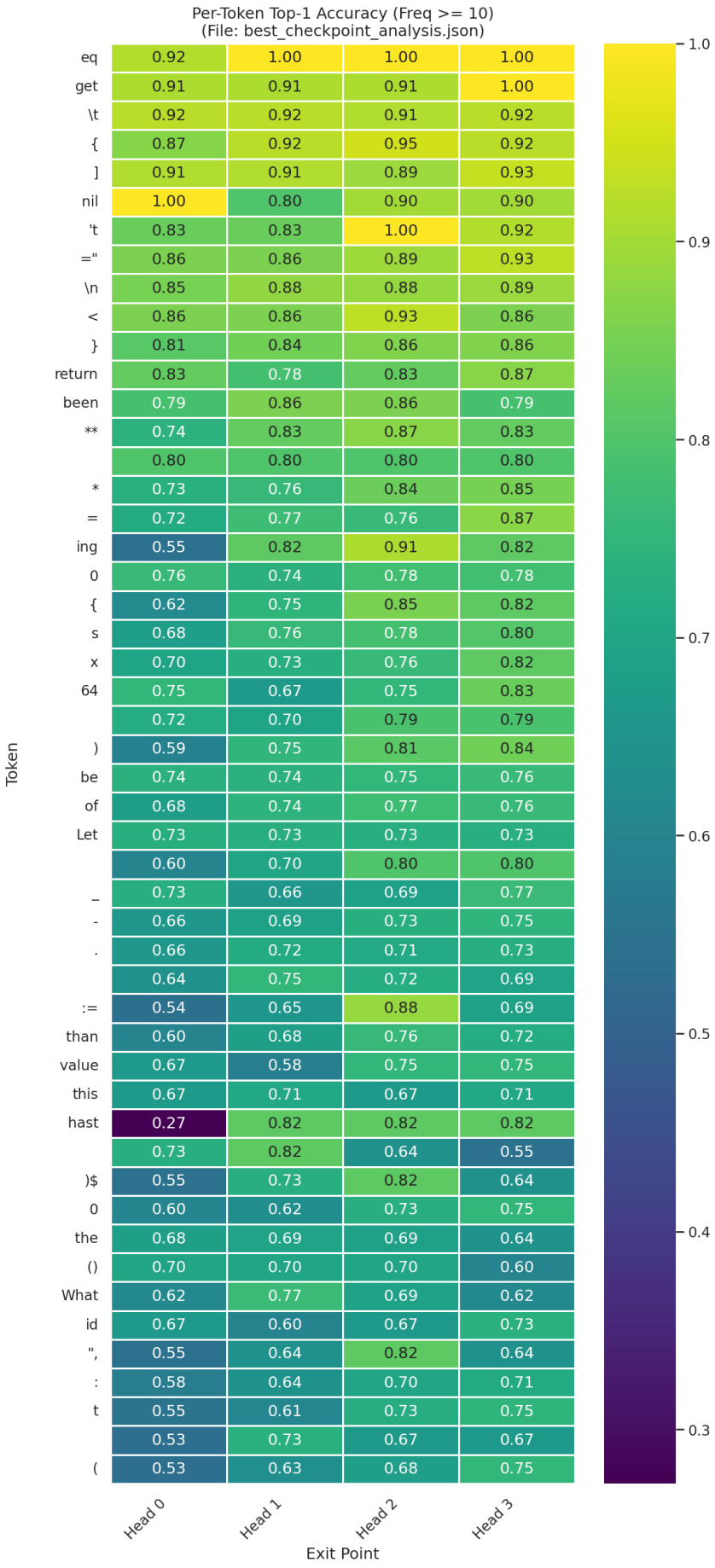


Figure 4.6: Heatmap of top-1 accuracy per token for each early exit head. Tokens with high accuracy at intermediate heads are typically common or easy tokens, supporting the use of early exit for efficient inference.

Conclusion and Perspectives

The computational demands of modern artificial intelligence have reached a critical juncture where the tension between performance and efficiency has become one of the most pressing challenges facing the field. This thesis has addressed this fundamental challenge through early exit neural networks—adaptive inference systems that dynamically adjust computational complexity based on input-specific requirements.

