Dynamic Early-Exiting Networks for Efficient Deep Learning

Haoran Peng, BSc

School of Computing and Communications

Lancaster University

A thesis submitted for the degree of

Doctor of Philosophy

February, 2026

Dynamic Early-Exiting Networks for Efficient Deep Learning

Haoran Peng, BSc.

School of Computing and Communications, Lancaster University

A thesis submitted for the degree of *Doctor of Philosophy*. February, 2026.

Abstract

Dynamic neural networks are an effective approach for accelerating deep neural networks. Compared with traditional deep neural networks, dynamic neural networks can adjust their architectures based on different inputs. This helps reduce computational cost. Although they have been widely applied across various domains, several fundamental questions remain open. In this thesis, we investigate several fundamental questions in dynamic networks, including how to address gradient conflicts among classifiers during training, how to enable effective collaboration among classifiers during inference, and how to apply dynamic neural networks to deep reinforcement learning. As a representative form of dynamic neural networks, early-exiting networks are often used to investigate such foundational research questions. By adding extra classifiers in the middle of the network, early-exiting networks can exit early at these intermediate classifiers. This thesis addresses these research questions by examining early-exiting networks from three perspectives: training, inference, and the application in Monte Carlo Tree Search (MCTS).

First, in the training process, dynamic early-exiting networks simultaneously train multiple classifiers, which often leads to gradient conflicts. This thesis addresses this issue from the perspective of considering whether the gradient is necessary. We introduce a damping mechanism to suppress unnecessary gradients and thereby improve training performance. Moreover, our method is compatible with existing approaches that focus on managing trade-offs among different gradients, thereby enabling further performance improvements.

Secondly, we explore the problem of decision-centered teamwork in dynamic

early-exiting networks. The mainstream approach typically selects the output of the deepest classifier available at the exit point as the final prediction. However, even weaker classifiers can still contribute to the final prediction through collaboration. To leverage this information, we investigate Bayesian updating and voting-based methods, which integrate the outputs of weaker classifiers to improve inference performance.

Finally, we extend our investigation to online planning with deep neural networks by deploying dynamic early-exiting networks within the AlphaZero algorithm. In the inference phase, our results show that, under a fixed computational budget, using weaker networks to perform more simulations can sometimes outperform stronger networks. In the training phase, we further propose a deeply supervised learning–based method, which enables the construction of a dynamic early-exiting variant of the network which computes the policy-values in the MCTS while maintaining comparable training cost.

Acknowledgements

Reflecting on my Ph.D. journey, I realize that it has been a period filled with challenges, growth, and invaluable support. Along the way, I have been fortunate to receive guidance, encouragement, and understanding from many people. Without their help, this dissertation would not have been possible. I am sincerely grateful to all who have supported me during this journey.

First and foremost, I would like to express my deepest gratitude to my supervisor, Leandro Soriano Marcolino. Without his guidance and support, I would not have been able to complete my doctoral studies. He provided exceptionally detailed supervision throughout my research and offered insightful advice at every stage. His wisdom, clarity of thought, and dedication to mentoring have greatly influenced my academic development. Under his guidance, I was able to grow steadily as a researcher and ultimately complete this dissertation.

Beyond his academic guidance, he has always been patient, supportive, and generous with his time. He consistently encouraged me to persevere through challenges and to approach research with confidence and independence. I am truly grateful for his mentorship and for the positive impact he has had on both my academic journey and personal growth.

I am especially grateful to my parents for their long-term support over the years. Whenever I faced difficulties, they were always there to encourage me and offer their help. Their constant belief in me has given me the strength to overcome challenges and continue moving forward.

I would also like to sincerely thank my girlfriend, Guangyi Li, for her long-term support, care, and companionship over the years. Throughout my PhD journey, her support has been a continuous source of strength. Her understanding and patience made the challenges more manageable and the journey more meaningful. Because of her presence, this doctoral experience has been brighter and more fulfilling.

I would also like to thank my colleagues and friends for their support throughout my PhD studies. I am especially grateful to Gao Peng and Guhan Zheng for their

valuable academic discussions and assistance in my research. Their insights and collaboration have greatly contributed to my progress. I also sincerely thank Xingdu Wang, Zhi Han, Ziyang Zhang, Xihan Xu, Jieyu Zhang, and Naif Alshammari for their encouragement and support. Their friendship and positive energy have made this journey more enjoyable and less stressful. In addition, I am thankful to many other friends whose names I am unable to list here, but whose support I deeply appreciate.

Finally, I would like to express my sincere gratitude once again to everyone who has supported me throughout this journey. This Ph.D. experience has shaped not only my academic path but also my personal growth. I am deeply thankful for the guidance, encouragement, and companionship I have received. As I move forward to the next stage of my career, I will carry with me the lessons, support, and inspiration gained during these years.

Declaration

I declare that the work presented in this thesis is, to the best of my knowledge and belief, original and my own work. The material has not been submitted, either in whole or in part, for a degree at this, or any other university. This thesis does not exceed the maximum permitted word length of 80,000 words including appendices and footnotes, but excluding the bibliography. A rough estimate of the word count is: 23826

Haoran Peng

Publications

The research presented in this thesis has led to the following publications, which played an important role in shaping the direction of the work and provided substantial support to its findings throughout my PhD.

Haoran Peng, Leandro Soriano Marcolino, Bryan M. Williams, and Erickson R. Nascimento (2025c). "Diminishing Non-important Gradients for Training Dynamic Early-Exiting Networks". In: *In submission*

Haoran Peng and Leandro Soriano Marcolino (2025a). "Decision-centered Teamwork in Dynamic Early Exiting Networks". In: *In submission*

Haoran Peng and Leandro Soriano Marcolino (2025b). "Dynamic Policy-Value Monte Carlo Tree Search". In: *In submission*

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Dynamic Early Exiting Networks

Deep neural networks (DNNs) have driven notable progress in different areas such as computer vision (Krizhevsky et al., 2012; Simonyan et al., 2014; He et al., 2016; Huang et al., 2019), natural language processing (Vaswani et al., 2017), and deep reinforcement learning (Silver et al., 2016; Silver et al., 2017). However, the strong performance of deep neural networks (DNNs) often comes with substantial computational demands, which limits their use in resource-constrained settings such as widely deployed mobile devices and Internet of Things (IoT) systems.

To reduce inference cost, prior works have explored several strategies, including network pruning (LeCun et al., 1989; Yang et al., 2021), weight quantization (Hubara et al., 2016; Han et al., 2015), and lightweight architecture design (Howard et al., 2017; Zhang et al., 2018; Sandler et al., 2018). These approaches achieve efficient models that can deliver competitive accuracy while requiring significantly fewer computational resources. However, their effectiveness is primarily measured by average performance across a dataset. In practice, data samples vary considerably in difficulty: simple instances can often be handled correctly by smaller, low-cost models, whereas more challenging samples still demand the stronger representational capacity of larger deep neural networks (Huang et al., 2017b; Lin et al., 2017).

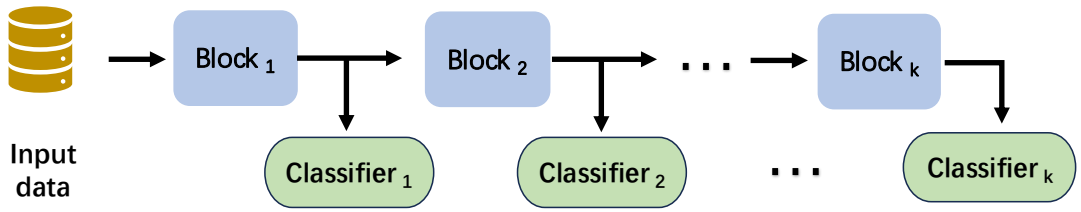


Figure 1.1: **Dynamic Early-Exiting Architecture. For easy inputs, the model can exit at intermediate classifiers to reduce computational cost.**

This discrepancy highlights the need for models that can adapt their computational effort to individual inputs rather than relying on a fixed architecture. Dynamic neural networks (Han et al., 2021) address this mismatch by performing data-dependent inference, selectively allocating computation—via layer/channel skipping (Wang et al., 2018; Veit et al., 2018; Jin et al., 2020), expert routing (Liu et al., 2018; Hehn et al., 2020), or early-exiting (Huang et al., 2017b; Yang et al., 2020b)—to achieve a better accuracy–efficiency trade-off.

Dynamic early-exiting networks are a representative form of dynamic neural networks. We present the architecture of a dynamic early-exiting network in Figure 1.1. They are motivated by the observation that many inputs can be correctly classified at shallow layers, making deeper and more costly computation unnecessary. To exploit this property, they augment the neural network backbone with multiple auxiliary classifiers attached at different depths, enabling early termination of inference, i.e., early-exiting.

During training, these classifiers are typically optimized simultaneously. A straightforward strategy is to sum their losses and jointly minimize them. At inference time, the model dynamically adjusts to each data sample: if an early

classifier is sufficiently confident (e.g., by exceeding a softmax threshold), the model exits at that classifier; otherwise, the data sample continues through deeper layers. In this way, the model terminates early for easy data samples, saving computation without uniformly reducing the model's capacity.

In recent years, dynamic early-exiting networks have been increasingly applied across various domains, including robotics (Yue et al., 2024), edge computing (Guo et al., 2024), large-scale models (Gao et al., 2023; Xin et al., 2020), and multimodal architectures (Tang et al., 2023). However, several fundamental challenges remain underexplored. For instance, during training, gradients from different exits may conflict, while at inference, relying solely on a single exit discards potentially useful information from other classifiers.

In this thesis, we investigate such fundamental problems in dynamic early-exiting networks. These problems are particularly important not only because they arise in widely used early-exiting architectures, but also because they can extend to other forms of dynamic neural networks. For example, gradient conflicts (Sun et al., 2022) encountered in training early-exiting networks similarly occur in other dynamic architectures, where multiple sub-networks must all achieve competitive performance during training.

Next, we present the research questions addressed in this thesis, along with our key contributions.

1.2 Research Questions and Our Contributions

In this section, we present the research problems and the corresponding contributions in this thesis.

1.2.1 Training of dynamic early-exiting networks

First, we explore the challenges that arise during the training phase of dynamic early-exiting networks. At dynamic early-exiting networks' inference time, these models

allocate computation *per input*—for example, deciding whether to exit early, skip layers, or route to a specialized branch—so any intermediate head may become the final decision for a substantial portion of samples (Han et al., 2021). As a result, training cannot focus on a single terminal classifier; all exits must be optimized jointly. This is because the classifiers share parameters, and updates to the shared layers affect the performance of multiple classifiers simultaneously. Work on foundational aspects of dynamic neural networks often adopts dynamic early exiting as the representative backbone, owing to its long research history, systematic exploration, and strong baseline implementations. Accordingly, this thesis investigates the following research question:

Q) How should we jointly train multiple classifiers in a dynamic early-exiting network?

Unlike conventional deep neural networks, dynamic early-exiting networks must train multiple classifiers jointly during training. Because we do not know which inputs will appear at test time, each classifier must be trained to perform well, since any of them may be selected during inference. In early designs of dynamic early-exiting networks, training followed a straightforward strategy, i.e. sum the losses, e.g. cross-entropy loss of all classifiers.

However, because each classifier's loss is computed independently, their gradients on the shared backbone can conflict even though the heads solve the same task. This interference leads to suboptimal updates and unstable exit performance. Recent work tackles this by explicitly managing trade-offs among gradients (e.g., reweighting or projection) (Han et al., 2022; Sun et al., 2022; Gong et al., 2024). In particular, meta-learning–based approaches use a small meta-network to assign adaptive weights to the per-head gradients during training, thereby balancing their influence. However, these methods overlook a critical premise: are all the gradients being traded off actually necessary for learning?

During training, deep neural networks can overfit, yielding gradients that no longer improve generalization. In dynamic early-exiting architectures, this

issue is amplified: multiple classifiers share a common backbone, so gradients that are unnecessary for one head not only fail to help that head but—via gradient conflict on shared parameters—can also degrade the training of other heads. Consequently, identifying and suppressing unnecessary gradients is a central requirement for effectively training dynamic early-exiting networks. Although dynamic early exiting has been studied extensively—and baseline architectures already deploy standard anti-overfitting techniques such as data augmentation, batch normalization, and early stopping—the architectural specifics of dynamic networks limit their effectiveness. For example, in baselines like MSDNet (Huang et al., 2017b) and RANet (Yang et al., 2020b), early stopping is applied at the model level: a single epoch is chosen based on validation performance. In practice, however, different exits typically have different optimal stopping points. A global choice therefore over-trains some heads and under-trains others, leaving substantial room for improvement.

This thesis proposes a novel training framework to address this challenge. We introduce a damping mechanism that jointly accounts for the training states of different classifiers. Concretely, an additional damping neuron is attached to the fully connected layer of each classifier. During training, besides optimizing the cross-entropy loss of all classifiers, we assign the damping neuron a small gradient, allowing it to dynamically suppress unnecessary updates according to the classifiers' individual training states. Building on this mechanism, we further design a *power-sqrt loss* to better balance the optimization across classifiers. We also demonstrate that our method is compatible with state-of-the-art linear scalarization methods (Han et al., 2022). Moreover, after damping redundant gradients, the values of the damping neurons can be reused as weights to manage the trade-offs among different gradients more effectively.

We summarize our contributions as follows:

1. We propose an adaptive damping mechanism that dynamically suppresses unnecessary gradients during training, leading to more stable and effective

optimization;

2. We design the *power-sqrt* loss, which incorporates the training states of all classifiers to better calibrate the damping gradient magnitude;

3. We show that our approach is fully compatible with existing linear scalarization methods, enabling seamless integration;

4. We conduct extensive experiments on CIFAR (Krizhevsky et al., 2009) and ImageNet (Deng et al., 2009), demonstrating consistent accuracy gains with negligible additional complexity.

1.2.2 Inference of dynamic early exiting networks

We move from training to inference and examine foundational issues in dynamic neural networks. In such models, low- and high-compute execution paths share most parameters; in canonical early exiting, deeper classifiers reuse all computations of shallower ones except for their terminal fully connected layer. Routing policies typically rely on shallow heads' confidence—e.g., the maximum softmax or an entropy threshold—to decide whether to exit early. Crucially, even when the policy chooses a deeper, more expensive head, the predictions from earlier heads have already been computed with negligible additional cost along that path. Yet prevailing systems discard these available outputs and return only the deepest head's prediction. Hence, we also study the following research question:

Q) Can we leverage the information already produced by shallow (weaker) classifiers?

In dynamic early-exiting architectures, deeper classifiers generally achieve higher accuracy on average, owing to their larger parameter capacity. However, this advantage is measured in aggregate performance. At the level of individual samples, an error made by a deeper classifier does not necessarily imply that the shallower classifiers also fail. In fact, due to the well-documented overthinking (Kaya et al.,

2019) phenomenon in deep neural networks, there are cases where a shallow classifier correctly recognizes an input that a deeper classifier misclassifies.

In multi-agent systems, it is well established that a group of weaker agents can sometimes outperform a single strong agent through effective collaboration (Marcolino et al., 2013). Inspired by this insight, we investigate whether cooperation among multiple classifiers in a dynamic early-exiting network can similarly enhance performance. However, extending this idea to deep learning presents two key challenges. First, large-scale tasks are considerably more complex—for example, ImageNet requires classification over 1,000 categories. Second, classifier confidence in deep neural networks remains an open problem, as existing confidence measures are often unreliable (Papamarkou et al., 2024).

This thesis explores multi-agent collaboration in dynamic early-exiting networks and introduces several complementary approaches. First, we propose a Bayesian updating method, where the likelihood of each classifier's output for different labels is estimated from a validation set and then incorporated through Bayesian updating. Synthetic experiments, however, reveal that reliable likelihood estimation is difficult with limited validation data, which restricts the effectiveness of this approach in certain scenarios.

To overcome this limitation, we develop a conformal prediction–based method that leverages its well-established reliability in large-scale deep learning. By providing more accurate confidence measures, conformal prediction enables refined likelihood estimation and improves the robustness of Bayesian updating.

Finally, we propose a logit voting strategy as a lightweight alternative. Although this method relies on less information and offers lower theoretical expressiveness than Bayesian updating, it often achieves strong empirical performance, making it a practical fallback when accurate likelihood estimation is infeasible.

Hence, in this thesis, we make the following contributions:

1. **Decision-centered collaboration in dynamic neural networks:** We investigate decision-centered collaboration in large-scale dynamic networks,

highlighting both the challenges and opportunities in this domain.

2. **Bayesian updating framework:** We develop a Bayesian updating–based collaboration method that strengthens the interaction among agents in dynamic networks.

3. **Conformal prediction for reliable likelihood estimation:** To mitigate inaccuracies in Bayesian updating, we propose a conformal prediction–based approach that significantly improves the reliability of likelihood estimation.

4. **Comprehensive empirical evaluation:** We conduct thorough experiments, including both synthetic studies and large-scale computer vision tasks, to validate the effectiveness and potential of the proposed Bayesian updating framework.

5. **Voting-based alternative:** We further examine a voting-based method as a lightweight alternative. Despite its simplicity and lower theoretical capacity, it achieves strong empirical results and serves as a practical fallback when Bayesian updating is limited by small validation sets.

1.2.3 Dynamic early exiting network in deep reinforcement learning

Although dynamic neural networks have found wide application in many popular domains, their use in deep reinforcement learning (DRL) remains largely unexplored. The original motivation of dynamic networks is to allocate computational resources adaptively based on the input instance. In DRL, however, this motivation takes on a unique form. Unlike conventional supervised learning, DRL does not rely on a fixed dataset; instead, training data are generated by the agent itself, as exemplified by the well-known AlphaZero (Silver et al., 2017) algorithm, which produces training samples through self-play. This characteristic of DRL introduces

distinctive challenges for the design and training of dynamic neural networks. In this chapter, we raise an intriguing research question:

Q) How can dynamic neural networks be applied in the domain of deep reinforcement learning?

We adopt AlphaZero as the baseline for our study. We then integrate dynamic early-exiting networks into the AlphaZero framework and evaluate their performance.

We choose AlphaZero because it is an improved version of the well-known AlphaGo algorithm. Unlike AlphaGo, it does not rely on human expert data. Instead, it learns from scratch by generating training data through self-play. This training strategy enables AlphaZero to achieve much stronger performance than AlphaGo, with a win rate of over 99%.

During training, unlike conventional deep learning that relies on a fixed dataset, AlphaZero generates its training samples through self-play. As a result, the quality of training data is directly tied to the current performance of the model itself. Moreover, AlphaZero requires substantial computational resources, making training speed a critical factor. Consequently, when applying dynamic early-exiting networks in this setting, it is essential to ensure that their use does not adversely affect training efficiency.