Ongoing Work

Building upon the theoretical foundations and practical frameworks established in this thesis, several research directions are currently being explored to further advance early exit neural networks.

Belief State Evolution in Early Exit Networks

A particularly promising avenue involves incorporating classification decisions into a *belief state* that evolves as information propagates through the network layers. This approach fundamentally reimagines early exit decision-making as a sequential decision process where the network maintains and updates probabilistic beliefs about the correct classification.

The core innovation lies in modeling early exit as a dynamic programming problem where each auxiliary head provides new evidence that updates the system's belief about the input's true label. At head ψ^i, the network maintains a belief state $p^{\psi^i} \in \Delta^{K-1}$ representing the probability distribution over possible classes given information from the first $\psi^1, \ldots, \psi^{i-1}$ heads. When processing proceeds to head ψ^{i+1}, the new prediction outcome $\xi^{\psi^{i+1}}$ serves as evidence to update the belief using Bayes' rule:

$$p^{\psi^{i+1}} = \mathcal{U}(p^{\psi^i}, \xi^{\psi^i}) \ ,$$

where $\mathcal{U}$ represents the Bayesian update operator.

The exit decision at each layer becomes a comparison between two costs: the immediate classification cost based on current beliefs versus the expected future cost of continuing processing. Formally, the network stops at head ψ^i if:

$$\text{Cost}_{\text{stop}}(p^{\psi^i}) < c^{\psi^{i+1}} + \mathbb{E}[\text{Cost}_{\text{continue}}(p^{\psi^{i+1}})]$$

where $c^{\psi^{i+1}}$ represents the computational cost of the next head and the expectation captures the average continuation cost over the random outcomes of head ψ^{i+1}.

This framework addresses several key challenges in current early exit methods. First, it provides a principled way to incorporate information from all previous heads rather than making decisions based solely on the current head's output. Second, it enables pre-computation of optimal stopping policies through dynamic programming, making inference extremely efficient. Third, it naturally handles the conditional dependence between auxiliary heads, capturing how classification confidence evolves through the network depth.

The approach requires careful consideration of how prediction outcomes ξ_l are defined and how the belief update mechanism accounts for the correlation structure between auxiliary heads. Current investigations explore various formulations ranging from simple independence assumptions to more complex conditional dependence models.

This belief state framework represents a significant advancement in understanding how classification decisions evolve within neural networks and what information can be extracted from the collective behavior of auxiliary heads. It provides both theoretical insights into the nature of hierarchical representation learning and practical improvements in exit decision quality.

Dynamic Early Exit for Speculative Decoding Enhancement

Another promising research direction extends the LLM early exit framework to enhance speculative decoding methods, particularly LayerSkip. While traditional speculative decoding approaches use separate draft models or fixed early exit configurations, this ongoing work demonstrates how dynamic early exit can significantly improve existing acceleration techniques.

LayerSkip represents the state-of-the-art in self-speculative decoding, using fixed early exit layers and a predetermined number of speculated tokens. However, this fixed approach requires manual optimization and may not adapt well to varying token complexities within sequences. The key innovation lies in replacing LayerSkip's static configuration with confidence-driven dynamic selection:

Dynamic Layer Selection: Instead of using a fixed early exit layer E, the system employs confidence-based dynamic selection where each token can exit at different depths based on its complexity and the model's certainty.

Adaptive Speculation: Rather than generating a fixed number S of speculated tokens, the approach dynamically determines speculation length based on confidence thresholds, allowing longer draft sequences for high-confidence regions and shorter ones for uncertain areas.

Preliminary experimental results on Pythia-2.8B demonstrate the complementary nature of these approaches. Standard LayerSkip with fixed heads achieved 27-49% token acceptance rates depending on the chosen layer configuration. The enhanced LayerSkip using dynamic early exit achieved 72% acceptance rate, representing a substantial improvement in both acceptance rate and draft sequence length.

This work highlights an important paradigm shift: rather than viewing early exit methods as competitors to speculative decoding, they can be synergistic enhancements. The dynamic early exit framework developed for accuracy-preserving LLM inference naturally extends to improve any speculative decoding approach by making exit decisions adaptive and confidence-driven. This opens possibilities for enhancing other speculative methods beyond LayerSkip, potentially benefiting the broader ecosystem of LLM acceleration techniques.