In the testing phase, the task setting in AlphaZero also differs from that of conventional deep learning. In standard supervised learning, a dynamic network decides how much computation to allocate (i.e., which classifier depth to use) for each input. By contrast, AlphaZero employs Monte Carlo Tree Search (MCTS), where multiple rollouts are performed on the same game state to estimate the distribution of action values, and the move with the highest estimated win probability is selected. This raises an important question: is a deeper network always more advantageous, or could a shallower network—when used to support more rollouts—yield better overall performance?

As discussed in earlier chapters, dynamic early exiting inherits several classical

challenges from supervised learning, including the distribution shift between training and deployment, the difficulty of reliable confidence estimation, and the sensitivity of hyperparameter-based exit policies. However, in deep reinforcement learning, the model itself can be sampled directly during training, which introduces new opportunities and constraints. These characteristics call for a careful re-examination of how early-exiting strategies should be designed and tuned in the reinforcement learning setting.

This thesis addresses the aforementioned challenges through novel contributions in both training and inference. During training, we introduce a knowledge distillation–based method that leverages the high-dimensional representations of a larger network to guide the optimization of a smaller one. Distinct from conventional deep learning approaches, our method exploits the properties of deep reinforcement learning by sampling directly from the model itself, thereby achieving more effective distillation. This enables policy–value MCTS networks with early-exiting architectures to attain improved performance without increasing training cost. During inference, we further build on the AlphaZero framework and propose a dynamic network selection strategy under a fixed computational budget. Compared with baseline methods, this approach yields significant performance gains.

Our main contributions are summarized as follows:

- We propose a knowledge-distillation–based training method tailored for reinforcement learning, which samples directly from the model to achieve more effective distillation.

- We enhance policy–value MCTS networks with early-exiting architectures, improving performance without additional training cost.

- We design a dynamic inference strategy under a fixed computational budget, achieving significant performance gains over baseline methods.

1.3 Thesis outline

This thesis is structured into six additional chapters following the introduction.

In Chapter 2, we provide the necessary background for understanding this thesis. We present a detailed explanation of the backbone network used in our work, the conventional training and inference methods for dynamic neural networks, and the metrics commonly employed to evaluate their performance. In addition, we review the fundamentals of Bayesian theory, introduce the task of computer Go as a representative problem in reinforcement learning, and describe the Monte Carlo Tree Search (MCTS) algorithm.

In Chapter 3, we review related work that is closely connected to the topics discussed in the subsequent chapters. In Chapter 4, we present our proposed training methodology for dynamic early-exiting networks. Chapter 5 focuses on our exploration of inference strategies, where we investigate dynamic network selection under computational constraints. In Chapter 6, we extend dynamic early-exiting to the reinforcement learning domain by integrating it with Monte Carlo Tree Search (MCTS).

Chapter 7 concludes this thesis by summarizing the main findings, discussing its limitations, and outlining directions for future research.

Chapter 2

Background

In this chapter, we present the key background material that supports the subsequent contributions and analyses of this thesis.

We begin in Section 2.1 with an introduction to dynamic early-exiting networks. In Section 2.1.1 we first present the classical early-exiting architectures, which are commonly used as backbones for studying general early-exiting problems. In Section 2.1.2, we present the conventional training and inference processes of dynamic early-exiting networks.

In Section 2.1.3, we present the experimental settings and evaluation metrics for dynamic early-exiting networks. As data-driven architectures with multiple computational paths, dynamic early-exiting networks are expected to perform well under varying resource budgets. Consequently, evaluation focuses less on absolute accuracy and more on the trade-off between accuracy and efficiency.

In Section 2.2, we introduce Monte Carlo Tree Search, which serves as the application scenario for Chapter 6. Section 2.2.1 begins with the topic of computer Go, followed by Section 2.2.2, which presents the well-known AlphaGo and AlphaZero algorithms.

2.1 Dynamic Early Exiting Networks

2.1.1 MSDNet and RANet backbones

Dynamic early-exiting is a representative dynamic neural network architecture. Because of its simple and straightforward design, it is often used to study general problems in dynamic neural networks. Dynamic early-exiting extends traditional deep neural networks, such as convolutional neural networks (CNNs) (Krizhevsky et al., 2012), by adding exit branches at intermediate layers—for example, attaching a fully connected classifier to a CNN block. This architecture enables the network to terminate inference early (i.e., perform an early exit), thereby reducing computational cost.

The first dynamic early-exiting architecture was proposed in BranchyNet (Teerapittayanon et al., 2016), which attaches fully connected classifiers to intermediate layers of convolutional neural networks (CNNs) such as ResNet (He et al., 2016) and VGG (Simonyan et al., 2014). This architecture remains widely used because it is easy to implement and does not alter the main backbone structure. For example, in Chapter 5, we adopt this approach to convert a ResNet backbone into a dynamic early-exiting network for Dynamic Policy Value Monte Carlo Tree Search. Concretely, we add a series of fully connected layers at different depths of the network, after several complete residual blocks, so that each can provide an intermediate output for early exiting.

Although the above approach achieves good acceleration, there is still room for improvement in model performance. Multi-Scale Dense Networks (MSDNet) (Huang et al., 2017b) provide a detailed analysis of this issue:

They first observed that when dynamic early-exiting is applied to mainstream deep neural network architectures, the performance of the deepest classifier tends to degrade, which limits the best achievable performance of the network. Moreover, the classifiers attached to intermediate classifiers also exhibit relatively limited performance. In traditional deep neural networks, shallow layers primarily learn

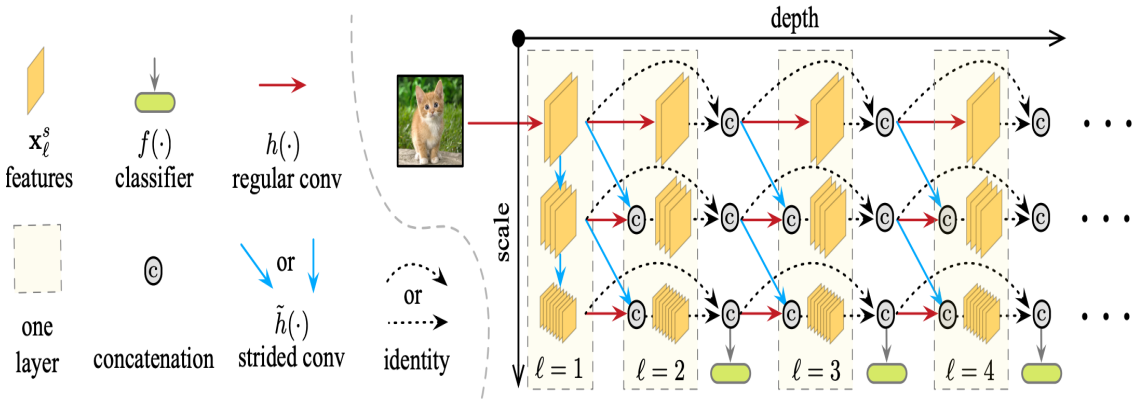


Figure 2.1: **MSDNet model architecture (from (Huang et al., 2017b))**

fine-scale features, while deeper layers capture more coarse-scale representations. However, in naïve dynamic early-exiting networks, the shallow classifiers lack access to coarse-scale features, even though such features are often crucial for reliable prediction. Moreover, shallow classifiers may also attempt to learn coarse-level features. While these features can provide short-term benefits by improving the performance of shallow exits, they are often less useful for deeper classifiers and may even hinder their training.

To address these issues, MSDNet introduces multi-scale feature maps together with dense connectivity to mitigate the limitations of naïve early-exiting networks. As shown in Figure 2.1, MSDNet restructures the network into a multi-scale design, which ensures that every classifier in the dynamic early-exiting architecture has access to coarse-level features. Moreover, MSDNet employs dense connections, which alleviate the problem that coarse-level features learned by shallow classifiers cannot be effectively utilized by deeper classifiers due to long propagation paths. This design also helps reduce interference among classifiers during training.

Compared with naïve dynamic early-exiting architectures that simply attach classifiers to intermediate layers, MSDNet achieves substantial performance improve-

ments and is widely regarded as a strong and essential baseline in this field. For instance, in studies on training dynamic early-exiting networks, MSDNet is almost always included as a baseline because its architectural design alleviates gradient conflicts during training. In Chapter 4, we present a detailed investigation of our methods built on the MSDNet backbone, along with comparative experiments.

Resolution Adaptive Network (RANet) (Yang et al., 2020b) approaches dynamic early-exiting from a novel perspective and further improves the architecture. The key observation is that simple samples can often be recognized using low-resolution images, whereas only difficult ones require high-resolution inputs. Building on the multi-scale feature maps of MSDNet, RANet introduces multi-scale input resolutions to further reduce computational cost. The overall network structure of RANet is illustrated in Figure 2.2.

RANet and MSDNet serve as standard baselines in the field of dynamic early-exiting, particularly for studying general challenges such as gradient conflicts during training and collaboration among classifiers at inference. We will further elaborate on these challenges in Chapters 4 and 5.

2.1.2 Conventional Training and Inference strategy of dynamic early-exiting

The conventional training strategy for early-exiting networks aggregates the losses from all classifiers, training them simultaneously from start to finish (Huang et al., 2017b; Yang et al., 2020b). The overall objective can be expressed as:

$$\mathcal{L} = \frac{1}{S} \sum_{k=1}^{K} \sum_{i=1}^{S} \mathcal{L}_i^{(k)}, \tag{2.1}$$

where

$$\mathcal{L}_i^{(k)} = \mathrm{CE}(f^{(k)}(\mathbf{x}_i, \theta^{(k)}), y_i). \tag{2.2}$$

Here, K denotes the number of classifiers, S the number of training samples, and y_i the ground-truth label.

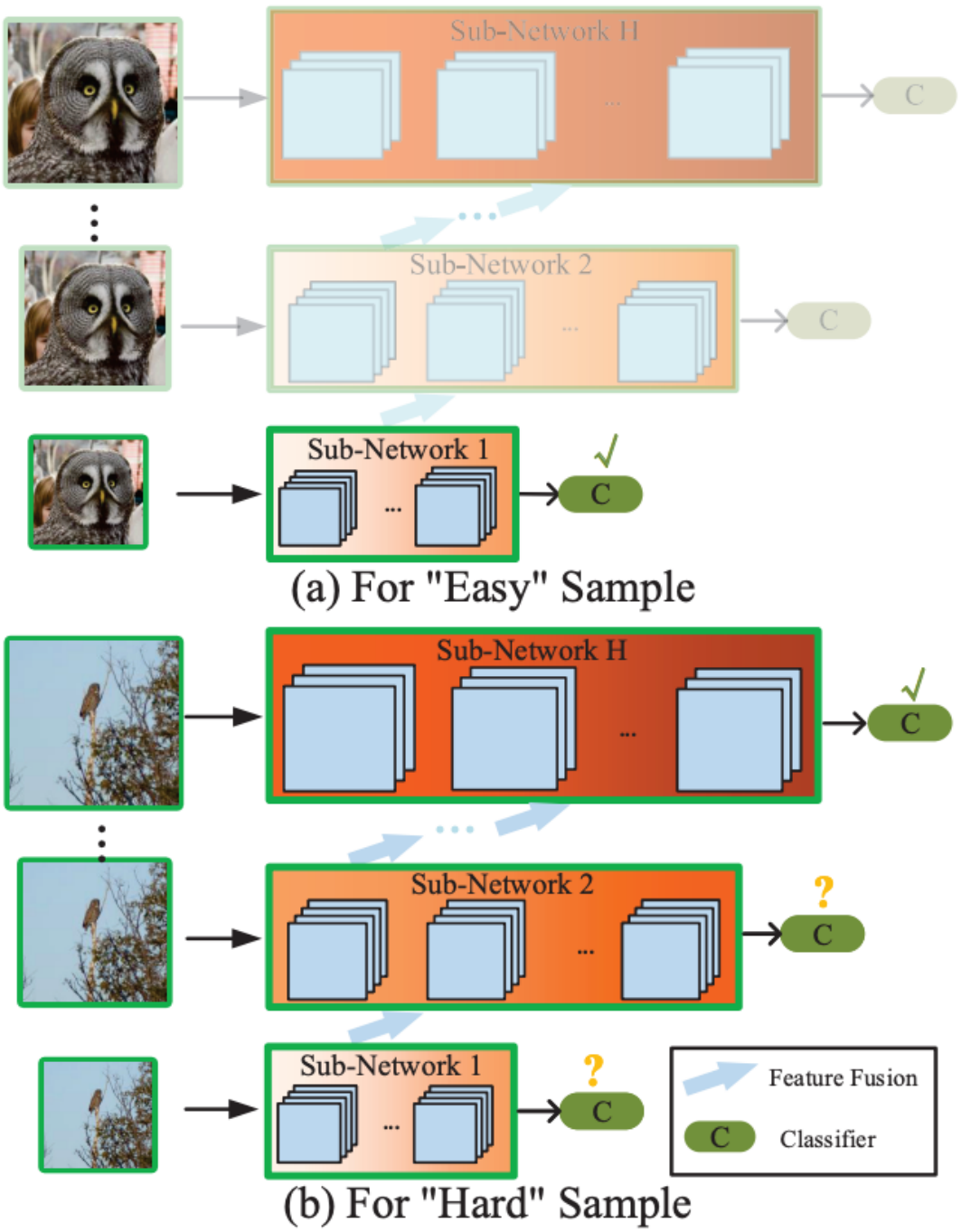


Figure 2.2: **RANet model architecture (from (Yang et al., 2020b)**

In the inference phase, the input data first passes through the shallow layers of the network and reaches an shallow intermediate classifier. At this point, a confidence

score, such as softmax score, is computed. If the score exceeds a predefined threshold, the network outputs the prediction and terminates the inference early. Otherwise, the input continues through deeper layers until it either meets the confidence requirement at another exit or reaches the final classifier.

The confidence score at exit k is defined as

$$c^{(k)} = \max_i \frac{\exp\left(z_i^{(k)}\right)}{\sum_j \exp\left(z_j^{(k)}\right)}, \tag{2.3}$$

where $z^{(k)} \in \mathbb{R}^C$ denotes the logits of the k-th classifier over C classes.

Early exiting is triggered if

$$c^{(k)} \geq \tau^{(k)}. \tag{2.4}$$

Where $\tau^{(k)}$ is the confidence threshold for exit k. This threshold can be determined using a validation set. Specifically, given a predefined computation budget, we search for the corresponding $\tau^{(k)}$ on the validation set and then use it during inference.

2.1.3 Experimental settings and Evaluation metrics for dynamic early-exiting networks

As a method that dynamically adjusts the network structure based on the input data, dynamic early-exiting networks focus on performance under all different computational budgets.

The effectiveness of methods in this domain is typically illustrated under two experimental settings:

Anytime prediction. Anytime prediction requires a model to produce a result within a given time budget (Grubb et al., 2012). In the context of dynamic early-exiting, this is typically evaluated by measuring the performance of classifiers at different exits. Such an evaluation is particularly important during training, since the goal is to improve the performance of individual classifiers without degrading that of others. Currently, the placement of intermediate classifiers in early-exiting

architectures is still largely based on empirical design. In practice, each additional classifier should have enough capacity to improve accuracy over the previous one, but it should not be too large, otherwise the overall efficiency gain becomes limited.

Budgeted batch classification. Another common evaluation setting is budgeted batch classification. In this scenario, a computational budget is specified for a batch of data, and the model dynamically selects exit points according to the difficulty of individual samples. Specifically, given a predefined budget, we first estimate the desired exit ratio at each classifier (e.g., about 20% of samples should exit at the first classifier). We then compute the corresponding confidence thresholds on the validation set to match this ratio. During inference, these thresholds are used as the criteria to decide whether to exit at a given classifier.

Floating Point Operations (FLOPs). To analyze the trade-off between accuracy and efficiency, FLOPs serve as an important metric in the study of dynamic early-exiting networks. FLOPs are widely used to measure the computational complexity of deep neural networks. Unlike the number of parameters, which reflects the memory footprint of a model, FLOPs quantify the actual amount of computation required for a single forward pass. This makes FLOPs a natural metric for comparing efficiency across models. In the context of dynamic early-exiting networks, the FLOPs of a sample depend on the exit point selected during inference, as earlier exits require fewer computations. As a result, FLOPs provide a practical way to analyze the trade-off between accuracy and computational cost under different evaluation settings such as anytime prediction and budgeted batch classification.

We compute the FLOPs using the official MSDNet PyTorch implementation available at

`https://github.com/kalviny/MSDNet-PyTorch`.

2.2 Monte Carlo Tree Search(MCTS)

MCTS is a decision-making algorithm that explores discrete spaces structured as trees. It evaluates actions through repeated simulations and concentrates computation on the most promising branches instead of exhaustively searching the entire tree.

Each iteration of MCTS consists of four steps. During selection, the algorithm traverses the tree using a multi-armed bandit strategy, typically the Upper Confidence bound for Trees (UCT), to balance exploration and exploitation. When an unexplored state is encountered, a new node is expanded. A simulation (rollout) is then performed from this state to obtain an outcome, and the result is backpropagated along the visited path to update node statistics. After many iterations, actions with higher visit counts or win rates are considered better decisions.

MCTS was first introduced to address challenges in computer Go (Kocsis et al., 2006). Go is widely recognized as the most challenging classic board game for AI, given its immense search space and the difficulty of position and move evaluation. MCTS rose to prominence in 2016, following AlphaGo's historic win against world Go champion Lee Sedol (Silver et al., 2016).

2.2.1 Computer Go

Go is one of the oldest board games in the world, with origins tracing back 3,000–4,000 years to ancient China. As one of the most challenging board games, Go has been a subject of artificial intelligence research for many years.

Mathematician I. J. Good wrote in 1965: 'Go on a computer? – In order to programme a computer to play a reasonable game of Go, rather than merely a legal game, it is necessary to formalise the principles of good strategy, or to design a learning programme. The principles are more qualitative and mysterious than in chess, and depend more on judgment. So I think it will be even more difficult to

programme a computer to play a reasonable game of Go than of chess.'

This description anticipated the trajectory of computer Go research. Before the introduction of Monte Carlo Tree Search (MCTS), computer Go programs relied on multiple modules to handle different aspects of the game, such as life-and-death problems, capturing races, opening theory, and endgame theory (Müller, 2002). With the introduction of MCTS, it quickly became the mainstream approach in computer Go, soon reaching the level of amateur 5-dan (Gelly et al., 2012), which represents a relatively high standard among human players. Nevertheless, they still did not reach the strength of professional Go players. MCTS is a tree search algorithm that abandons the modular design used in earlier Go programs and instead applies a general search procedure to all stages of the game. MCTS-based systems still employed hand-crafted heuristics to improve the efficiency and accuracy of the search (Helmbold, 2009), but they performed far better than non-MCTS approaches. Go programs then continued to improve through algorithmic refinements and more effective heuristics. The milestone of Go AI, AlphaGo (Silver et al., 2016), used deep neural networks to learn strategies, thereby eliminating the dependence on manually designed heuristics.

In the following, we present the well-known AlphaGo (Silver et al., 2016) and its subsequent version, AlphaZero (Silver et al., 2017).

2.2.2 AlphaGo and AlphaZero

AlphaGo stands as a milestone achievement in the development of computer Go. Because of the enormous search space of Go, traditional methods proved ineffective. The introduction of MCTS greatly advanced computer Go, yet programs still could not reach professional strength. Standard AI techniques struggled to evaluate the diversity of moves and lacked the creativity and intuition of human players.

A key innovation of AlphaGo was the replacement of manually designed heuristics with deep neural networks capable of learning Go knowledge directly from data.

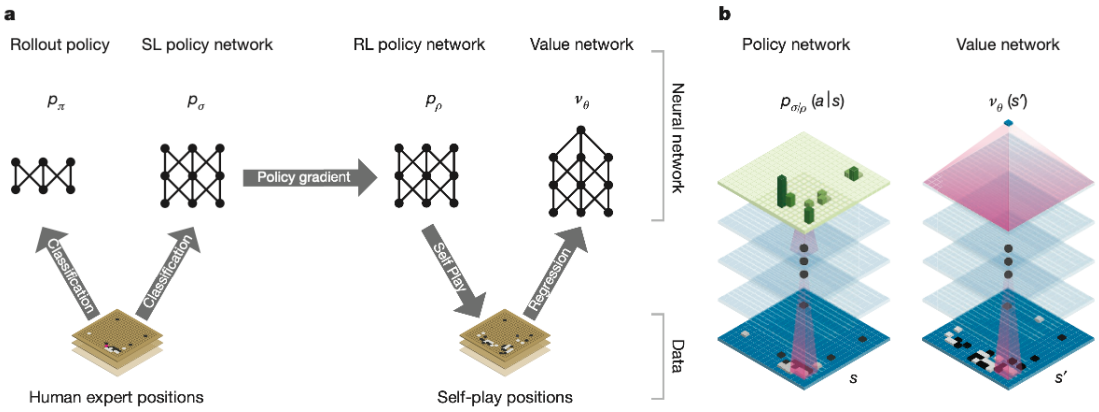


Figure 2.3: **AlphaGo's nerual networks architecture and training method. (from (Silver et al., 2016)**

As shown in Figure 2.3b, the architecture of AlphaGo consists of two deep neural networks: the policy network and the value network. The policy network outputs a probability distribution over all legal moves in a given position, thereby guiding the search toward promising actions. In contrast, the value network produces a single scalar that estimates the probability of winning from the current position for the player to move. Together, these two networks provide prior knowledge and evaluative feedback that significantly enhance the efficiency of MCTS.

AlphaGo's training proceeded in two phases. The model was first trained with supervised learning on expert games to reach a strong baseline. It was then refined by self-play, where deep reinforcement learning further improved its policy and value estimates. Figure 2.3a provides an illustration of the training process.

It is worth noting that, despite the strong performance of deep neural networks, they alone were not sufficient to reach professional strength. MCTS remained a crucial component of the AlphaGo algorithm.

AlphaGo employed the *Predictor + Upper Confidence Bound for Trees* (PUCT) algorithm as the tree policy in its Monte Carlo Tree Search (MCTS). PUCT extends the standard UCT (Upper Confidence Bound for Trees) (Kocsis et al., 2006) by incorporating a prior term provided by a predictor—in this case, the policy network.

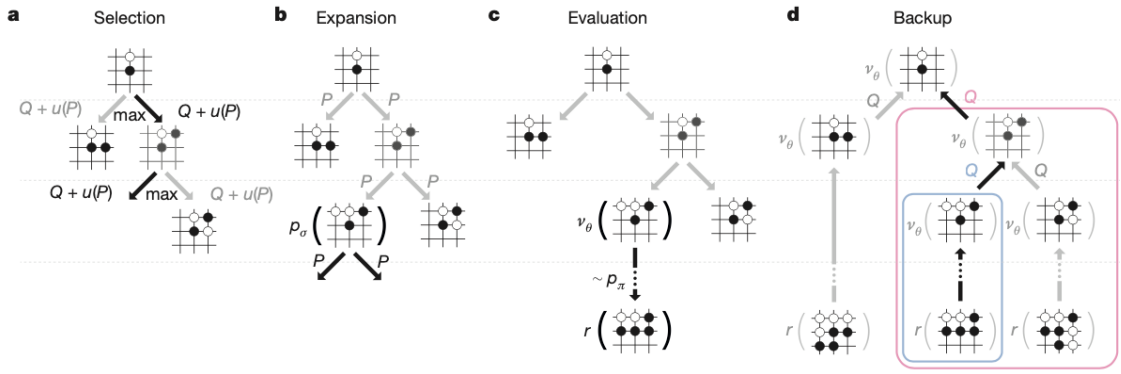


Figure 2.4: **AlphaGo's MCTS process.** (from (Silver et al., 2016))

The selection score for a state s and action a is defined as

$$U(s,a) = Q(s,a) + c_{\text{puct}} \cdot P(s,a) \cdot \frac{\sqrt{N(s)}}{1 + N(s,a)}, \tag{2.5}$$

where $Q(s,a)$ is the mean action value, $N(s)$ and $N(s,a)$ are visit counts for the parent and child nodes, $P(s,a)$ is the prior probability given by the policy network, and c_{puct} is an exploration constant.

In this formulation, the policy network contributes directly through $P(s,a)$, steering the search toward moves it considers promising. The value network contributes indirectly: at each newly expanded node, it outputs a scalar evaluation v estimating the win probability for the player to move. This estimate is propagated upward during backpropagation, updating $Q(s,a)$ along the path. Thus, while the policy network determines the prior exploration term in PUCT, the value network influences the exploitation term by shaping the empirical averages $Q(s,a)$. Together, these two networks allow PUCT to integrate learned predictions with search statistics, making AlphaGo's MCTS substantially stronger than earlier approaches based only on handcrafted heuristics or random rollouts.

As illustrated in Figure 2.4, each simulation proceeds by repeatedly selecting the child node with the largest $U(s,a)$ until a leaf node is reached. At this point, AlphaGo evaluates the leaf in two ways: (i) by performing a rollout using a fast policy until the end of the game and returning the outcome, and (ii) by applying

the value network to predict the win probability of the position. The final evaluation is a weighted average of these two estimates. This value is then backed up along the path to the root, updating the visit counts and action value estimates for all traversed nodes.

AlphaZero can be seen as a further improvement upon AlphaGo. In terms of architecture, AlphaZero uses a single backbone network shared by both the policy and value components, with separate policy and value heads added at the last layer. The model architecture is shown in Figure 2.5. This design is reasonable, as the policy and value predictions rely on largely overlapping features of the game state.

In contrast to AlphaGo, which was initially trained on human expert games, AlphaZero dispenses with human data and is trained solely from self-play experience. In the MCTS procedure, AlphaZero does not employ the rollouts used in AlphaGo. Instead, it directly relies on the value network's prediction to evaluate leaf nodes.

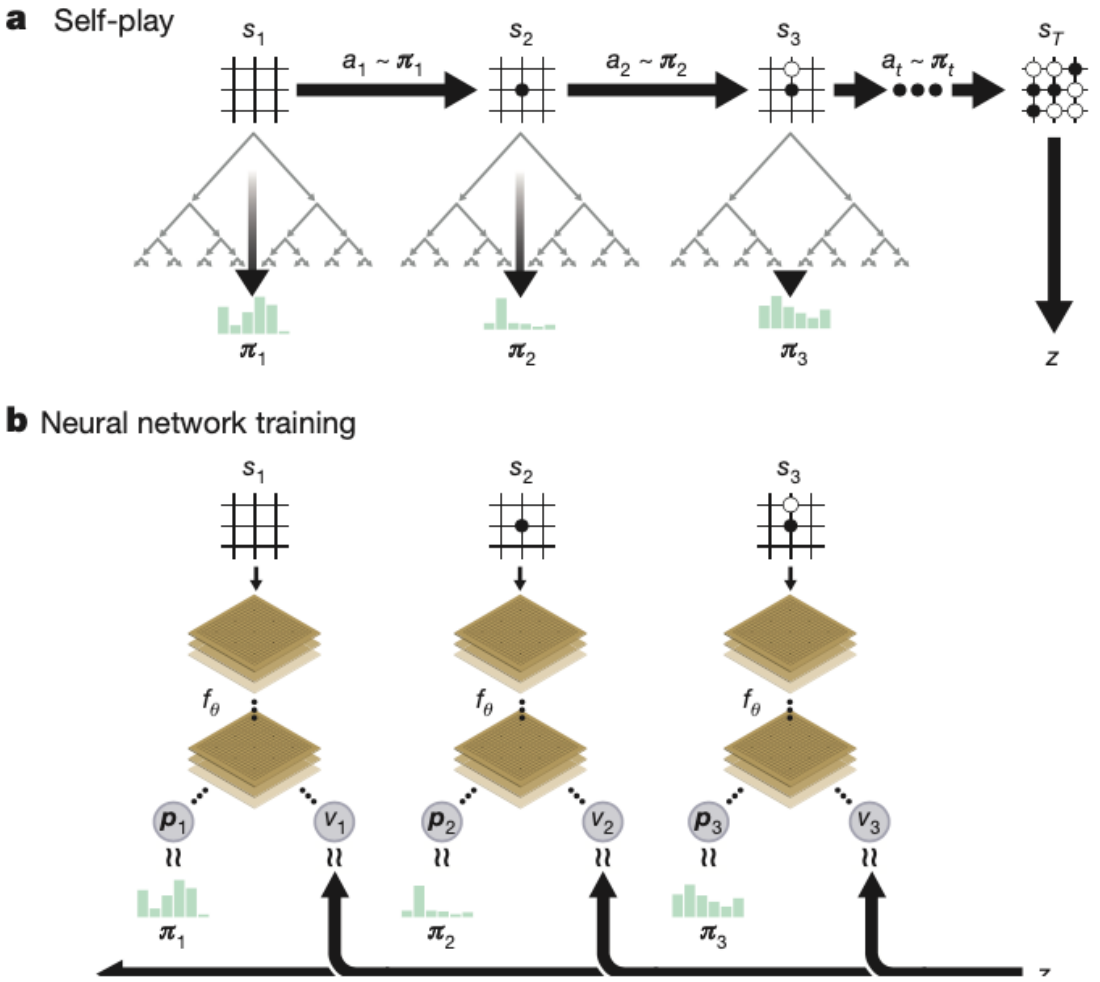


Figure 2.5: **AlphaZero's nerual networks architecture and training method. (from (Silver et al., 2017))**

Chapter 3

Related works

In this chapter, we present related work directly connected to our research problems, as well as contributions from other research areas that provide inspiration for our approach. Furthermore, we highlight how our method differs from these prior studies, thereby making our contributions more explicit.

First, in Section 3.1, we review related work on dynamic early-exiting networks. Section 3.1.1 focuses on existing studies concerning the training of dynamic early-exiting networks and highlights the remaining research gaps. Section 3.1.2 then discusses prior work on inference strategies for dynamic early-exiting networks.

We also review work related to our investigation of training mechanisms for dynamic early-exiting networks. In Section 3.2, we discuss research in the area of multi-task learning, and in Section 3.3, we review studies on overfitting.

We also explore work related to the collaboration of different classifiers during the inference phase of dynamic early-exiting networks. In Section 3.4, we review studies on conformal prediction, and in Section 3.5, we discuss research on multi-agent collaboration.

In Section 3.6, we discuss related work on MCTS, as it provides the methodological foundation for our subsequent exploration.

3.1 Dynamic early exiting networks

Early exiting exemplifies a dynamic neural network architecture that enables easy samples to be processed and output by intermediate classifiers. This technique has attracted widespread attention across domains such as computer vision (Huang et al., 2017b; Kouris et al., 2022; Yang et al., 2023; Yu et al., 2024; Niu et al., 2024; Jiang et al., 2024; Wang et al., 2021; Yang et al., 2020b), natural language processing (Bajpai et al., 2024; Zhou et al., 2020; Elbayad et al., 2020; Xin et al., 2021; Mangrulkar et al., 2022; Schuster et al., 2022), and multimodal tasks (Tang et al., 2023; Fei et al., 2022; Yue et al., 2024).

Some studies focus on the architectural design of early exiting models. BranchyNet (Teerapittayanon et al., 2016) places classifiers at multiple depths to enable early decisions and reduce computation. MSDNet (Huang et al., 2017b) provides classifiers with features from different levels to handle varying input complexities. RANet (Yang et al., 2020b) uses conditional activation to process high-resolution data only when necessary, balancing accuracy and efficiency. Additionally, ZTW (Wołczyk et al., 2021) emphasizes feature reuse, where features extracted by shallow classifiers are reutilized by deeper ones, effectively reducing errors not addressed by earlier classifiers. Beyond architecture, several studies have investigated exit strategies. For example, reinforcement learning has been utilized to learn effective exit policies (Huang et al., 2017a), while learned policy networks have been developed to optimize the exit decision (Chen et al., 2020).

3.1.1 Training process of dynamic early exiting networks

Training dynamic early-exiting models presents unique challenges due to gradient conflicts among different exits, which compete to update shared parameters (Sun et al., 2022). DFS (Gong et al., 2024) mitigates gradient conflicts through feature partitioning. Meta-learning techniques (Sun et al., 2022; Han et al., 2022) have also been explored to dynamically weight gradients from different exits, thereby reducing

the gradients conflict. However, they did not consider whether the gradients provided by different classifiers actually benefit the model. We focus on identifying and leveraging gradients that are truly beneficial to the model's performance. A detailed presentation of our study on this topic is provided in Chapter 4.

3.1.2 Inference process of dynamic early exiting networks

PABEE (Zhou et al., 2020) introduces a simple yet effective early exiting mechanism based on output stability. It defines a patience score that monitors the consistency of predictions across successive classifiers. When the same output is observed over a predefined number of consecutive layers, the network triggers an early exit. This strategy implicitly aggregates information from multiple classifiers and can improve prediction accuracy. However, the exit criterion is often overly strict, requiring multiple identical outputs before termination. As a result, it tends to delay exiting, leading to a suboptimal trade-off between accuracy and efficiency.

In contrast, our method directly leverages the predictions of earlier classifiers to enhance the performance of the current classifier, without modifying the existing exit mechanism. A detailed description of this approach is presented in Chapter 5. As a result, it achieves a more favorable trade-off between accuracy and efficiency. Moreover, our aggregation strategy goes beyond simple majority voting schemes by incorporating richer information than just the predicted labels.

3.2 Multi-task learning

Compared to training early-exiting networks, multi-task learning has received more attention. The primary challenge in multi-task learning is managing gradient conflicts between different tasks (Yu et al., 2020), a problem that is also prevalent in early-exiting networks. Techniques from multi-task learning, such as knowledge distillation (Xu et al., 2023; Ghiasi et al., 2021), feature partitioning (Ding et al., 2023a) and gradient selection (Liu et al., 2021), have also been applied to the training

of early-exiting models (Li et al., 2019; Phuong et al., 2019; Sun et al., 2022; Han et al., 2022; Gong et al., 2024; Addad et al., 2025). The state-of-the-art methods for training early-exiting networks use meta-learning to adaptively weight the losses of different classifiers. These approaches essentially follow the linear scalarization framework from multi-task learning (Xin et al., 2022; Hu et al., 2024), where all loss terms are combined using a weighted sum. In contrast to previous methods that focus on resolving gradient conflicts, in this thesis, our work examines whether all gradients are necessary in the first place. As such, our approach is complementary to conflict-resolution techniques and can be integrated with them. Our approach can be integrated with linear scalarization, but in contrast to meta-learning strategies, it offers a simpler alternative by directly deriving gradient weights from damping neuron values, thereby avoiding the extra backpropagation steps and reducing training overhead.

3.3 Overfitting

A key challenge in neural networks is inadequate generalization, particularly in adversarial learning (Kim et al., 2021) and generative models (Loaiza-Ganem et al., 2022). Many studies address overfitting using regularization techniques such as dropout (Srivastava et al., 2014) and label smoothing (Szegedy et al., 2015). Early stopping (Prechelt, 2002) is a widely used technique to prevent overfitting, typically identifying the best epoch on the validation set and stopping training before overfitting occurs. Existing early-exiting baselines already incorporate standard overfitting mitigation techniques, such as data augmentation, regularization, and learning rate decay. However, some traditional methods are not directly applicable, since classifiers at different exits are trained to different extents. For example, early stopping is unsuitable for early-exiting architectures because different classifiers reach their optimal stopping points at different times. In Chapter 4, we focus on the unnecessary gradient problem during the training of early-exiting networks.

Although overfitting is related to this issue, we observe that unnecessary gradients remain significant even when standard overfitting mitigation techniques are applied. In this thesis, our work specifically targets early-exiting architectures, addressing unnecessary gradients while jointly considering the distinct roles and training states of intermediate classifiers.

3.4 Conformal Prediction

Conformal prediction (CP) provides a practical framework for uncertainty quantification by constructing prediction sets based on model confidence. In classification, CP uses a calibration set to compute a softmax-based threshold for a desired coverage level. At test time, it forms a prediction set by including labels whose cumulative softmax scores exceed this threshold, ensuring the true label is likely to be contained. Originating from the work of Vovk et al. (Vovk et al., 2005), CP operates under minimal assumptions—requiring only data exchangeability—and has been widely applied in classification, regression, and other machine learning tasks.

In classification, CP produces adaptive prediction sets that reflect the model's confidence, typically based on softmax-derived nonconformity scores (Angelopoulos et al., 2021; Shafer et al., 2008). As a simple post hoc method, CP is particularly appealing in the deep learning era: it requires no additional training and utilizes only the outputs of a pretrained model. This makes it an efficient and scalable approach to uncertainty estimation in modern deep neural networks.

During inference in early-exiting networks, confidence estimation is a critical factor. In our multi-classifier collaborative setting, accurate confidence leads to better performance. Specifically, in our Bayesian update framework, likelihood prediction requires more fine-grained information than what is typically provided by standard conformal sets. As more conditioning factors are introduced, the amount of calibration data available for each condition decreases, which can significantly degrade the reliability of conformal prediction (Ding et al., 2023b). Prior work

by Angelopoulos et al. has shown that even when the conformal set contains only a single label, the method can still perform well (Angelopoulos et al., n.d.). Building on this insight, we adapt conformal prediction to our task by using only high-confidence, single-label conformal sets to enhance likelihood estimation.

3.5 Multi-agent collaboration

Multi-agent collaboration has been extensively studied in the context of reinforcement learning (RL) and other distributed systems. At its core, multi-agent collaboration involves multiple agents working together to achieve a shared goal, often requiring effective communication, coordination, and learning of joint policies.