Synthesis of Contributions

This work advances early exit neural networks through two complementary paradigms addressing distinct deployment scenarios. **EERO (Early Exit with Reject Option)** provides the first principled approach for budget-constrained classification with formal computational guarantees through exponential weights aggregation, enabling direct budget specification rather than heuristic threshold tuning. The **LLM early exit framework** achieves substantial efficiency gains (up to 58.6% speedup) while preserving accuracy through self-supervised training and conformal prediction guarantees.

The theoretical contributions establish rigorous mathematical foundations connecting adaptive neural networks to statistical learning theory. The formalization of early exit as classification with reject option enables formal performance guarantees — whether by enforcing a computational budget or ensuring a target accuracy. The pluggable framework advantage enables adding early exit capabilities to existing pre-trained models without expensive retraining, democratizing access to adaptive inference.

Impact and Implications

These contributions address two critical research gaps: the need for strict computational budget adherence in resource-constrained environments, and efficient inference for large language models with accuracy preservation. The work enables practical deployment in edge computing, real-time systems, and large-scale cloud services while contributing to more sustainable AI through reduced computational requirements.

The paradigm shift from uniform computation to input-adaptive processing represents a fundamental change in neural network design, with early exit methods becoming components of comprehensive efficiency strategies alongside pruning, quantization, and distillation approaches.

Future Directions

Key research directions include: **architectural innovations** extending to multimodal models with automated exit placement and hardware-aware optimization; **training methodologies** developing unified objectives and meta-learning approaches; **inference advances** incorporating context-aware policies and multi-objective optimization; and **theoretical extensions** to generation tasks with robustness guarantees and sample complexity analysis.

Limitations and Open Challenges

Current limitations include batching inefficiency in practical deployment, confidence estimation challenges under distribution shift, and the need for principled exit placement optimization. Open research questions involve characterizing optimal trade-off surfaces across multiple objectives, developing dynamic adaptation capabilities, and improving interpretability of exit decisions.

Long-term Vision

The ultimate goal is ubiquitous adaptive AI systems where every neural network can dynamically adjust computational behavior based on context and requirements. This vision encompasses personalized efficiency balancing user preferences with resource constraints, contributing to sustainable AI scaling that enables continued advancement while maintaining environmental compatibility. The work presented here provides foundational frameworks toward intelligent, adaptive, and sustainable AI systems that automatically optimize performance-efficiency trade-offs across diverse deployment scenarios.

Bibliography

[1] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in Neural Information Processing Systems*, 35:23716–23736, 2022.

[2] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] H. Bai, W. Zhang, L. Hou, L. Shang, J. Jin, X. Jiang, Q. Liu, M. Lyu, and I. King. BinaryBERT: Pushing the Limit of BERT Quantization, July 2021. URL http://arxiv.org/abs/2012.15701. arXiv:2012.15701 [cs].

[4] A. Bakhtiarnia, Q. Zhang, and A. Iosifidis. Multi-exit vision transformer for dynamic inference. *arXiv preprint arXiv:2106.15183*, 2021.

[5] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, pages 610–623, 2021.

[6] E. Bengio, P.-A. Bacon, J. Pineau, and D. Precup. Conditional computation in neural networks for faster models. In *ICLR 2016 Workshop*, 2015.

[7] S. Biderman, H. Schoelkopf, Q. Anthony, H. Bradley, K. O'Brien, E. Hallahan, M. A. Khan, S. Purohit, U. S. Prashanth, E. Raff, A. Skowron, L. Sutawika, and O. van der Wal. Pythia: A suite for analyzing large language models across training and scaling, 2023. URL https://arxiv.org/abs/2304.01373.

[8] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama. Adaptive neural networks for efficient inference, 2017. URL https://arxiv.org/abs/1702.07811.

[9] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[10] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[11] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[12] M.-A. Charusaie and S. Samadi. A unifying post-processing framework for multi-objective learn-to-defer problems. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 23705–23755. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/2a5a41a536d3ada8fbf61a9d6fbf18d2-Paper-Conference.pdf.

[13] C. Chen, S. Borgeaud, G. Irving, J.-B. Lespiau, L. Sifre, and J. Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

[14] K. Chen, J. Wang, X. Liu, et al. SmartBERT: Dynamic layer skipping and early exiting for efficient transformer inference. In *Proc. of IJCAI*, 2023.

[15] X. Chen, H. Dai, Y. Li, X. Gao, and L. Song. Learning to stop while learning to predict. In *International Conference on Machine Learning*, pages 1520–1530. PMLR, 2020.

[16] Y. Chen, Y. Li, T. Kong, L. Qi, R. Chu, L. Li, and J. Jia. Multi-modal early fusion for efficient multi-domain learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8527–8537, 2021.

[17] Y. Chen, X. Liu, H. Xu, X. Wang, X. He, A. Li, T. Chen, and Z. Wang. Eellm: Large-scale training and inference of early-exit large language models with 3d parallelism. In *Proceedings of the 41st International Conference on Machine Learning*, pages 7074–7091, 2024.

[18] C. Chow. An optimum character recognition system using decision functions. *IRE Transactions on Electronic Computers*, EC-6(4):247–254, 1957.

[19] C. Chow. On optimum recognition error and reject tradeoff. *IEEE Transactions on Information Theory*, 16(1):41–46, 1970.

[20] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel. PaLM: Scaling Language Modeling with Pathways, Oct. 2022. URL http://arxiv.org/abs/2204.02311. arXiv:2204.02311 [cs].

[21] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[22] C. Cortes, G. DeSalvo, and M. Mohri. Learning with rejection. In *International Conference on Algorithmic Learning Theory*, pages 67–82. Springer, 2016.

[23] A. Dalalyan and A. B. Tsybakov. Aggregation by exponential weighting, sharp pac-bayesian bounds and sparsity. *Machine Learning*, 72(1–2):39–61, Apr. 2008. ISSN 1573-0565. doi: 10.1007/s10994-008-5051-0. URL http://dx.doi.org/10.1007/s10994-008-5051-0.

[24] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. IEEE, 2009.

[25] J. Deng, J. Krause, A. C. Berg, and L. Fei-Fei. Large-scale object classification using label relation graphs. In *European conference on computer vision*, pages 48–64. Springer, 2014.

[26] C. Denis and M. Hebiri. Consistency of plug-in confidence sets for classification in semi-supervised learning. *Journal of Nonparametric Statistics*, 2019.

[27] C. Denis, M. Hebiri, and A. Zaoui. Regression with reject option and application to knn. *arXiv preprint arXiv:2006.16597*, 2020.

[28] C. Denis, M. Hebiri, B. Njike, and X. Siebert. Active learning algorithm through the lens of rejection arguments. *arXiv preprint arXiv: arXiv:2208.14682*, 2022.

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

[30] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv:2010.11929 [cs]*, Oct. 2020. URL http://arxiv.org/abs/2010.11929. arXiv: 2010.11929.

[31] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[32] R. El-Yaniv and Y. Wiener. Foundations of machine learning. *MIT Press*, 2010.

[33] M. Elbayad, J. Gu, E. Grave, and M. Auli. Depth-adaptive transformer. In *International Conference on Learning Representations*, 2020.

[34] A. Fan, E. Grave, and A. Joulin. Reducing Transformer Depth on Demand with Structured Dropout, Sept. 2019. URL http://arxiv.org/abs/1909.11556. arXiv:1909.11556 [cs, stat].

[35] W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

[36] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1039–1048, 2017.

[37] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[38] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning (ICML)*, pages 1050–1059, 2016.

[39] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, et al. The Pile: An 800GB dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[40] Y. Geifman and R. El-Yaniv. Selective classification for deep neural networks. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[41] I. Gibbs and E. Candes. Adaptive conformal inference under distribution shift. In *Advances in Neural Information Processing Systems*, volume 34, pages 1331–1343, 2021.

[42] Y. Grandvalet, A. Rakotomamonjy, J. Keshet, and S. Canu. Support vector machines with a reject option. In *NIPS*, pages 537–544, 2009.

[43] A. Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

[44] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning (ICML)*, pages 1321–1330, 2017.

[45] N. Gupta, H. Narasimhan, W. Jitkrittum, A. S. Rawat, A. K. Menon, and S. Kumar. Language model cascades: Token-level uncertainty and beyond, 2024. URL https://arxiv.org/abs/2404.10136.