In particular, in decision-centered teamwork, agents aggregate their opinions to take a joint team decision. Marcolino et al. (Marcolino et al., 2013; Marcolino et al., 2014) have shown that when there is sufficient diversity among agents, a voting-based approach can enable a group of relatively weaker agents to outperform a single, stronger agent. This method leverages the diversity within the group, allowing the collective decision-making process to mitigate individual weaknesses and harness the strengths of each agent, resulting in more robust and effective outcomes.

Regarding the Bayesian approach, a study has employed Bayesian updating in multi-agent collaboration to detect disruptive agents (Carmo Alves et al., 2024). By iteratively updating beliefs based on observed behaviours, the system can identify and isolate potentially harmful individuals, enhancing overall group stability and performance. However, that work was aimed at a different coordination task, and did not tackle the decision-centered teamwork challenge as we do in this work. Additionally, they could estimate likelihoods from the outcomes of a Monte Carlo Tree Search process, which is not applicable in deep learning tasks. However, the application of multi-agent collaboration within dynamic early-exiting networks remains underexplored. In our work, we treat the collaboration among multiple classifiers in an early-exiting network as a multi-agent cooperation problem. In

large-scale deep learning settings, tasks are often more complex, which poses significant challenges for likelihood estimation in Bayesian update methods. Our thesis addresses this issue by exploring multi-agent collaboration in dynamic early-exiting networks.

3.6 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a tree-structured decision-making algorithm. It was initially introduced to address challenges in computer Go (Kocsis et al., 2006). MCTS quickly advanced the strength of computer Go programs, raising their level from amateur kyu ranks to dan-level play within a short period of time (Gelly et al., 2012).

What truly brought MCTS to prominence was its role in AlphaGo, developed by Google DeepMind, which defeated the world Go champion Lee Sedol in 2016 (Silver et al., 2016). AlphaGo exemplifies the strength of combining MCTS with deep reinforcement learning, where neural networks provide effective policy and value estimates to guide the search, greatly reducing the effective search space while improving decision quality.

Gao et al. (2017) proposed MoHex-CNN, which outperformed the previous state-of-the-art MoHex 2.0 (Huang et al., 2013). MoHex-CNN integrates a deep convolutional policy network with MCTS to guide move prediction. (Świechowski et al., 2018) extended the combination of MCTS and deep neural networks to the domain of Hearthstone, a modern card game. Unlike Go, Hearthstone is a game of imperfect information, as players cannot observe the opponent's hand or predict their own future draws.

Despite the strong performance of AlphaGo (Silver et al., 2016) and AlphaZero (Silver et al., 2017), several subsequent works have sought to improve upon them. Yang et al. (2020a) focused on the komi problem in Go, where the player with the black stones receives a compensation to offset the advantage of moving first. Their

work designed networks capable of progressively estimating the appropriate komi value, which is of great importance in handicap games.

In recent years, increasing attention has been directed toward the deep neural network component of the AlphaZero algorithm, such as its security (Yang et al., 2020a). However, the acceleration of deep neural networks within this framework remains largely unexplored. To the best of our knowledge, we are the first to deploy dynamic neural networks—a widely used acceleration technique in deep learning—within the Policy–Value MCTS architecture.

Chapter 4

Diminishing Non-important Gradients for Training Dynamic Early-Exiting Networks

Dynamic early-exiting is an effective mechanism to improve computation efficiency. By adding classifiers to intermediate layers of deep learning networks, early exiting networks can terminate the inference early for easy samples, thus reducing the average inference time. Gradient conflicts between different classifiers are a key challenge in training early-exiting networks. However, current state-of-the-art methods focus solely on the trade-off between gradients, without evaluating whether these gradients are actually necessary.

In this chapter, we approach the problem from the perspective of gradient necessity in training dynamic early-exiting networks. Based on this view, we propose an adaptive damping strategy that selectively suppresses non-essential gradients during training according to both data samples and classifiers. By adding a damping neuron to the last fully connected layer of each classifier and using our proposed damping loss, our approach effectively reduces gradients that are unlikely to be beneficial. Moreover, we propose power-sqrt loss to concentrate the gradients of damping neurons on classifiers that exhibit relatively better training performance.

Experiments on CIFAR and ImageNet demonstrate our proposed method gains significant accuracy improvement for all classifiers with negligible computation increases.

4.1 Introduction

Although deep neural networks have achieved remarkable success across various tasks (Krizhevsky et al., 2012; Simonyan et al., 2014; He et al., 2016; Huang et al., 2019), their high computational costs limit their application on resource-constrained devices. Many efforts have been made to improve the inference efficiency of deep neural networks such as network pruning (LeCun et al., 1989; Yang et al., 2021), weight quantization (Hubara et al., 2016; Han et al., 2015), and lightweight network architecture design (Howard et al., 2017; Zhang et al., 2018; Sandler et al., 2018). While these efficient models achieve competitive accuracy, many challenging data samples still demand the use of larger networks (Huang et al., 2017b; Lin et al., 2017). By exploiting this fact, dynamic networks (Han et al., 2021), which perform a data-dependent inference procedure by dynamically adjusting the network structure, have attracted considerable research interest. As a typical dynamic network, early-exiting attaches multiple intermediate classifiers (early exits) to the network. In the inference stage, early-exiting networks adaptively terminate inference when an early exit satisfies the predefined exiting criterion such as the confidence score of softmax (Huang et al., 2017b) or according to a learned policy (Chen et al., 2020).

Unlike conventional deep neural networks, early-exiting networks have multiple exits that share parameters. This shared structure causes interference among exits. Gradients from different exits often conflict during training (Sun et al., 2022). The current state-of-the-art methods of training early exiting networks adopts a meta-learning approach, where a meta-network is used to learn the weights of individual gradients during the training of early exiting networks (Han et al., 2022; Sun et al., 2022). This approach belongs to the category of linear scalarization (Hu et al., 2024),

a mainstream method in multi-task learning. It addresses the trade-off between gradients from different tasks during the training of early exiting networks, thereby mitigating the issue of gradient conflicts.

Despite the advances, current meta-learning methods (Han et al., 2022; Sun et al., 2022) do not fully consider whether the gradients from each classifier are actually necessary for its performance. During neural network training, overfitting can occur, where unnecessary gradients not only fail to improve but may even degrade performance. The parameter space wasted by these gradients could be better allocated to improve other classifiers. Moreover, standard early-stopping methods (Prechelt, 2002) are not ideal for early-exiting networks, as different classifiers require different stopping times. Thus, evaluating the necessity of gradients during training plays a key role on optimizing early-exiting networks, ensuring that only beneficial gradients are preserved.

To address this issue, we propose a novel training strategy that adaptively diminishes gradients based on the classifier's performance with the current data sample. As illustrated in Figure 4.1, the classifier diminishes the gradient when it already performs well on a given sample, as indicated by a sufficiently high softmax score. Specifically, when the classifier already achieves a high softmax score on the current data sample, our damping neuron gets assigned a higher gradient, effectively preventing unnecessary gradients. Conversely, when the classifier's performance is suboptimal and the softmax score is low, we introduce only a minimal gradient to the damping neuron, limiting its influence at this stage. Moreover, we introduce the *power-sqrt* loss function, which refines the distribution of gradients in damping neurons by concentrating them on the classifiers that exhibits superior performance relative to others. This strategy enables a more effective damping mechanism by jointly considering the status of all classifiers, thereby further improving the training process across the network. Additionally, we leverage the values of the damping neuron to assign dynamic weights to different classifiers, demonstrating the compatibility of our method with the linear scalarization approach.

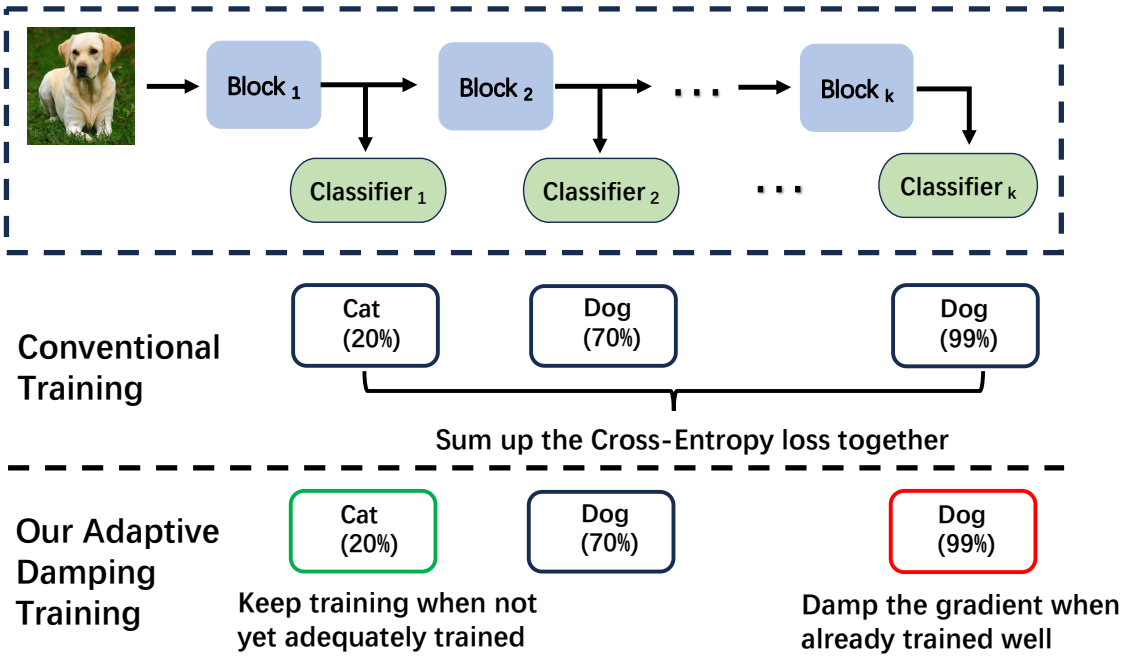


Figure 4.1: **Our adaptive damping training strategy.** During training, different classifiers evolve at different rates. Instead of summing the cross-entropy losses, our method applies adaptive gradient damping, reducing the influence of already well-trained classifiers.

Our main contributions are summarized as follows:

1. We introduce a novel adaptive damping mechanism that dynamically reduces unnecessary gradients during training, improving overall performance;

2. We further propose the *power-sqrt* loss, which jointly considers the training status of all classifiers to more effectively determine the appropriate damping gradient magnitude;

3. We further demonstrate the compatibility of our method with the current linear scalarization approaches;

4. We perform extensive experiments on CIFAR (Krizhevsky et al., 2009) and ImageNet (Deng et al., 2009) datasets demonstrating the superiority of our

method, which achieves a significant improvement in accuracy with negligible complexity increases.

4.2 Methodology

In this section, we first present conventional early exiting networks and the current training methodologies. Then, we provide a detailed explanation of our proposed approach.

4.2.1 Preliminaries

4.2.1.1 Early Exiting Networks.

Contrasting with standard deep learning models, K-exit early exiting networks integrate $K-1$ classifiers at various layers within the original deep learning architecture (see Fig. 4.1).

The prediction for the i-th input by classifier $f^{(k)}$ is defined as

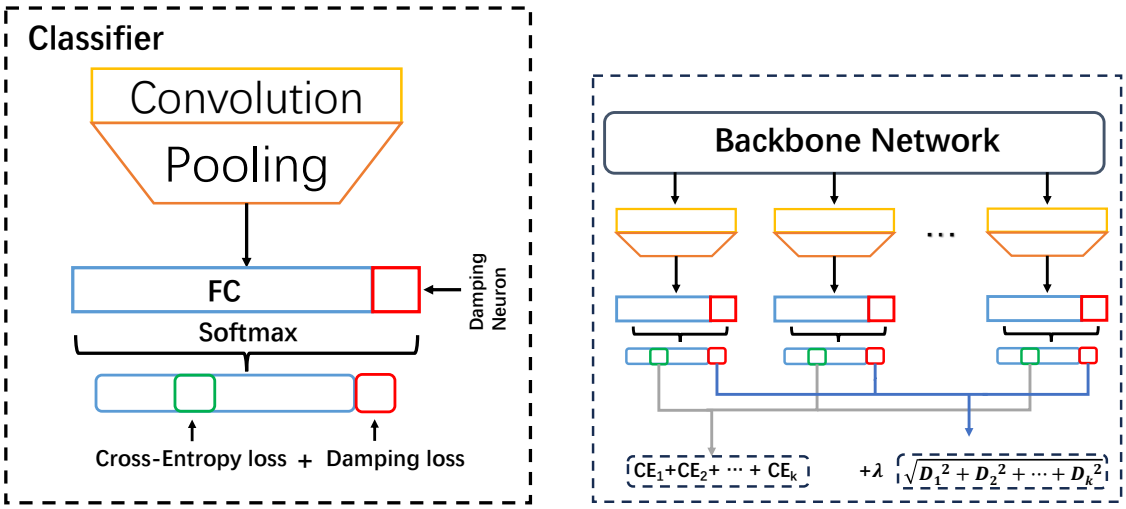
$$p_i^{(k)} = f^{(k)}(\mathbf{x}_i, \theta^{(k)}), \tag{4.1}$$

where $\mathbf{x}_i$ denotes the input data sample and $\theta^{(k)}$ the parameters of the k-th classifier. Note that these sub-networks share a portion of their parameters.

Early exiting networks enable dynamic inference. They use an exit mechanism based on a confidence score. This score is typically the maximum output of the classifier's softmax result (Huang et al., 2017b; Yang et al., 2020b). When a classifier's score reaches a predefined threshold, the model stops the inference process at this classifier, saving computational resources.

4.2.1.2 Training Strategies of Early Exiting Networks.

The conventional training strategy for early exiting networks aggregates the losses from all classifiers, with all classifiers trained simultaneously from beginning to end (Huang et al., 2017b; Yang et al., 2020b).



(a) Damping loss (b) Power-sqrt loss

Figure 4.2: **The damping neuron and power-sqrt loss.** (a) We add a non-class neuron (red square) to the classifier's FC layer and assign it a small gradient to adaptively damp gradients in the cross-entropy part. (b) For joint training, we sum cross-entropy losses across classifiers; for damping neurons we apply sum-of-squares followed by square root to focus damping gradients on relatively better-performing classifiers.

The total loss is defined as

$$\mathcal{L} = \frac{1}{S} \sum_{k=1}^{K} \sum_{i=1}^{S} \mathcal{L}_i^{(k)}, \tag{4.2}$$

where

$$\mathcal{L}_i^{(k)} = \mathrm{CE}\left(f^{(k)}(\mathbf{x}_i, \theta^{(k)}), y_i\right) \tag{4.3}$$

denotes the cross-entropy loss for the k-th classifier on the i-th data sample. Here, K is the number of classifiers, S the number of training samples, and y_i the ground truth label.

Training early-exiting networks often encounters gradient conflicts among classifiers. Existing methods focus on resolving these conflicts by balancing different gradients but overlook a critical question: Are all gradients necessary? Most approaches assume that every gradient should be incorporated into the trade-off,

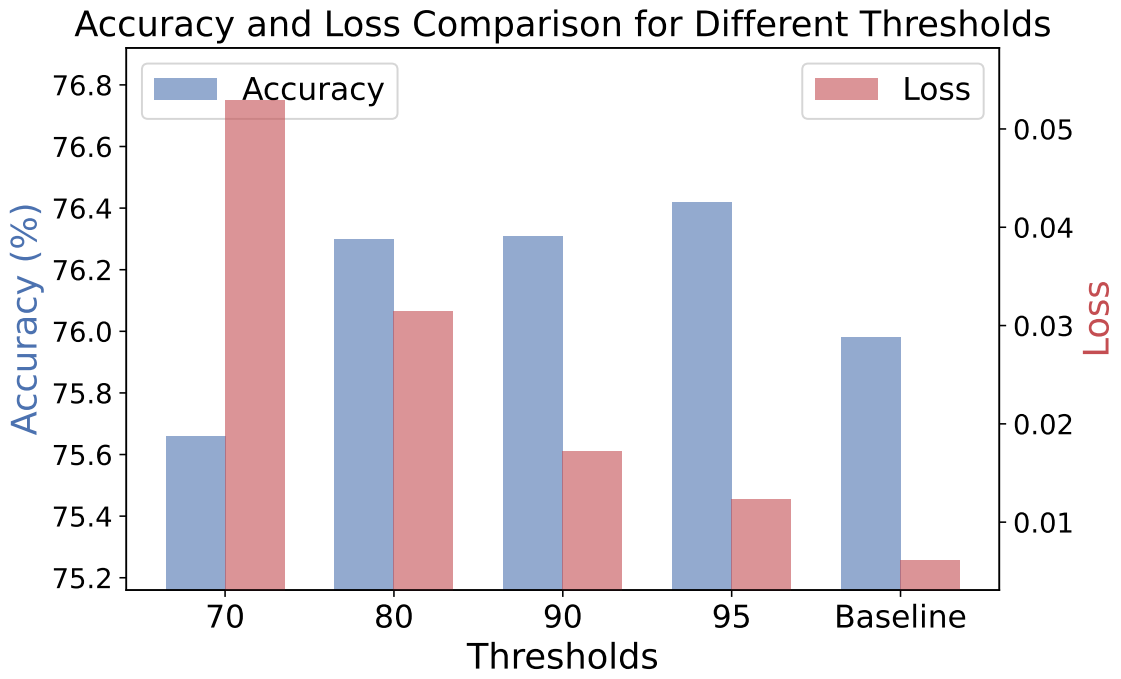


Figure 4.3: The relationship between softmax values and gradient necessity. As softmax increases, gradients are more likely to be unnecessary or even harmful.

without considering whether these gradients are necessary.

4.2.2 Should All Gradients Be Considered in the Trade-off?

A recent study (Wei et al., 2022) suggests that cross-entropy loss drives the softmax value to continue increasing even when it has already reached a high value, which may not always benefit model performance.

To investigate this effect, we train only the deepest classifier in MSDNet. We use the same architecture as in Table 4.1 and conduct experiments on CIFAR-100. The results are shown in Figure 4.3.

We conducted experiments on the MSDNet architecture, training only the deepest classifier to assess whether the gradients it receives improve its own performance. We introduced a threshold based on the softmax value corresponding to the correct label during training. Specifically, we discarded gradients when the

softmax value exceeded a given threshold. For instance, a threshold of 90 means that only gradients where the correct label's softmax value is below 90% are considered.

We recorded the final converged accuracy and loss under different thresholds. Our experimental results show that as more gradients are included, the final converged loss decreases. Maintaining a higher loss allows for more optimization flexibility, which provides additional capacity for other classifiers in early-exiting architectures. Despite this, the performance does not continue to improve significantly. Specifically, we observe that gradients corresponding to softmax values between 70% and 80% significantly improve model performance, while those in the 80%–95% range contribute little, and those in the 95%–100% range even degrade performance.