[46] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

[47] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang. Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[48] Y. Han, Y. Pu, Z. Lai, C. Wang, S. Song, J. Cao, W. Huang, C. Deng, and G. Huang. Learning to weight samples for dynamic early-exiting networks, 2022. URL https://arxiv.org/abs/2209.08310.

[49] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[50] 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 (CVPR)*, June 2016.

[51] R. Herbei and M. Wegkamp. Classification with reject option. *Canad. J. Statist.*, 34 (4):709–721, 2006.

[52] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[53] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[54] S. Hor, Y. Qian, M. Pilanci, and A. Arbabian. Adaptive inference: Theoretical limits and unexplored opportunities, 2024. URL https://arxiv.org/abs/2402.04359.

[55] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[56] S. R. Howard, A. Ramdas, J. McAuliffe, and J. Sekhon. Time-uniform, nonparametric, nonasymptotic confidence sequences. *The Annals of Statistics*, 49(2):1055–1080, 2021.

[57] G. Huang, D. Chen, T. Li, F. Wu, L. Van Der Maaten, and K. Weinberger. Multi-scale dense networks for resource efficient image classification. In *6th International Conference on Learning Representations, ICLR 2018*. OpenReview.net, 2018.

[58] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

[59] F. Ilhan, K.-H. Chow, S. Hu, T. Huang, S. Tekin, W. Wei, Y. Wu, M. Lee, R. Kompella, H. Latapie, G. Liu, and L. Liu. Adaptive deep neural network inference optimization with eenet, 2023. URL https://arxiv.org/abs/2301.07099.

[60] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2017. URL https://arxiv.org/abs/1712.05877.

[61] W. Jitkrittum, N. Gupta, A. K. Menon, H. Narasimhan, A. S. Rawat, and S. Kumar. When does confidence-based cascade deferral suffice?, 2024. URL https://arxiv.org/abs/2307.02764.

[62] J. Kaddour. The minipile challenge for data-efficient language models. *arXiv preprint arXiv:2304.08442*, 2023.

[63] C. Kang, S. Xiang, S. Liao, C. Xu, and C. Pan. Big-little net: An efficient multi-scale feature representation for visual and speech recognition. In *International Conference on Learning Representations*, 2017.

[64] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[65] Y. Kaya, S. Hong, and T. Dumitras. Shallow-deep networks: Understanding and mitigating network overthinking. In *International conference on machine learning*, pages 3301–3310. PMLR, 2019.

[66] A. Kendall, Y. Gal, and R. Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Vision Recognition (CVPR)*, pages 7482–7491, 2018.

[67] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-100. *(Canadian Institute for Advanced Research)*, 2009.

[68] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[69] P. Kubaty, B. Wójcik, B. Krzepkowski, M. Michaluk, T. Trzciński, J. Pomponi, and K. Adamczewski. How to train your multi-exit model? analyzing the impact of training strategies, 2025. URL https://arxiv.org/abs/2407.14320.

[70] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane. Spinn: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–15, 2020.

[71] S. Laskaridis, A. Kouris, and N. Lane. Adaptive inference through early-exit networks: Design, challenges and directions. In *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, pages 1–6, 2021.

[72] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. *Advances in neural information processing systems*, 2:598–605, 1990.

[73] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[74] J. Lei. Classification with confidence. *Biometrika*, 101(4):755–769, 2014.

[75] Y. Leviathan, M. Kalman, and Y. Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, pages 23100–23116, 2023.

[76] J. Li, Z. Chen, and Q. Zhao. Random loss weighting for deep multi-task learning, 2019.

[77] T. Li, A. Beirami, M. Sanjabi, and V. Smith. Edgebert: Sentence-level energy optimizations for latency-aware multi-task nlp inference. In *International Conference on Machine Learning*, pages 6140–6150. PMLR, 2021.

[78] X. Li, C. Lou, Z. Zhu, Y. Chen, Y. Shen, Y. Ma, and A. Zou. Predictive exit: Prediction of fine-grained early exits for computation- and energy-efficient inference, 2022. URL https://arxiv.org/abs/2206.04685.

[79] Y. Li, R. M. Wehbe, F. S. Ahmad, H. Wang, and Y. Luo. Hierarchical neural networks for sequential sentence classification in medical scientific abstracts. *EMNLP-IJCNLP*, pages 3100–3109, 2019.

[80] Z. Li, S. Shen, Y. Xu, et al. SkipBERT: Skipping layers for efficient BERT inference. In *Proc. of ACL*, 2022.

[81] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021.

[82] A. Ligozat, J. Lefèvre, A. Bugeau, and J. Combaz. Unraveling the hidden environmental impacts of AI solutions for environment. *CoRR*, abs/2110.11822, 2021. URL https://arxiv.org/abs/2110.11822.

[83] S. Lin, B. Ji, R. Ji, and A. Yao. A closer look at branch classifiers of multi-exit architectures. *arXiv preprint arXiv:2204.13347*, 2022.

[84] M. Liu, S. Malladi, N. Jojic, P. Kohli, and R. Frostig. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*, 2024.

[85] S. Liu, K. Zhao, Y. Zhang, Q. Chen, S. Tang, and Y. Zhuang. Daisy: Data adaptive self-supervised early exit for speed-accuracy trade-offs, 2022.

[86] T. Liu, S. Ren, H. Zhou, et al. Early exit is a natural capability of transformer-based language models. *arXiv preprint arXiv:2402.00000*, 2024.

[87] W. Liu, P. Zhou, Z. Zhao, Z. Wang, H. Deng, and Q. Ju. Fastbert: a self-distilling bert with adaptive inference time, 2020.

[88] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021.

[89] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11976–11986, 2022.

[90] T. Luo, K. Kramer, D. Goldgof, L. Hall, S. Samson, A. Remsen, T. Hopkins, and D. Cohn. Active learning to recognize multiple types of plankton. *Journal of Machine Learning Research*, 6(4), 2005.

[91] Y. Matsubara, M. Levorato, and F. Restuccia. Split computing and early exiting for deep learning applications: Survey and research challenges, 2022.

[92] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[93] L. Meng, H. Zhao, Z. Rong, C. Wu, T. Furlanello, and R. Urtasun. Adavit: Adaptive vision transformers for efficient image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12309–12318, 2022.

[94] D. Molchanov, A. Ashukha, and D. Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, pages 2498–2507. PMLR, 2017.

[95] M. Naadeem, J. Zucker, and B. Hanczar. Accuracy-rejection curves (ARCs) for comparing classification methods with a reject option. In *MLSB*, pages 65–81, 2010.

[96] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[97] N. Okati, A. De, and M. Gomez-Rodriguez. Differentiable learning under triage, 2021. URL https://arxiv.org/abs/2103.08902.

[98] R. G. Pacheco, D. J. Bajpai, M. Shifrin, R. S. Couto, D. S. Menasché, M. K. Hanawal, and M. E. M. Campista. Ucbee: A multi armed bandit approach for early-exit in neural networks. *IEEE Transactions on Network and Service Management*, 22(1): 107–120, 2025. doi: 10.1109/TNSM.2024.3479076.

[99] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[100] A. Radford, J. W. Jeong, J. Clark, et al. Robust speech recognition via large-scale weak supervision. *arXiv preprint arXiv:2302.04200*, 2023.

[101] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[102] R. Raina, A. Madhavan, and A. Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning*, pages 873–880, 2009.

[103] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

[104] F. Regol, J. Chataoui, and M. Coates. Jointly-learned exit and inference for a dynamic neural network : Jei-dnn, 2024. URL https://arxiv.org/abs/2310.09163.

[105] P. Rigollet and A. B. Tsybakov. Sparse Estimation by Exponential Weighting. *Statistical Science*, 27(4):558 – 575, 2012.

[106] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[107] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[108] C. Sacco, R. Esposito, and M. Cacciari. Zero time waste in pre-trained early exit neural networks. *Neural Networks*, 168:593–604, 2023.

[109] S. Samsi, D. Zhao, J. McDonald, B. Li, A. Michaleas, M. Jones, W. Bergeron, J. Kepner, D. Tiwari, and V. Gadepally. From Words to Watts: Benchmarking the Energy Costs of Large Language Model Inference, Oct. 2023. URL http://arxiv.org/abs/2310.03003. arXiv:2310.03003 [cs].