While including more gradients reduces the final loss, accuracy does not always improve. This suggests that some gradients are **unnecessary**, as they do not benefit the classifier's own performance. When softmax values are already high, cross-entropy loss continues to push them further, potentially leading the model to learn features that are less generalizable and primarily beneficial to a small subset of data samples.

4.2.3 Damping loss

In order to tackle the aforementioned issues, we propose a dynamic damping mechanism to diminish these non-beneficial gradients for training early-exiting.

We have modified the Cross-Entropy loss by adding our novel damping neuron, which does not represent any specific class in the classification task as shown in Figure 4.2a. The Softmax function is jointly applied to the outputs from the original fully connected layer and our newly introduced damping neuron:

$$F_\theta(x)[j] = \frac{\exp\left(f_\theta(x)[j]\right)}{\sum_{n=1}^{N+1} \exp\left(f_\theta(x)[n]\right)}, \tag{4.4}$$

where N is the number of classes in the classification task, $f_\theta(x)[j]$ denotes the output of the neurons from both the fully connected layer and our added damping

neuron, and $F_\theta(x)[j]$ is the output of the Softmax function. We add an additional neuron, increasing the total number of neurons to $N+1$. Unlike traditional Cross-Entropy loss, which solely focuses on increasing the neuron value of the correct label, our damping loss additionally provides a small gradient to the $N+1$ neuron, which damps the gradient from the cross-entropy part:

$$\min_\theta \mathbb{E}_{t(x)} \left[-\log F_\theta(x)[y] \right] + \lambda \mathbb{E}_{t(x)} \left[-\log F_\theta(x)[N+1] \right], \tag{4.5}$$

where $t(x)$ denotes the training dataset, y presents the correct label, and λ is a hyperparameter, typically assigned a small value to ensure that the majority of our loss remains focused on the classification task. The first term of our loss function is the Cross-Entropy loss, while the second term is designed to encourage an increase in the $N+1$-th neuron for any data sample. Additionally, this gradient counteracts the effect of the gradient from the Cross-Entropy term, thereby dampening its influence. Our method introduces only one additional neuron to the fully connected layer, thereby rendering the extra computational demand negligible.

Our in-depth theoretical analysis demonstrates several advantages of our damping loss:

1. The damping component of our loss generates stronger inhibitory gradients as the softmax value increases, effectively suppressing unnecessary gradients that are more likely to appear at higher softmax values.

2. Unlike Cross-Entropy loss, our damping loss does not continuously encourage the softmax value of the correct label to approach 1, preventing the generation of potentially unnecessary gradients.

3. Our damping loss does not perpetually promote the increase of the damping neuron's softmax value, preventing adverse effects on the model's performance.

In the following, we first present a proof sketch and then provide the detailed gradient derivation.

Proposition 1. The gradient of the damping component with respect to the neuron corresponding to the correct label $\frac{\partial -\log F_\theta(x)[N+1]}{\partial f_\theta(x)[y]}$ is proportional to its softmax value $F_\theta(x)[y]$. When the neuron $F_\theta(x)[y]$ achieves $\frac{1}{1+\lambda}$, the gradient from our damping loss $g \geq 0$.

Proof sketch. We demonstrate our proposition by analyzing the gradients generated by our damping loss for each neuron in the fully connected layer. Because our damping loss comprises two components, each neuron in the fully connected layer is influenced by gradients from both the Cross-Entropy component and the damping component.

For the neuron corresponding to the correct label, the gradient from the damping component of our damping loss is $\lambda F_\theta(x)[y]$, which is positive and thus encourages a reduction in the neuron value. This gradient is proportional to the softmax value of the neuron corresponding to the correct label, meaning that for larger softmax values, where gradients are more likely to be unnecessary, it generates a stronger opposing effect to suppress further increases.

Moreover, the gradient from the cross-entropy component of the loss is $F_\theta(x)[y]-1$. Thus, the total gradient for the neuron corresponding to the correct label is $(\lambda+1)F_\theta(x)[y]-1$. Our damping loss does not continuously encourage the softmax value of the correct label to grow. Once it reaches $\frac{1}{1+\lambda}$, the overall gradient becomes positive, preventing further encouragement of its growth. □

Detailed Proof of gradient. Our damping loss's gradient has two components: the cross-entropy part $\bigtriangledown -\log F_\theta(x)[y]$ and the gradient of our damping neuron $\bigtriangledown -\log F_\theta(x)[N+1]$. We demonstrate the gradients transmitted from the loss of the damping neuron part to each neuron as follows:

$$\frac{\partial -\log F_\theta(x)[N+1]}{\partial f_\theta(x)[i]} \tag{4.6}$$

According to the chain rule:

$$= -\frac{1}{F_\theta(x)[N+1]} \times \frac{\partial F_\theta(x)[N+1]}{\partial f_\theta(x)[i]} \tag{4.7}$$

$$= -\frac{1}{F_\theta(x)[N+1]} \times \frac{\partial \frac{\exp(f_\theta(x)[N+1])}{\sum_{j=1}^{N+1}\exp(f_\theta(x)[j])}}{\partial f_\theta(x)[i]} \tag{4.8}$$

According to the quotient rule:

$$= -\frac{1}{F_\theta(x)[N+1]} \times (\frac{\partial \exp\left(f_\theta(x)[N+1]\right)}{\partial f_\theta(x)[i]} \times \frac{1}{\sum_{j=1}^{N+1}\exp\left(f_\theta(x)[j]\right)} \tag{4.9}$$
$$- \exp\left(f_\theta(x)[N+1]\right) \times \frac{\partial \sum_{j=1}^{N+1}\exp\left(f_\theta(x)[j]\right)}{\partial f_\theta(x)[i]} \times \frac{1}{\left(\sum_{j=1}^{N+1}\exp\left(f_\theta(x)[j]\right)\right)^2})$$

$$= -\frac{1}{F_\theta(x)[N+1]} \times (\frac{\partial \exp\left(f_\theta(x)[N+1]\right)}{\partial f_\theta(x)[i]} \times \frac{1}{\sum_{j=1}^{N+1}\exp\left(f_\theta(x)[j]\right)} \tag{4.10}$$
$$- \exp\left(f_\theta(x)[N+1]\right) \times \frac{\partial \exp\left(f_\theta(x)[i]\right)}{\partial f_\theta(x)[i]} \times \frac{1}{\left(\sum_{j=1}^{N+1}\exp\left(f_\theta(x)[j]\right)\right)^2})$$

$$= -\frac{1}{F_\theta(x)[N+1]} \times (\frac{\partial \exp\left(f_\theta(x)[N+1]\right)}{\partial f_\theta(x)[i]} \times \frac{1}{\sum_{j=1}^{N+1}\exp\left(f_\theta(x)[j]\right)} \tag{4.11}$$
$$- \exp\left(f_\theta(x)[N+1]\right) \times \exp\left(f_\theta(x)[i]\right) \times \frac{1}{\left(\sum_{j=1}^{N+1}\exp\left(f_\theta(x)[j]\right)\right)^2})$$

$$= -\frac{1}{F_\theta(x)[N+1]} \times (\frac{\partial \exp\left(f_\theta(x)[N+1]\right)}{\partial f_\theta(x)[i]} \times \frac{1}{\sum_{j=1}^{N+1}\exp\left(f_\theta(x)[j]\right)} \tag{4.12}$$
$$- F_\theta(x)[N+1] \times F_\theta(x)[i])$$

when $i \neq N+1$, $\frac{\partial \exp(f_\theta(x)[N+1])}{\partial f_\theta(x)[i]} = 0$,

$$eq.(7) = -\frac{1}{F_\theta(x)[N+1]} \times -F_\theta(x)[N+1] \times F_\theta(x)[i]$$

$= F_\theta(x)[i]$

when $i = N+1$, $\frac{\partial \exp(f_\theta(x)[N+1])}{\partial f_\theta(x)[i]} = \exp\left(f_\theta(x)[N+1]\right)$,

$$eq.(7) = -\frac{1}{F_\theta(x)[N+1]} \times (F_\theta(x)[N+1] - (F_\theta(x)[N+1])^2)$$

$= F_\theta(x)[N+1] - 1$

Thus, for the damping neuron, where $i = N+1$, the gradient from damping component is $F_\theta(x)[N+1] - 1$. For all other neurons, the gradient is $F_\theta(x)[i]$.

The same derivation applies to the gradient of the cross-entropy component as well.

The gradients from the cross-entropy component are $F_\theta(x)[y] - 1$ for the neuron corresponding to the correct label, and $F_\theta(x)[i]$ for the other neurons.

Proposition 2. Our damping loss does not encourage the continuous increase of the damping neuron's softmax value $F_\theta(x)[N+1]$. Once the damping neuron's softmax $F_\theta(x)[N+1] \geq \frac{\lambda}{1+\lambda}$, the gradient of our damping loss $g \geq 0$.

Proof sketch. Similar to the proof of Proposition 1, we analyze the gradients from the two components of the damping loss. The damping neuron receives a gradient from the cross-entropy component as $F_\theta(x)[N+1]$ and from the damping component as $\lambda(F_\theta(x)[N+1]-1)$. Hence, the total gradient is $(1+\lambda)F_\theta(x)[N+1]-\lambda$. Once it reaches $\frac{\lambda}{1+\lambda}$, no further gradient encourages its growth. □

4.2.4 Power-sqrt loss

Equation 4.5 details our loss function for each classifier. In an early-exiting architecture, which incorporates multiple classifiers, the total loss function is presented as follows:

$$\min_{\theta} \sum_{k=1}^{K} \mathbb{E}_{p(x)} \left[-\log F_{\theta}^{(k)}(x)[y] \right] + \sum_{k=1}^{K} \lambda^{(k)} \mathbb{E}_{p(x)} \left[-\log F_{\theta}^{(k)}(x)[N+1] \right] \tag{4.13}$$

where k presents the index of the classifier. Our damping loss requires different hyperparameters for each classifier, as their optimal values vary, making fine-tuning a challenging task. We found that uniformly setting the hyperparameters $\lambda^{(k)}$ for all classifiers does not yield good performance. While some classifiers improved, others declined. Manually adjusting $\lambda^{(k)}$ for each classifier can enhance performance, but the vast number of possible combinations complicates tuning.

We observed that the values of damping neurons, specifically the damping neuron post-Softmax, are consistently higher in deeper classifiers. A detailed analysis is provided in the Experiment section. This indicates that deeper classifiers are more likely to trigger the damping mechanism, thereby freeing up resources for other classifiers. Additionally, we noted that a higher λ value correlates with reduced accuracy in shallow classifiers. Hence, assigning the same λ value to shallow classifiers as to deeper ones adversely affects their performance.

Moreover, the damping mechanism should consider all classifiers collectively. Specifically, when some classifiers perform better than others, the damping mechanism should prioritize these, rather than overly diminishing the gradients of underperforming classifiers. This strategy reallocates resources to underperforming classifiers, optimizing overall performance. Thus, a joint-damping mechanism is essential for training classifiers effectively.

Motivated by these findings, we introduce the power-sqrt loss, which dynamically adjusts the hyperparameter λ for each classifier, as shown in Figure 4.2b. We modify the second term of Equation 4.13 as follows:

$$\lambda \mathbb{E}_{p(x)} \sqrt{\sum_{k=1}^{K} (-\log F_{\theta}^{k}(x)[N+1])^2}. \tag{4.14}$$

We power the loss associated with the damping neuron of each classifier, sum these squared values, and then extract the square root of the aggregate to form the

final loss component.

The power operation intensifies the gradient, focusing it more on the larger values. Since our damping loss must balance the damping component with the cross-entropy component, we apply a square root to the combined damping components across classifiers after performing the power operation. This ensures that the overall damping component remains balanced with the cross-entropy component.

Unlike the commonly used L2 (MSE) loss, which sums the squared errors of each neuron, our method aggregates the damping components and then applies a square root. In L2, the squaring operation amplifies large residuals, so the gradients from the damping neurons can become excessively large and dominate the optimization, breaking the balance with the cross-entropy loss. Our square root formulation limits this effect by adjusting the overall damping magnitude, allowing the damping term to remain balanced with the cross-entropy term during training.

As presented in Proposition 1, the gradient of our damping component generates a stronger inhibitory effect when the softmax value of the correct label is large. After applying the power-sqrt modification, this inhibitory gradient becomes:

$$\frac{-\log F_\theta^k(x)[N+1] \cdot F_\theta^k(x)[y]}{\sqrt{\sum_{k=1}^{K}\left(-\log F_\theta^k(x)[N+1]-1\right)^2}} \tag{4.15}$$

The damping component gradient for the neuron corresponding to the correct label from the power-sqrt loss is:

$$\frac{\partial\sqrt{\sum_{k=1}^{K}\left(-\log F_\theta^k(x)[N+1]\right)^2}}{f_\theta(x)[y]} \tag{4.16}$$

Similarly, for the k-th classifier, the gradient of the damping component with respect to the neuron corresponding to the correct label is:

$$F_\theta^k(x)[y] \tag{4.17}$$

According to the chain rule, we obtain the final gradient:

$$\frac{-\log F_\theta^k(x)[N+1] \cdot F_\theta^k(x)[y]}{\sqrt{\sum_{k=1}^{K}\left(-\log F_\theta^k(x)[N+1]-1\right)^2}} \tag{4.18}$$

Compared with damping loss, our power-sqrt loss introduces a weight $\frac{-\log F_\theta^k(x)[N+1]}{\sqrt{\sum_{k=1}^{K}\left(-\log F_\theta^k(x)[N+1]-1\right)^2}}$ to modify the gradient. For different classifiers, the denominator of the weight remains the same, while the numerator assigns larger gradients to neurons with smaller damping neuron softmax values.

Since the softmax value of the damping neuron is computed alongside all other neurons that correspond to class labels, a larger softmax value for the correct label is often accompanied by a smaller softmax value for the damping neuron. As a result, our power-sqrt loss focuses the damping gradients more on well-performing classifiers, allowing more parameter space to be allocated to underperforming classifiers.

4.2.5 Sample wise dynamic training

The current approach to training early-exiting networks is linear scalarization, where weights are applied to the gradients of different classifiers during training to manage the tradeoffs between them. Our method, however, focuses on evaluating whether these gradients are necessary. After identifying the essential gradients using our approach, linear scaling can still be applied to manage the tradeoff among these necessary gradients.

Unlike the current meta-learning methods (Han et al., 2022; Sun et al., 2022), which train a meta net during the training process to determine the weights for different classifiers, we propose a further simplified approach. Our method avoids the additional training overhead and hyperparameters associated with a meta net by directly using the values from our damping neurons to determine the weights.

Following (Han et al., 2022), we normalize the damping-neuron outputs to weights $\tilde{w} \in [-\alpha, \alpha]$, with $\alpha = 0.8$ on CIFAR and $\alpha = 0.3$ on ImageNet. The

final weights are then defined as

$$w = \tilde{w} + 1. \tag{4.19}$$

Specifically, for a sample x_i, we compute

$$\tilde{w}_i = -\log F_\theta(x_i)[N+1]. \tag{4.20}$$

The classifier-training loss is then formulated as

$$\mathcal{L} = \sum_{k=1}^{K} \frac{1}{S} \sum_{i=1}^{S} w_i^{(k)} \mathcal{L}_i^{(k)}, \tag{4.21}$$

where K is the number of classifiers, S is the number of samples, and $\mathcal{L}_i^{(k)}$ denotes our power-sqrt loss.

We treat the damping neuron values as constant weights when calculating the gradient of the loss function. This is because incorporating gradients of these weights into the loss gradient would alter the relative importance of different classifiers during training. Specifically, deeper classifiers, which tend to achieve better results, would experience reduced loss. Consequently, if these weights were also differentiated during the gradient computation, instead of being treated as constants, it would encourage increasing the weights applied to deeper networks, thus further reducing the overall loss. However, this would encourage a decrease in the damping neuron values of deeper classifiers, conflicting with the design of our damping loss and degrading performance. Further detailed analyses are provided in the ablation study section.

4.3 Experiments

In this section, we evaluate our method through extensive experiments conducted on the CIFAR (Krizhevsky et al., 2009) and ImageNet (Deng et al., 2009) datasets. Our training strategy is implemented on MSDNet (Huang et al., 2017b) and RANet (Yang et al., 2020b), which are representative early-exiting architectures commonly

used as backbones to evaluate the performance of related methods (Han et al., 2022; Meronen et al., 2024; Gong et al., 2024).

We compare our method with the meta-learning training approach WPN (Han et al., 2022) and the feature partitioning method DFS (Gong et al., 2024). Furthermore, our method can be integrated with linear scalarization techniques to further enhance performance.

Datasets. CIFAR-10 and CIFAR-100 (Krizhevsky et al., 2009) both contain 50,000 training images and 10,000 test images. The size of the image is 32×32. CIFAR-10 has 10 classes, and CIFAR-100 has 100 classes for the classification task. ImageNet (Deng et al., 2009) has 1.2 million 224×224 images for training, 50,000 images for validation and 1000 classes for the classification task. For the sake of fair comparison, we followed (Han et al., 2022) setting data augmentations which contain data normalization, random crop, and random flip.

Backbone architecture and implementation. Our method can be easily applied to any early exit network. We conduct experiments on two representative early exit architectures, MSDNet (Huang et al., 2017b) and RANet (Yang et al., 2020b).

We follow (Han et al., 2022) in selecting MSDNet and RANet as backbone architectures. For the CIFAR-100 and CIFAR-10 datasets, we train for 300 epochs with a batch size of 64, using SGD optimizer with a momentum of 0.9 and an initial learning rate of 0.1 decaying with a cosine shape. For the ImageNet dataset, we train for 100 epochs with a batch size of 256, using the same SGD optimizer configuration.

4.3.1 Performance Evaluation

Results on CIFAR dataset. We first implement our training strategy on MSDNet with seven exits, for both the CIFAR-10 and CIFAR-100 datasets. We set the hyperparameter λ to 0.005 for CIFAR-100 and to 0.075 for CIFAR-10. Initially, we present the 'Anytime Prediction' setting on a 7-exit MSDNet, which details the accuracy of each classifier alongside the corresponding FLOPs (floating point

Table 4.1: **Anytime prediction results** of a 7-exit MSDNet on CIFAR100. (Bold means the best result.)

Exit	1	2	3	4	5	6	7
Params ($\times 10^6$)	0.30	0.65	1.11	1.73	2.38	3.05	4.00
FLOPs ($\times 10^6$)	6.86	14.35	27.29	48.45	76.43	108.90	137.30
MSDNet (Huang et al., 2017b)	60.78	64.54	68.51	71.41	73.68	75.61	76.31
WPN (Han et al., 2022)	62.47	**66.32**	68.10	71.29	73.21	74.87	75.81
Damping	61.53	64.71	68.22	71.27	73.76	75.27	75.76
Power-sqrt	62.07	65.44	69.32	71.61	73.88	75.89	76.45
+ Dynamic	**62.74**	65.69	**69.76**	**71.77**	**74.61**	**75.96**	**76.63**

operations, a common metric for assessing the computational budget of the model) as shown in Table 4.1.