[110] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[111] S. Scardapane, D. Comminiello, M. Scarpiniti, and A. Uncini. Differentiable branching in deep networks for fast inference. In *2020 28th European Signal Processing Conference (EUSIPCO)*, pages 1475–1479. IEEE, 2020.

[112] S. Scardapane, A. Scardapane, D. Bacciu, and A. Uncini. Why should we add early exits to neural networks? *Cognitive Computation*, 12(5):954–966, 2020.

[113] S. Scardapane, M. Scarpiniti, E. Baccarelli, and A. Uncini. Class-specific early exit design methodology for convolutional neural networks. *Applied Soft Computing*, 110: 107661, 2021.

[114] T. Schuster, A. Fisch, and R. Barzilay. Confident adaptive language modeling. *arXiv preprint arXiv:2107.11874*, 2021.

[115] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni. The right tool for the job: Matching model complexity to instance difficulty. In *Proc. of ACL*, 2020.

[116] R. Schwartz, G. Stanovsky, S. Swayamdipta, J. Dodge, and N. A. Smith. The right tool for the job: Matching model and instance complexities. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6640–6651, 2020.

[117] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

[118] S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer. Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT, Sept. 2019. URL http://arxiv.org/abs/1909.05840. arXiv:1909.05840 [cs].

[119] C. N. Silla Jr and A. A. Freitas. A survey of hierarchical classification across different application domains. *Data mining and knowledge discovery*, 22(1):31–72, 2011.

[120] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[121] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[122] E. Strubell, A. Ganesh, and A. McCallum. Energy and policy considerations for deep learning in nlp. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, 2019.

[123] S. Sun, Y. Cheng, Z. Gan, and J. Liu. Patient Knowledge Distillation for BERT Model Compression, Aug. 2019. URL http://arxiv.org/abs/1908.09355. arXiv:1908.09355 [cs].

[124] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou. MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices, Apr. 2020. URL http://arxiv.org/abs/2004.02984. arXiv:2004.02984 [cs].

[125] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[126] M. Tan and Q. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[127] S. Tang, B. Li, S. Wang, S. Liu, H. Zhang, L. Li, Y. Qiao, and W. Ouyang. You need multiple exiting: Dynamic early exiting for accelerating unified vision language model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15366–15376, 2023.

[128] S. Teerapittayanon, B. McDanel, and H.-T. Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE, 2016.

[129] R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du, Y. Li, H. Lee, H. S. Zheng, A. Ghafouri, M. Menegali, Y. Huang, M. Krikun, D. Lepikhin, J. Qin, D. Chen, Y. Xu, Z. Chen, A. Roberts, M. Bosma, V. Zhao, Y. Zhou, C.-C. Chang, I. Krivokon, W. Rusch, M. Pickett, P. Srinivasan, L. Man, K. Meier-Hellstern, M. R. Morris, T. Doshi, R. D. Santos, T. Duke, J. Soraker, B. Zevenbergen, V. Prabhakaran, M. Diaz, B. Hutchinson, K. Olson, A. Molina, E. Hoffman-John, J. Lee, L. Aroyo, R. Rajakumar, A. Butryna, M. Lamm, V. Kuzmina, J. Fenton, A. Cohen, R. Bernstein, R. Kurzweil, B. Aguera-Arcas, C. Cui, M. Croak, E. Chi, and Q. Le. LaMDA: Language Models for Dialog Applications, Feb. 2022. URL http://arxiv.org/abs/2201.08239. arXiv:2201.08239 [cs].

[130] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR, 2021.

[131] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. LLaMA: Open and Efficient Foundation Language Models, Feb. 2023. URL http://arxiv.org/abs/2302.13971. arXiv:2302.13971 [cs].

[132] F. Valade, M. Hebiri, and P. Gay. Eero: Early exit with reject option for efficient classification with limited budget, 2024. URL https://arxiv.org/abs/2402.03779.

[133] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[134] V. Vovk, A. Gammerman, and C. Saunders. Machine-learning applications of algorithmic randomness. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, pages 444–453. Morgan Kaufmann, 1999.

[135] V. Vovk, A. Gammerman, and G. Shafer. *Algorithmic learning in a random world*. Springer, New York, 2005.

[136] D. Wang, C. Devin, Q.-Z. Cai, F. Yu, and T. Darrell. Multi-scale continuous crfs as sequential deep networks for monocular depth estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5354–5362, 2018.

[137] G. Wang, S. Zhao, X. Zhang, J. Ding, J. Liu, J. Tang, and J. Liu. Device-cloud collaborative learning for multimodal large language models. In *Advances in Neural Information Processing Systems*, volume 34, pages 14840–14853, 2021.

[138] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni. Federated learning with adaptive early exiting. In *International Conference on Artificial Intelligence and Statistics*, pages 2864–2872. PMLR, 2021.

[139] J. Wang, H. Chen, R. He, et al. FREE: Fast, reliable early-exit transformers for efficient autoregressive generation. In *Proc. of EMNLP*, 2023.

[140] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 568–578, 2021.

[141] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424, 2018.

[142] Y. Wang, R. Huang, S. Song, Z. Huang, and G. Huang. Not all images are worth 16x16 words: Dynamic transformers for efficient image recognition. *Advances in Neural Information Processing Systems*, 34:11960–11973, 2021.

[143] Y. Wang, T. Li, L. Zhang, Z. Xu, and Z. Liu. Harmonized dense knowledge distillation training for multi-exit architectures. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10218–10226, 2021.

[144] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. Davis, K. Grauman, and R. Feris. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8817–8826, 2018.

[145] J. Xin, R. Tang, J. Lee, Y. Yu, and J. Lin. Deebert: Dynamic early exiting for accelerating bert inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2246–2251, 2020.

[146] J. Xin, R. Tang, J. Lee, Y. Yu, and J. Lin. DeeBERT: Dynamic Early Exiting for Accelerating BERT Inference, Apr. 2020. URL http://arxiv.org/abs/2004.12993. arXiv:2004.12993 [cs].

[147] J. Xin, R. Tang, and J. Lin. BERxiT: Early exiting strategies for BERT. In *Proc. of EACL*, 2021.

[148] C. Xu, W. Zhou, T. Ge, F. Wei, and M. Zhou. Elue: Efficient language understanding evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 1234–1249, 2020.

[149] Y. Yang, Y. Zheng, X. Li, et al. DEED: Dynamic early exiting for efficient decoding of large language models. In *Findings of NAACL*, 2024.

[150] Z. Yao, R. Y. Aminabadi, M. Zhang, X. Wu, C. Li, and Y. He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Neural Information Processing Systems*, 2022. doi: 10.48550/ARXIV.2206.01861.

[151] G.-I. Yu, J. S. Choi, Y. Kim, and B.-G. Lee. Orca: A distributed serving system for transformer-based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, 2022.

[152] M. Yuan and M. Wegkamp. Classification methods with reject option based on convex risk minimization. *J. Mach. Learn. Res.*, 11:111–130, 2010.

[153] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[154] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer. OPT: Open Pre-trained Transformer Language Models, June 2022. URL http://arxiv.org/abs/2205.01068. arXiv:2205.01068 [cs].

[155] Z. Zhang, W. Zhu, J. Zhang, P. Wang, R. Jin, and T.-S. Chung. PCEE-BERT: Accelerating BERT inference via patient and confident early exiting. In M. Carpuat, M.-C. de Marneffe, and I. V. Meza Ruiz, editors, *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 327–338, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-naacl.25. URL https://aclanthology.org/2022.findings-naacl.25.

[156] W. Zhou, C. Xu, T. Ge, J. McAuley, K. Xu, and F. Wei. Bert loses patience: Fast and robust inference with early exit. In *Advances in Neural Information Processing Systems*, volume 33, pages 18330–18341, 2020.

[157] W. Zhou, C. Xu, T. Ge, J. McAuley, K. Xu, and F. Wei. BERT Loses Patience: Fast and Robust Inference with Early Exit, Oct. 2020. URL http://arxiv.org/abs/2006.04152. arXiv:2006.04152 [cs].

[158] X. Zhu, J. Li, Y. Liu, C. Ma, and W. Wang. A Survey on Model Compression for Large Language Models, Sept. 2023. URL http://arxiv.org/abs/2308.07633. arXiv:2308.07633 [cs].