Compared to the MSDNet baseline, our method achieves notable improvements across nearly all classifiers. Additionally, it outperforms the current state-of-the-art meta-learning approach, demonstrating its effectiveness in early-exiting networks.

We show that using the same hyperparameters for all classifiers with our damping loss method led to performance improvements in some classifiers while causing declines in others. However, when incorporating our power-sqrt gradient adjustment, the results showed significant overall improvement. Furthermore, our method is also compatible with existing linear scalarization approaches. When combined with our proposed simplified Dynamic training, it achieves further performance gains.

We also present results on the CIFAR10 dataset in Table 4.2, where our method continues to achieve stable improvements. Notably, the enhancements on CIFAR10 are less pronounced compared to those on CIFAR100. This discrepancy arises because we employ the same MSDNet model architecture for both CIFAR100 and CIFAR10, and the parameter space provided by the model is more than sufficient for CIFAR10, thus reducing the impact of overfitting. This comparison further

illustrates the effectiveness of our method in unlocking the potential of the parameter space.

Table 4.2: **Anytime prediction results** of a 7-exit MSDNet on CIFAR10

Exit index	1	2	3	4	5	6	7
Params($\times 10^6$)	0.30	0.65	1.11	1.73	2.38	3.05	4.00
FLOPs($\times 10^6$)	6.86	14.35	27.29	48.45	76.43	108.90	137.30
MSDNet	88.51	90.38	92.15	93.21	93.89	94.22	94.54
WPN	88.54	90.19	91.61	92.55	93.28	93.40	93.67
Power-sqrt	88.38	90.33	92.06	93.71	94.23	94.45	94.49
+ Dynamic training	**88.42**	**90.21**	**92.01**	**93.58**	**94.27**	**94.50**	**94.55**

4.3.2 Results on RANet.

We extended our experiments to include RANet, another representative early exiting architecture, to demonstrate the generality of our method. The anytime prediction results for the six exit RANet are displayed in Table 4.3. The improvements observed with our method on RANet are more significant than those on MSDNet.

This difference can be attributed to RANet's hierarchical processing of data resolutions, where shallow classifiers operate on low-resolution data, while deeper classifiers are exclusively fed high-resolution features. This design increases the disparity between features processed at different classifier depths, making the negative impact of classifier overfitting on others more severe. Consequently, the reduction of unnecessary gradients and the optimized parameter space utilization provided by our method become even more crucial in mitigating this effect.

Results on ImageNet. To further demonstrate the effectiveness of our method, we conducted experiments on the large-scale ImageNet dataset. We use MSDNet with five exits as the backbone architecture. For the ImageNet dataset, we set the

Table 4.3: **Anytime prediction results** of a 6-exit RANet on CIFAR100

Exit index	1	2	3	4	5	6
Params($\times 10^6$)	0.36	0.90	1.30	1.80	2.19	2.62
FLOPs($\times 10^6$)	8.37	21.79	32.88	41.57	53.28	58.99
RANet	65.28	68.16	70.52	70.64	72.39	72.75
WPN	65.33	68.69	70.36	70.80	72.57	72.45
Power-sqrt	65.63	68.96	71.49	71.65	73.19	73.69
+ Dynamic training	**65.67**	**69.38**	**71.88**	**71.92**	**74.19**	**74.26**

Table 4.4: **Anytime prediction results** of a 5-exit MSDNet on ImageNet.

Exit	1	2	3	4	5	Δ
Params ($\times 10^6$)	4.24	8.77	13.07	16.75	23.96	/
FLOPs ($\times 10^9$)	0.34	0.69	1.01	1.25	1.36	/
MSDNet	59.03	66.49	70.56	72.39	74.20	–
WPN	59.54	67.22	71.03	72.33	73.93	$\uparrow$ **1.39**
DFS	61.80	68.03	70.75	71.79	72.88	$\uparrow$ **2.58**
Power-sqrt	59.43	67.12	71.21	72.91	74.45	$\uparrow$ **2.46**
+ Dynamic	59.58	67.46	71.33	73.19	74.74	$\uparrow$ **3.63**

hyperparameter λ to 0.01. Results presented in Table 5.3 show that our method continues to achieve significant improvements on the large-scale ImageNet dataset.

Dynamic inference results. In the Dynamic Inference experimental setting, we evaluate a 5-exit MSDNet on the ImageNet dataset, where early-exiting networks dynamically select classifiers based on the computation budget to process incoming data. The anytime prediction results are presented in Table 5.3. As shown in Fig. 4.4, deeper classifiers have a greater impact on overall performance in this setting. For instance, the meta-learning approach significantly outperforms

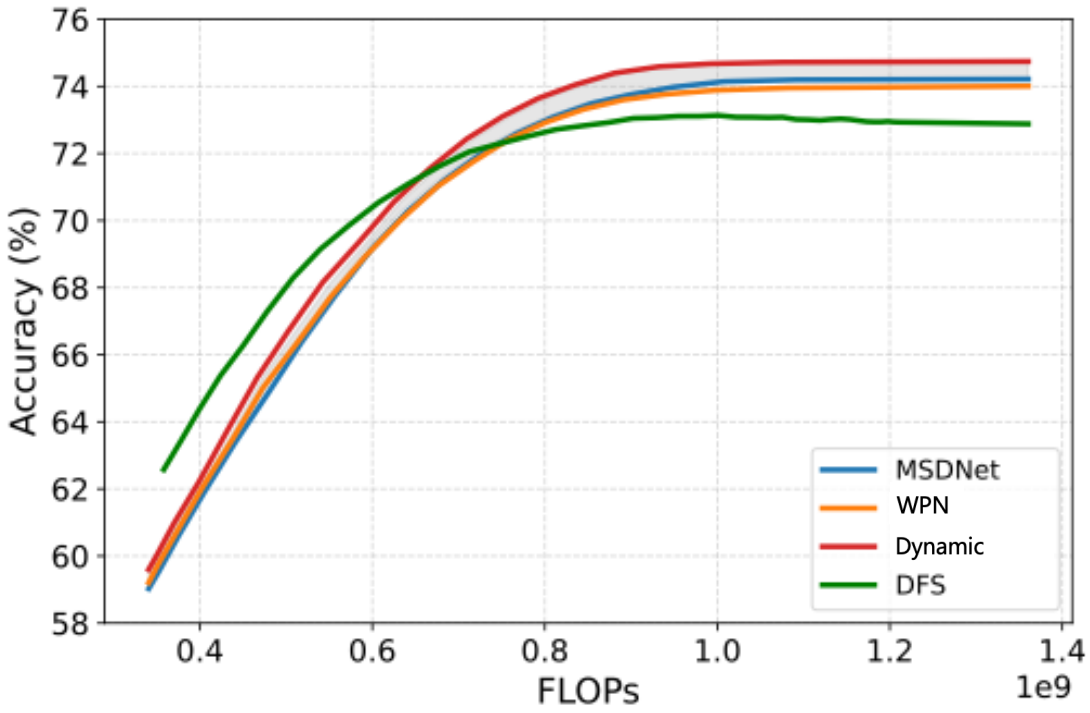


Figure 4.4: Dynamic inference results on ImageNet.

the MSDNet baseline in the first three classifiers. However, MSDNet achieves better performance in its deepest classifier. As a result, the performance gap between MSDNet and the meta-learning approach is relatively small under Dynamic Inference.

This is because, in the Dynamic Inference setting, when shallow classifiers misclassify samples, the model has the flexibility to defer the decision to deeper classifiers. Since the performance of the deepest classifier often represents the upper bound of the model's capacity, this setting inherently mitigates the limitations of weaker classifiers. Our method, by filtering out unnecessary gradients, provides the model with greater learning capacity, leading to improved performance in deeper classifiers. Consequently, it achieves competitive results in the Dynamic Inference setting.

Comparison with label smoothing While our method may appear superficially similar to confidence-based regularization techniques such as label smoothing,

its design and purpose are fundamentally different. Label smoothing applies a uniform confidence penalty to each classifier independently. By contrast, our method is specifically tailored for early-exiting architectures, where all classifiers must be trained jointly.

To achieve this, we introduce a *damping neuron* whose output acts as a learned coordination signal. This signal drives two key components of our approach:

- **Dynamic training**, where it adaptively controls the weight assigned to each classifier's gradient;
- **Power-sqrt loss**, where it regulates the degree of gradient damping across classifiers.

The critical distinction is that our method explicitly coordinates the optimization of all classifiers through a shared and interpretable mechanism, rather than regularizing them in isolation.

To further highlight this difference, we compared our approach with label smoothing in early-exiting experiments (Table 4.4). These experiments are conducted on well-established image classification benchmarks (e.g., MSDNet backbones), which already incorporate mature overfitting-prevention techniques. In this setting, label smoothing fails to provide additional benefits. In contrast, our method directly addresses the unique challenge of early-exiting training—*jointly optimizing multiple classifiers under potential gradient conflicts*—and thus yields consistent improvements.

4.3.3 Ablation Study

We conduct ablation studies on the 7-exit MSDNet which is used in our CIFAR experiments. These studies assess the effectiveness of our damping criterion, examine the sensitivity of our hyperparameter λ, and evaluate the impact of our damping neuron values.

Table 4.5: **Comparison with label smoothing** on 7-exits MSDNet on CIFAR100.

Exit	1	2	3	4	5	6	7
MSDNet	60.78	64.54	68.51	71.41	73.68	75.61	76.31
Label smoothing	61.33	64.80	68.32	70.88	73.10	74.75	75.68
Power-sqrt	**62.07**	**65.44**	**69.32**	**71.61**	**73.88**	**75.89**	**76.45**

Table 4.6: Ablation results on the 7-exit MSDNet (CIFAR100).

	Accuracy
MSDNet last only	75.98
Threshold last only	76.41
Damping loss last only	76.63

4.3.3.1 Adaptive Damping Criterion.

We modify MSDNet by removing all intermediate classifiers, keeping only the final one to evaluate the effectiveness of our damping loss in a standard setting. In this setup, our method dynamically applies damping to a single classifier based on different training data, without managing multiple classifiers.

The results in Table 4.6 demonstrate the effectiveness of our approach. **MSDNet last only** represents the performance with only the last classifier retained in MSDNet, while **Damping loss last only** applies our damping loss in this setting. We also compare our method with **Threshold last only**, which incorporates a threshold-based gradient selection mechanism into MSDNet with a single classifier.

The results of this experiment confirm that our damping gradients do not degrade classifier performance. Furthermore, as the training states of different data samples vary, our dynamic damping strategy effectively adapts to these variations. Notably, the **MSDNet last only** results presented here are based on the best-performing epoch, following the conventional early stopping method. This highlights

Table 4.7: **Ablation study of hyperparameter λ** *(values are mean ± std)*

Method	C1	C2	C3	C4	C5	C6	C7	Avg
damping 0.005	61.650 ± 0.495	64.908 ± 0.330	68.022 ± 0.480	71.160 ± 0.510	73.592 ± 0.480	75.164 ± 0.630	75.840 ± 0.395	70.048 ± 0.196
damping 0.025	61.360 ± 0.440	64.802 ± 0.395	67.792 ± 0.555	70.782 ± 0.255	73.746 ± 0.710	75.640 ± 0.800	76.238 ± 0.630	70.051 ± 0.212
damping 0.05	61.096 ± 0.545	64.136 ± 0.825	67.872 ± 0.610	70.652 ± 0.325	73.492 ± 0.385	75.510 ± 0.350	75.750 ± 0.470	69.787 ± 0.248
damping 0.075	61.408 ± 0.735	64.378 ± 0.315	68.036 ± 0.745	70.756 ± 0.235	73.566 ± 0.145	75.344 ± 0.190	75.818 ± 0.360	69.901 ± 0.129
power-sqrt 0.005	62.094 ± 0.295	65.226 ± 0.420	68.448 ± 0.405	71.654 ± 0.460	73.828 ± 0.295	75.494 ± 0.375	76.012 ± 0.470	70.394 ± 0.215
power-sqrt 0.025	61.884 ± 0.480	65.432 ± 0.525	68.578 ± 0.425	71.788 ± 0.670	74.152 ± 0.160	75.638 ± 0.135	76.116 ± 0.400	70.513 ± 0.139
power-sqrt 0.05	62.010 ± 0.405	65.046 ± 0.305	68.594 ± 0.470	71.560 ± 0.565	74.028 ± 0.320	75.860 ± 0.395	76.238 ± 0.535	70.477 ± 0.086
power-sqrt 0.075	61.806 ± 0.360	65.028 ± 0.450	68.520 ± 0.940	71.708 ± 0.335	74.192 ± 0.535	75.996 ± 0.195	76.220 ± 0.255	70.496 ± 0.221

that our dynamic damping approach outperforms both Threshold last only and early stopping, demonstrating its superior adaptability in single-classifier training. Moreover, the power-sqrt loss builds on the original damping loss by further utilizing the damping neuron to encode the relative convergence of each classifier, thereby enabling more fine-grained gradient control.

4.3.3.2 Sensitivity analysis.

We present the sensitivity analysis of the hyperparameter λ in Table 4.7. For each parameter setting, we conduct five independent runs and report the mean and variance of the results. The analysis shows that while our damping loss is relatively sensitive to the choice of λ, the power-sqrt loss significantly reduces this sensitivity. With the power-sqrt loss, small λ values achieve similarly effective results.

4.3.4 Damping neuron.

In this section, we conduct a detailed analysis of the values generated by the damping neuron and their effects, as well as their gradients in dynamic training. We keep the hyperparameter λ to 0.005 for damping loss, power-sqrt, and dynamic training.

Table 4.8 provides a detailed presentation of the average values of the damping neuron for each classifier across different methods on the test set. In Table 4.8, the numbers on the left represent accuracy, while the bolded values in parentheses indicate the average values of the damping neuron. We observe that the values of the damping neuron are consistently higher in deeper classifiers compared to shallower

Table 4.8: **Ablation study of the damping neuron.** Bold numbers in parentheses are the average damping-neuron values.

Exit index	1	2	3	4	5	6	7
Params ($\times 10^6$)	0.30	0.65	1.11	1.73	2.38	3.05	4.00
FLOPs ($\times 10^6$)	6.86	14.35	27.29	48.45	76.43	108.90	137.30
Damping loss	61.53(**0.006**)	64.71(**0.008**)	68.22(**0.015**)	71.27(**0.028**)	73.76(**0.050**)	75.27(**0.070**)	75.76(**0.071**)
+ Power-sqrt	62.07(**0.002**)	65.44(**0.003**)	69.32(**0.006**)	71.61(**0.012**)	73.88(**0.024**)	75.89(**0.036**)	76.45(**0.037**)
+ Dynamic training	62.74(**0.004**)	65.69(**0.006**)	69.76(**0.010**)	71.77(**0.019**)	74.61(**0.031**)	75.96(**0.049**)	76.63(**0.044**)
Dynamic training with gradient	62.66(**0.041**)	66.19(**0.029**)	70.19(**0.020**)	72.13(**0.010**)	74.28(**0.001**)	75.64(**0.004**)	76.07(**0.006**)

ones. This suggests that deeper classifiers are more likely to achieve superior training performance, thereby more frequently activating the damping mechanism. We observe that while deeper classifiers tend to have higher damping neuron values, these values are smaller in the power-sqrt version of the damping neuron. This occurs because the power-sqrt approach takes into account the training conditions of different classifiers collectively.

We delve deeper into the analysis of weight gradients in dynamic training. As seen from Table 4.8, when these weights possess gradients during training, they tend to assign larger weights to deeper classifiers since they exhibit lower losses, thereby aiming for an overall reduction in total loss. However, this conflicts with the design of our damping loss. It is observed that, when retaining the gradients of weights, deeper networks paradoxically exhibit smaller damping neuron values, contrary to the previously observed pattern. This conflict can lead to a decline in training performance. When we use the values of the damping neurons as weights without propagating gradients, the outcomes are generally consistent with our earlier observations of our power-sqrt version and yield better performance.

4.4 Conclusion

In this chapter, we present an adaptive damping mechanism specifically for training early exiting networks. Each classifier's fully connected layer is augmented with

a damping neuron, receiving a small gradient to enable adaptive damping when sufficiently trained. Our power-sqrt loss further incorporates a joint consideration of the damping mechanisms across different classifiers. This adaptive damping mechanism significantly enhances the training effectiveness of early exiting networks. By freeing up parameter space that would typically be wasted on overfitting in traditional training methods, the performance of early-existing networks significantly outperforms the current methods. Furthermore, our approach is compatible with state-of-the-art linear scalarization training methodologies.

4.5 Computation resources

We run experiments on an Nvidia RTX4090 GPU, 12 cores Xeon(R) Platinum 8352V and 90GB RAM. For CIFAR100 experiments, our methods (both damping loss and power-sqrt loss) need approximately 14 hours for total 300 epochs. For ImageNet, our methods need approximately 40 hours for total 100 epochs.

Chapter 5

Decision-centered teamwork in Dynamic Early Exiting Networks

Dynamic early exiting is an effective method for improving the efficiency of deep neural networks. By adding classifiers into intermediate layers of deep neural networks, early exiting allows inference to conclude early for simpler samples. However, usually early exiting networks use the output of the final processed classifier, ignoring the information given by the previous classifiers.

In this chapter, we explore teamwork in dynamic networks. Prior work in multi-agent systems shows that effective teamwork requires flexible coordination under uncertainty, since agents may have incomplete or inconsistent views of the environment (Tambe, 1997). We treat the multi-agent collaboration in early-exiting networks as decision-centred teamwork (Marcolino, 2016), where multiple classifiers collaborate to make decisions.

We propose a Bayesian updating method to effectively integrate information from earlier classifiers and introduce a conformal prediction-based approach to improve likelihood estimation. Additionally, we propose a logit voting method that, despite using less information, performs well in practical deep learning applications. It provides a practical alternative when accurate likelihoods for Bayesian updating are difficult to obtain. Through synthetic and real-world experiments, we demonstrate

the effectiveness of these methods and the significant potential of the Bayesian updating approach.

5.1 Introduction

Although deep neural networks have achieved remarkable success across a wide range of tasks (Krizhevsky et al., 2012; He et al., 2016; Huang et al., 2019; Dosovitskiy, 2020), their computational demands present a significant challenge for deployment on resource-constrained devices. Various approaches have been explored to improve the inference efficiency of deep neural networks, such as network pruning (LeCun et al., 1989; He et al., 2019; Yang et al., 2021), weight quantization (Hubara et al., 2016; Han et al., 2015; Jung et al., 2019), and the development of lightweight architectures (Howard et al., 2017; Zhang et al., 2018; Sandler et al., 2018). Although these methods are effective, handling challenging data samples still requires large-scale deep neural networks. Consequently, dynamic neural networks (Han et al., 2021; Bolukbasi et al., 2017) have gained attention for the ability to perform data-dependent inference by dynamically adapting the models.

Dynamic early exiting, a representative approach in dynamic neural networks, adds multiple classifiers to intermediate layers. It enables early termination of inference when predefined criteria, such as softmax confidence scores (Huang et al., 2017b) or a learned policy (Chen et al., 2020), are met. Current mainstream early exiting methods typically use the output of the deepest classifier at the exit point as the final result, ignoring the potential value of intermediate classifier outputs. However, these outputs, as suggested by Bayesian theory, can provide valuable contextual information to enhance model performance.

In the field of early exiting, collaboration among multiple classifiers has primarily focused on the training phase. For example, Han *et al.*(Han et al., 2022) and Sun *et al.*(Sun et al., 2022) proposed a meta-learning approach to address gradient conflicts between classifiers during training, while Wolczyk *et al.* (Wołczyk et al., 2021)

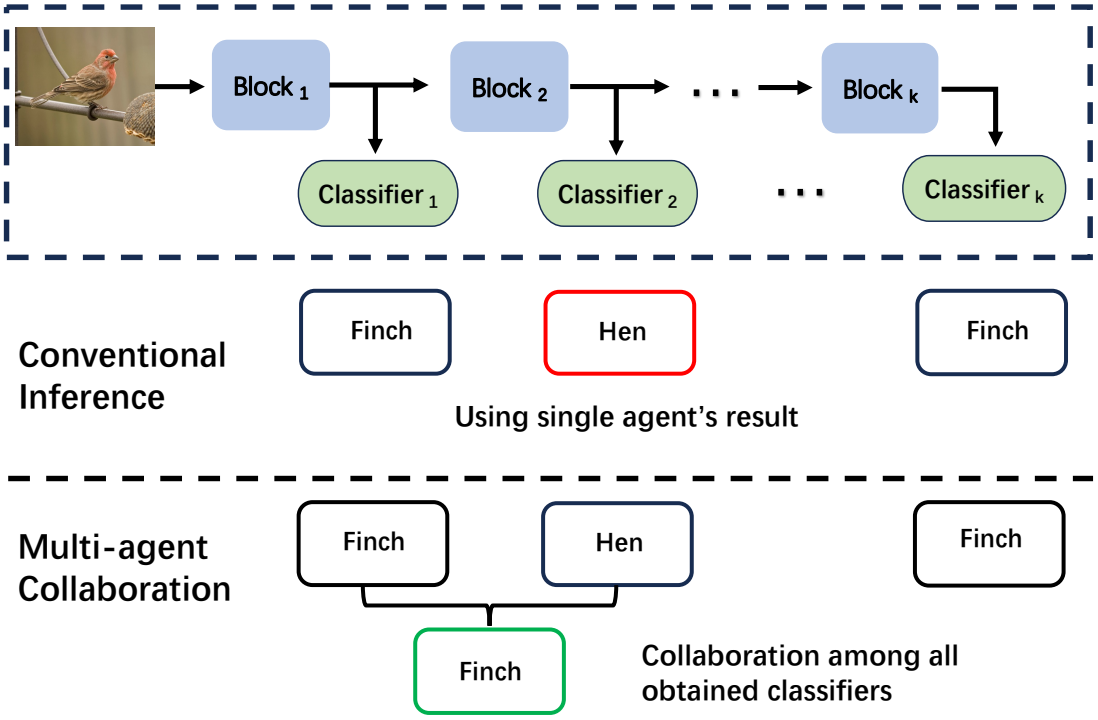


Figure 5.1: **Multi-agent collaboration-based inference strategy.** During inference, instead of relying solely on the deepest classifier, we utilize the collaborative output from all classifiers that have produced results so far.

introduced skip connections and knowledge distillation techniques to improve overall training performance. In the inference phase, Patience-based Early Exit (PBEE) (Zhou et al., 2020) proposed a novel exit mechanism where the model exits when consecutive classifiers produce the same output. Although this approach improves accuracy, it often fails to achieve a satisfactory trade-off between efficiency and accuracy.

In this chapter, we explore multi-agent collaboration within dynamic early exiting networks. In particular, in decision-centered teamwork, agents solve challenges by taking joint decisions as a team (Marcolino et al., 2013; Marcolino, 2016; Marcolino et al., 2014). However, in deep learning, the application of ensemble methods faces one issue: training multiple agents (classifiers) often requires significant computational resources. Early exiting architectures, which have become

increasingly important in the era of large models, inherently contain multiple classifiers, effectively avoiding the additional computational costs associated with ensemble methods. This makes early exiting a promising framework for exploring decision-centred teamwork in large-scale deep learning area.

Despite this potential, applying multi-agent collaboration in deep learning introduces new challenges. Large-scale deep learning tasks are often complex, such as classification problems with hundreds or even thousands of categories. To explore multi-agent collaboration in dynamic early exiting networks, we propose a Bayesian update-based collaboration method. Specifically, we obtain the likelihood of each classifier's output for different labels by counting and analyzing results from a validation set, which are then used for Bayesian updating. However, through synthetic experiments, we find that accurately estimating likelihoods when the validation set is not large is challenging, significantly limiting the performance of Bayesian updating in some scenarios.

To address this limitation, we propose a conformal prediction-based approach to improve the accuracy of likelihood estimation. I.e., we leverage the effectiveness of conformal prediction in large-scale deep learning tasks and use its more accurate confidence levels to refine the confidence estimates within the likelihood calculation.

Additionally, we propose a logit voting method as an alternative. While this approach utilizes less information and may have lower theoretical potential compared to Bayesian updating, it often performs well in practical deep learning scenarios, making it a viable fallback when accurate likelihoods for Bayesian updating are difficult to obtain.

Hence, in this work, we make the following contributions:

1. **Decision-centered teamwork in dynamic neural networks**: We explore decision-centered teamwork in large-scale dynamic networks, a domain with significant challenges and opportunities.

2. **Bayesian updating method:** We propose a Bayesian updating-based collaboration method to enhance the interactions between agents in dynamic

networks.

3. **Conformal prediction for likelihood improvement:** To address the inaccuracy of likelihood estimation in Bayesian updating, we introduce a conformal prediction-based approach that significantly improves the reliability of the method.

4. **Extensive experimental validation:** We conduct extensive experiments, including synthetic analyses and real-world computer vision applications, to demonstrate the effectiveness and strong potential of the Bayesian updating method.

5. **Voting-based alternative:** We study a voting-based method as a simpler alternative. Despite its simplicity, it performs well in practice and serves as a viable option when Bayesian updating is constrained by the size of the validation set.

Therefore, our work introduces a new challenge for the multi-agent systems community, with significant potential for broader applications in large-scale deep learning.

5.2 Background

We start by introducing dynamic early exiting networks, and presenting the motivation for our work.

5.2.1 Dynamic Early Exiting Networks

Unlike static neural networks, dynamic early exiting networks add classifiers to intermediate layers of the original architecture. For each classifier C_c:

$$p_c = C_c\left(\mathbf{x}, \theta_c\right), \tag{5.1}$$

where $\mathbf{x}$ represents the input data sample, θ_c indicates the parameters of the c-th backbone and classifier. The prediction of the c-th classifier for the data sample $\mathbf{x}$ is denoted as p_c, representing the probability distribution over all classes. Each classifier consists of a fully connected layer followed by a softmax operation, which generates this probability distribution.

During the inference phase, each classifier produces an output along with a confidence score, typically represented by the maximum softmax value. If this confidence score exceeds a predefined threshold, the model exits early; otherwise, it continues to process deeper layers.

5.2.2 Multi-agent collaboration

In dynamic multi-exit networks, classifiers are placed at different depths to handle inputs of varying difficulty—easy samples exit early, while harder ones proceed to deeper layers. Although these classifiers share a common backbone, each one makes decisions at different stages and with different levels of abstraction. This motivates a multi-agent perspective, where each classifier acts as an agent making independent decisions based on its local view.

However, unlike traditional ensembles that rely on fully independent models, our work investigates aggregation in a shared-backbone setting, which is specific to dynamic neural networks. That being said, parameter sharing does not negate the need for coordination. Prior work on decision-centered teamwork (Marcolino et al., 2013) shows that combining individual decisions can outperform isolated reasoning. Building on this insight, we introduce a framework that integrates multi-agent collaboration into dynamic networks, bridging two active research areas in a unified approach.

5.2.3 Motivations

In dynamic early exiting networks, the mainstream approach during inference is still to select the output of a single classifier p_c as the final result. When obtaining the

output from the c-th classifier, the results from the 1st to the $(c-1)$th classifiers have already been generated but are discarded. Generally, deeper networks tend to offer better accuracy, however, studies have shown that dynamic early exiting networks may suffer from overthinking, where a shallow classifier produces the correct result while deeper classifiers make errors (Kaya et al., 2019). In the field of multi-agent collaboration, there are works suggesting that a team of diverse (and weak) agents can, through collaboration, outperform a single, stronger agent (Marcolino et al., 2013).

Bayesian updating is a promising approach for refining the probabilities of different options based on a series of observations. In image recognition tasks, analyzing the confusion matrix reveals that classifier errors often concentrate on certain classes. For instance, a model may frequently confuse cats with tigers but rarely with airplanes. In such cases, leveraging predictions from earlier shallow classifiers when the model produces easily confusable outputs can lead to more accurate results.

5.3 Methodology

5.3.1 Bayesian updating based multi-classifiers collaboration

We propose a Bayesian updating method to facilitate collaboration between classifiers in dynamic early exiting networks. This method integrates outputs from all previous classifiers to calculate the probability of each class.

The Bayesian equation is reformulated as follows to compute the probability of label l_i given the output θ_c of classifier c:

$$P(l_i|\theta_c) = \frac{P(\theta_c|l_i) * P(l_i|\theta_{c-1})}{P(\theta_c)} \tag{5.2}$$

$P(l_i|\theta_c)$ represents the posterior distribution obtained in the current round of

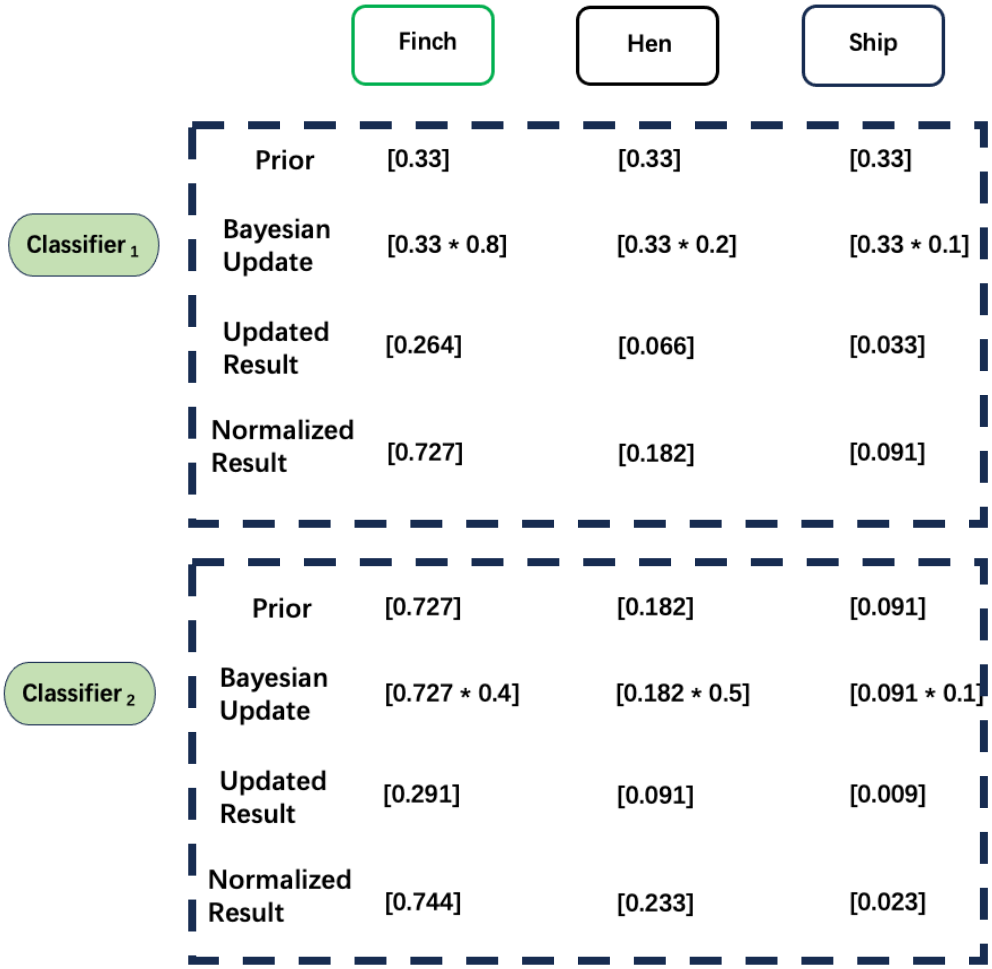


Figure 5.2: **An illustration of Bayesian Updating.**

Bayesian updating, while $P(l_i|\theta_{c-1})$ is the posterior distribution from the previous round, which serves as the prior distribution for the current update. In the first round of Bayesian updating, the prior distribution is initialized as a uniform distribution across all labels. $P(\theta_c|l_i)$ represents the likelihood of the current classifier, which can be estimated based on the model's performance on the validation set. For the class probabilities $P(l_i|\theta_c)$ that we ultimately need, $P(\theta_c)$ is identical across all classes l_i, and thus can be normalized out. This process is applied recursively across classifiers in the dynamic neural network, updating the posterior

distribution.

We use a simple 3-class classification example to illustrate our Bayesian updating method, as shown in Figure 5.2. The prior distribution is initialized as uniform at the first classifier. The likelihood is derived from the output of the current classifier. The posterior distribution is then calculated by multiplying the likelihood with the prior and normalizing it to obtain a valid probability distribution. In subsequent rounds, the posterior from the previous round is used as the prior, and the process is repeated.

The likelihoods in our Bayesian updating method incorporate confidence levels and common misclassification patterns. For example, in Figure 5.2, Classifier 2 makes a common error, confusing 'hen' with 'finch', while being unlikely to confuse them with 'ship'. The likelihood from Classifier 2 assigns higher probability to 'hen' while also reflecting this common confusion. By leveraging prior information from earlier classifiers, which were more confident in 'finch', the error can be corrected.

5.3.2 Likelihood estimation

Here, we start with the simplest representation, where θ_c corresponds to the label with the highest confidence (e.g., the argmax of a softmax output). Later, we will introduce a more advanced version that incorporates softmax information for improved performance.

The likelihood $P(\theta_c \mid l_i)$ is estimated by deriving it from the confusion matrix of the corresponding classifier on the validation set, which shows how often each output is produced for each ground truth label. Specifically, for each classifier c, the confusion matrix records the frequency with which it outputs a specific label when the true label is l^*. Aggregating these frequencies across the validation set provides an approximation of $P(\theta_c \mid l_i)$ for all possible outputs and labels l_i.

Specifically, we use the validation set to construct a confusion matrix that records the count of each combination of output θ_c and its corresponding ground truth label l_i. This provides an estimate of $P(\theta_c \mid l_i)$, the probability of producing θ_c given that

the ground truth is l_i. In our Bayesian updating process, θ_c represents the observed output of the current classifier. Using the information from the confusion matrix, we estimate $P(\theta_c \mid l_i)$ for different ground truth labels l_i.

5.3.3 Challenges of Bayesian Updating in Large-Scale Deep Learning

In the era of large models, deep learning is often used to tackle complex tasks and large datasets. This complexity makes it challenging to accurately estimate the likelihood in Bayesian updating. For instance, in classification tasks with hundreds or even thousands of classes, computing the likelihood from a validation set requires a significant amount of data to maintain accuracy. Additionally, discrepancies between the distribution of the validation set and the test set in real-world applications further impact the reliability of the likelihood estimates. For detailed analysis, please refer to our synthetic experiments.

We also explored the potential of Bayesian updating under ideal conditions by calculating likelihoods using the test set to construct the confusion matrix. This serves only as the upper bound performance under oracle likelihood conditions. Notably, this setup does not involve using the test set for training and therefore does not affect the model's performance. It only influences the confidence levels of the predictions. To ensure fairness, we retained the initial values for each class to prevent the model from exploiting exclusions to achieve better results. Our experiments demonstrate that Bayesian updating has significant potential, with the accuracy of likelihood estimation being the primary limiting factor. Confidence estimation in large-scale deep learning remains a challenging and open problem.

5.3.4 Enhanced Likelihood Estimation Through Conformal Prediction

Estimating likelihoods in Bayesian updating is difficult, especially in large-scale deep learning, which requires better methods for confidence estimation. Conformal prediction is a promising approach for confidence estimation, showing strong performance in large-scale deep learning scenarios (Angelopoulos et al., 2021). Moreover, it incorporates information from softmax outputs to provide a more precise definition of outputs in our Bayesian method, enhancing the accuracy of likelihood estimation.

However, conformal prediction does not directly provide a confidence score. By analyzing the softmax scores on a validation set, it generates conformal sets that capture the uncertainty of predictions (Angelopoulos et al., n.d.). Specifically, the method accumulates softmax probabilities until a predefined threshold is reached, at which point the corresponding labels are included in the conformal set. These sets ensure that the true label is contained with a predefined confidence level, making the predictions more reliable and interpretable.

We observed that conformal prediction maintains high confidence accuracy even for single outputs (i.e., when the conformal set is limited to one). Leveraging this property, we further refined our likelihood estimation method. Specifically, we categorized outputs into three groups based on predefined confidence levels (e.g., above 90%, 90%-80%, and below 80%) using their associated softmax values. That is, for each label l_i, a classifier may output $\theta_c = \{l_i, > 90\%\}$, $\theta_c = \{l_i, 80\% - 90\%\}$, or $\theta_c = \{l_i, < 80\%\}$. We then adjusted our likelihood computation by performing counts separately within each group.

By incorporating conformal prediction, we include softmax information in the output and achieve more accurate confidence estimates.

5.3.5 Logit Voting

List et al. (List et al., 2001) extended the Condorcet Jury Theorem by showing that if each agent is more likely to give the correct answer than any particular incorrect one, then as the number of agents approaches infinity, the probability that majority voting yields the correct outcome converges to 1. This condition on individual accuracy is easily satisfied in large-scale deep learning settings. As a result, collaborative voting can still provide benefits, even when all participating agents are relatively weak.

Diversity has been widely recognized as a key factor for the effectiveness of voting-based methods. However, in the context of dynamic early exiting, diversity arises in a fundamentally different way. Unlike traditional ensembles composed of independent and uniformly strong models, dynamic networks generate a sequence of intermediate predictions with varying capacity. In this setting, collaboration is inherently constrained: each exit can only aggregate information from earlier, already computed predictions, which are typically less accurate but provide complementary views.

Definition 1 (Diversity in Dynamic Neural Networks)**.** *We define diversity in a dynamic neural network as the occurrence of disagreement between classifiers at different exits, where an earlier (shallower) classifier produces the correct prediction while the current (deeper) classifier yields an incorrect one.*

In dynamic neural networks, agents (i.e., exit classifiers) differ in their parameter capacity, leading to varying levels of performance. Deeper classifiers are typically more accurate due to their access to more expressive features. As a result, conventional measures of diversity—often designed for ensembles of uniformly strong models—are not directly applicable in this asymmetric setting. A dynamic neural network can benefit from multi-agent collaboration only when the form of diversity we define is present.

We also study the performance of a simple voting mechanism in Dynamic Neural

Networks. In the voting mechanism, we employ a ranked-voting based approach, where the full logits output of each classifier is considered. Therefore, we calculate $O(\mathbf{x}) = \sum_c w_c * \theta_c(\mathbf{x})$, where here $\theta_c(\mathbf{x})$ represents the full logits output vector of a classifier c, given an item $\mathbf{x}$, and w_c is the weight assigned to the classifier c. The system then outputs the label l with the highest value in the vector $O(\mathbf{x})$.

Inspired by conformal prediction, we observe that deep neural networks inherently encode confidence information, often reflected in the magnitude of softmax outputs. Higher softmax scores are generally associated with higher confidence. However, using raw softmax values as confidence estimates can lead to overconfidence issues. We note that softmax is not directly optimized during training; instead, the loss is computed over the log-softmax (i.e., logits combined with the log function). As a result, the loss becomes highly sensitive when softmax values approach 1—small changes in softmax can lead to large changes in loss. Based on this observation, we propose to use logits directly for voting. Empirically, we find that logit-based voting significantly outperforms softmax-based voting.

Note that the full logits output represents how confident a classifier is at each possible output label. Therefore, labels which are assigned higher confidence will receive higher weight by the voting mechanism. This weight is then further multiplied by how much confidence we have on the classifier *per se* (w_c). We can employ a standard voting mechanism, where each classifier has the same weight ($w_c = 1$), or the performance of each classifier can be estimated in a validation set, in order to estimate the weights w_c. We find that in practical applications, assigning weights based on classifier performance often yields better results. This is particularly effective when certain classifiers perform significantly worse than others, as demonstrated in our experiments on the ImageNet dataset.

Contrary to usual ensemble-based approaches, we do not always consider the full set of available classifiers. As mentioned before, a dynamic neural network may decide to stop processing an item further, based on some policy. We then aggregate the outputs from the processed layers (classifiers), as the results from the

unprocessed layers are not yet available.

5.4 Experiments

In this section, we conduct extensive experiments on the CIFAR (Krizhevsky et al., 2009) and ImageNet (Deng et al., 2009) dataset, which is a representative dataset in computer vision, and pioneered the large-scale application of deep learning in image recognition. We demonstrate the effectiveness of our method on two representative dynamic early exiting network architectures, MSDNet (Huang et al., 2017b) and RANet (Yang et al., 2020b). We also present detailed synthetic experiments to further analyze the factors influencing the Bayesian updating method. Additionally, we provide oracle results, assuming ideal likelihoods, to demonstrate the potential of the Bayesian updating approach.

Datasets: Both CIFAR-100 and CIFAR-10 dataset (Krizhevsky et al., 2009) consists of 50,000 training images and 10,000 test images, each with a resolution of 32×32 pixels. CIFAR-100 includes 100 distinct classes for classification and CIFAR-10 have 10 classes. ImageNet (Deng et al., 2009), on the other hand, comprises 1.2 million training images and 50,000 validation images, each sized at 224×224 pixels, and spans 1,000 classes. We used a standard approach to select the validation set, randomly sampling 5,000 images from the training sets of CIFAR (Krizhevsky et al., 2009), and 200,000 images from the training set of ImageNet (Deng et al., 2009).

Backbone Architecture and Implementation details: Our approach can be easily integrated into any early exit network. In our experiments, we focus on two of the most representative early exit architectures: MSDNet (Huang et al., 2017b) and RANet (Yang et al., 2020b). These architectures are widely used as backbones in the field of dynamic neural networks due to their effectiveness and versatility. We employed a 7-exit MSDNet and a 6-exit RANet on the CIFAR-100 dataset, and a 5-exit MSDNet on the ImageNet dataset.

5.4.1 Main results

Results on CIFAR-100: We evaluate our method on the CIFAR-100 dataset with a 7-classifier MSDNet. We present the 'Anytime Prediction' setting, showing each classifier's accuracy and its corresponding FLOPs, a standard metric for evaluating computational cost, as detailed in Table 5.1. The results show that our Bayesian updating method significantly improves performance. By incorporating conformal prediction (CP) and utilizing softmax values, we achieved more accurate likelihood estimates, leading to further performance gains.

We also demonstrated the potential of the Bayesian updating method when accurate likelihoods are available. The results from the Oracle likelihood experiments clearly show the significant performance improvements that Bayesian updating can achieve, highlighting its strong potential in dynamic early exiting networks.

We also evaluated simple voting methods, including standard voting and weighted voting, where weights were based on the model's performance on the validation set. Despite its simplicity and limited use of information, the voting method performs well in practical deep learning applications and serves as a strong alternative when the accuracy of likelihoods limits the performance of Bayesian updating.

Results on RANet: To further demonstrate the general applicability of our approach, we conducted experiments on another representative dynamic early exiting architecture, RANet, shown in Table 5.2. Our Bayesian updating and voting methods continued to show significant improvements when applied to RANet, confirming their effectiveness across different network designs.

Results on CIFAR-10: We evaluated the performance of a 7-exit MSDNet on the CIFAR-10 dataset showing in Table 5.3. Even when each classifier already achieved high accuracy, both the Bayesian updating and voting methods were able to provide further improvements. Since CIFAR-10 has only 10 classes, the sample size for each class was relatively sufficient, leading to results from our Bayesian updating method that closely matched those obtained with oracle likelihoods.

Table 5.1: **Anytime prediction results** of a 7-exit MSDNet on CIFAR-100 (Bold means the best)

Method	1	2	3	4	5	6	7
Params($\times 10^6$)	0.30	0.65	1.11	1.73	2.38	3.05	4.00
FLOPs($\times 10^6$)	6.86	14.35	27.29	48.45	76.43	108.90	137.30
MSDNet (Huang et al., 2017b)	64.68	67.00	69.69	71.13	71.13	72.20	73.06
Bayes (Ours)	64.68	66.32	70.45	72.57	73.78	74.47	75.00
Bayes + CP (Ours)	64.68	67.80	70.85	73.04	73.96	74.61	75.02
Bayes + Oracle (Ours)	64.68	70.34	73.36	75.25	76.14	77.10	77.66
Bayes + CP + Oracle (Ours)	64.68	**71.44**	**74.75**	**76.63**	**77.68**	**78.48**	**78.98**
Logit Voting (Ours)	64.68	69.84	72.53	73.91	74.64	75.28	75.60
FLOPs W. Voting (Ours)	64.68	69.56	72.55	73.99	74.54	75.14	75.49
Acc W. Voting (Ours)	64.68	69.86	72.64	73.87	74.74	75.29	75.59

Results on ImageNet: We tested our method on the large-scale ImageNet dataset, which consists of 1,000 classes, making it a more challenging benchmark, shown in Table 5.4. For our Bayesian updating method, a higher number of classes increases the difficulty of obtaining reliable likelihood estimates through sampling. Although the ImageNet validation set contains a substantial amount of data, there are only 200 samples per class. Deriving a distribution across 1,000 classes using just 200 samples per class is particularly challenging.

We also evaluate accuracy under varying computational budgets in the dynamic inference setting in Fig 5.3. Compared to PABEE, our method does not rely on strict output consistency across layers, which allows it to maintain a better accuracy-efficiency trade-off across a wide range of FLOPs.

Table 5.2: **Anytime prediction results** of a 6-exit RANet on CIFAR-100

Exit	1	2	3	4	5	6
Params ($\times 10^6$)	0.36	0.90	1.30	1.80	2.19	2.62
FLOPs ($\times 10^6$)	8.37	21.79	32.88	41.57	53.28	58.99
RANet (Yang et al., 2020b)	64.09	67.07	69.42	69.86	71.28	71.77
Bayes (Ours)	64.09	65.69	70.30	71.61	72.92	73.52
Bayes + CP (Ours)	64.09	67.55	70.18	71.83	72.83	73.60
Bayes + Oracle (Ours)	64.09	70.62	73.00	74.34	75.57	76.15
Bayes + CP + Oracle (Ours)	64.09	**71.37**	**74.21**	**76.07**	**77.30**	**77.71**
Logit Voting (Ours)	64.09	69.47	72.19	73.04	73.89	74.21

5.4.2 Synthetic Experiments

We conducted extensive synthetic experiments to study how different factors impact the performance of Bayesian updating in practical applications. Using a Dirichlet distribution, we generated output distributions for each classifier and label, creating a confusion matrix where the correct label has a higher probability than others.

The generated confusion matrix was used as the true likelihood for the Bayesian updating process. Next, we sampled and counted from the true likelihood to simulate real-world scenarios, replicating the process of practical usage. We performed Bayesian updating with both the true distributions from the classifiers and the sampled empirical distributions. By calculating the error intervals of the posterior distributions at each step, we studied how various factors affect Bayesian updating. These factors include the number of classifiers (updating steps), the number of classes, the sample size per class, and the accuracy of the classifiers.

Note that the synthetic experiments are not using any real machine learning model and dataset. We simulate the experiment settings on ImageNet and CIFAR-100 using probability distributions to better understand the reasons behind our

Table 5.3: **Anytime prediction results** of a 7-exit MSDNet on CIFAR-10

Exit	1	2	3	4	5	6	7
Params ($\times 10^6$)	0.30	0.65	1.11	1.73	2.38	3.05	4.00
FLOPs ($\times 10^6$)	6.86	14.35	27.29	48.45	76.43	108.90	137.30
MSDNet (Huang et al., 2017b)	90.18	91.62	92.25	92.47	92.78	92.91	92.98
Bayes (Ours)	90.18	91.38	92.33	92.81	92.98	93.07	93.12
Bayes + CP (Ours)	90.18	91.34	92.48	92.92	93.02	93.14	93.07
Bayes + Oracle (Ours)	90.18	91.59	92.29	92.73	93.04	93.20	93.11
Bayes + CP + Oracle (Ours)	90.18	91.76	92.50	92.97	**93.14**	93.23	93.20
Logit Voting (Ours)	90.18	**91.83**	**92.78**	**93.01**	93.13	**93.24**	**93.23**

results in the real-world experiments. Furthermore, we simulate different settings to analyse how different parameters may impact the Bayesian-based aggregation approach.

Simulation Settings on CIFAR-100 and ImageNet: Our Bayesian updating method performed less effectively on ImageNet compared to CIFAR-100. To investigate this further, we conducted synthetic experiments, shown in Fig 5.4, to simulate the task scales of the two datasets, including the number of classes and the number of samples per class. For CIFAR-100 (100 classes), we performed 50 samples to estimate the likelihood used in our Bayesian method. For ImageNet (1,000 classes), we performed 200 samples to obtain the corresponding likelihood estimates. We reported the 90% confidence interval, selecting values within the 5%-95% error range. Our synthetic experiments, which simulated the task scale of ImageNet, show that having 1,000 classes with only 50 samples per class caused a large gap between the empirical and true distributions. Even after several steps of Bayesian updating, the error confidence interval remained wide, making the updates unstable. As a result, Bayesian updating on the simulated ImageNet task was far less reliable than when using ideal (oracle) likelihoods. In contrast, on the simulated CIFAR-100 task, the error gradually reduced after a few rounds of Bayesian updating, leading to more

Table 5.4: **Anytime prediction results** of a 5-exit MSDNet on ImageNet

Exit	1	2	3	4	5
Params ($\times 10^6$)	4.24	8.77	13.07	16.75	23.96
FLOPs ($\times 10^9$)	0.34	0.69	1.01	1.25	1.36
MSDNet (Huang et al., 2017b)	58.78	66.51	69.63	70.74	72.05
Bayes (Ours)	58.78	63.83	68.26	70.29	71.63
Bayes + CP (Ours)	58.78	65.27	68.66	70.71	71.84
Bayes + Oracle (Ours)	58.78	**68.94**	71.79	73.63	74.61
Bayes + CP + Oracle (Ours)	58.78	68.89	**72.05**	**74.31**	**75.11**
Logit Voting (Ours)	58.78	66.72	70.10	71.42	72.32
Softmax Voting (Ours)	58.78	66.03	69.51	70.96	72.02

stable results despite the limited sample size.

Simulation Analysis on Sample Size: We conducted simulation experiments to study how sample size affects the performance of Bayesian updating. Specifically, we analyzed how changes in the number of samples per class impact posterior accuracy, especially in large-class scenarios like ImageNet. Our experiments showed that smaller sample sizes per class make posterior estimates less reliable, adding more noise to the Bayesian updating process. Using a 100-class classification task as a benchmark, we employ Monte Carlo simulation to study how different parameters affect the Bayesian update process. Figure 5.5 illustrates the impact of sample size on posterior estimation. As shown, larger sample sizes improve the stability of Bayesian updating, leading to more consistent and accurate posterior estimates. This trend highlights the potential of Bayesian updating with sufficient data, demonstrating its effectiveness with larger sample sizes.

Simulation Analysis on Class Size: We also studied how classification tasks with varying numbers of classes affect the performance of the Bayesian updating method, shown in Fig 5.6. In our experiments, the number of samples per class

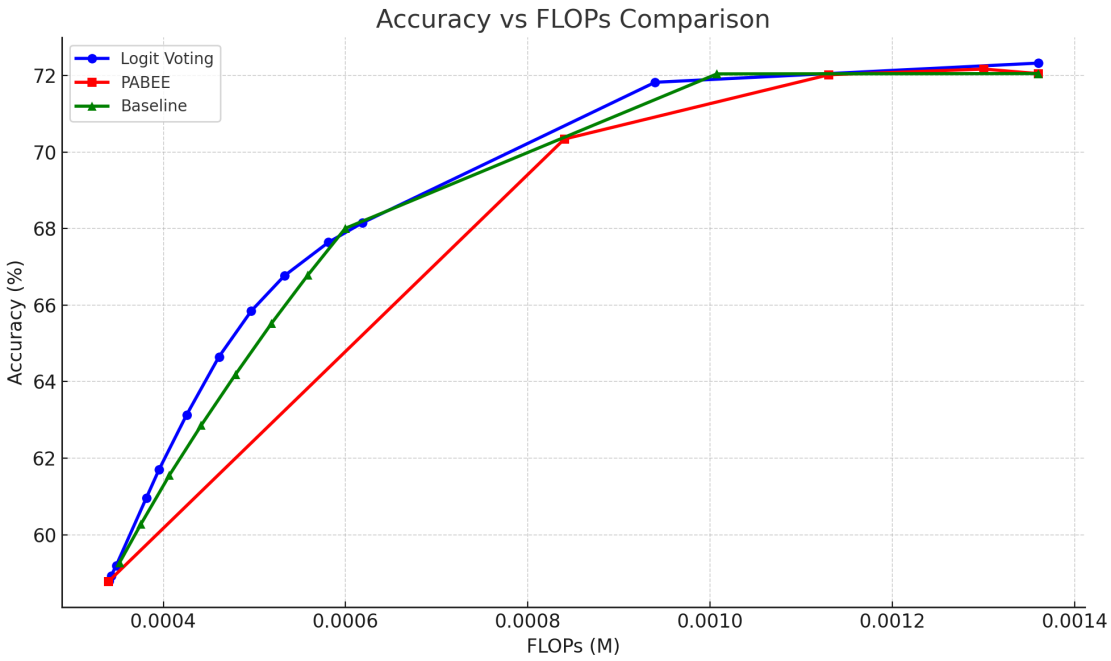


Figure 5.3: **Dynamic inference** of a 5-exit MSDNet on ImageNet

was set equal to the number of classes in each task. The results reveal that as the number of classes increases, Bayesian updating can maintain good performance if the sample size scales proportionally. This further highlights the potential of the Bayesian updating method in large-scale deep learning tasks.

Simulation Analysis on Classifier's accuracy: Our simulation experiments also examined how the accuracy of individual classifiers affects the Bayesian updating process, revealing that more accurate classifier outputs lead to faster and more reliable convergence to the correct final prediction. Unlike solely depending on the accuracy of early classifiers, we found that the precision of each classifier's information throughout the updating process is crucial.

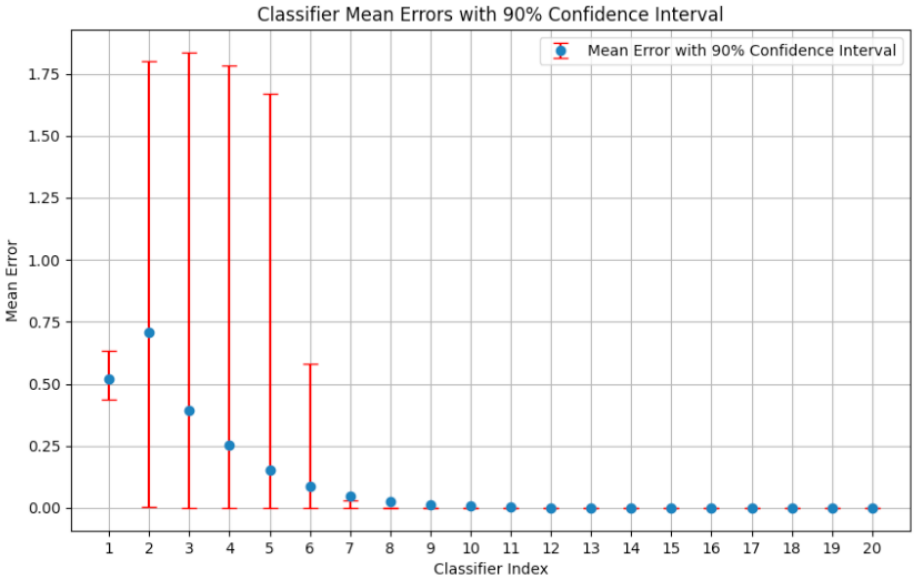
When classifiers at any stage of the Bayesian update provide accurate and reliable predictions, the method converges more quickly to a stable posterior distribution. This faster convergence reduces the probability of errors propagating and improves the overall efficiency of the model. In contrast, if any classifiers offer less accurate outputs, the Bayesian updates become less stable, requiring more iterations to reach

a confident decision, and may still risk incorrect conclusions due to accumulated noise.

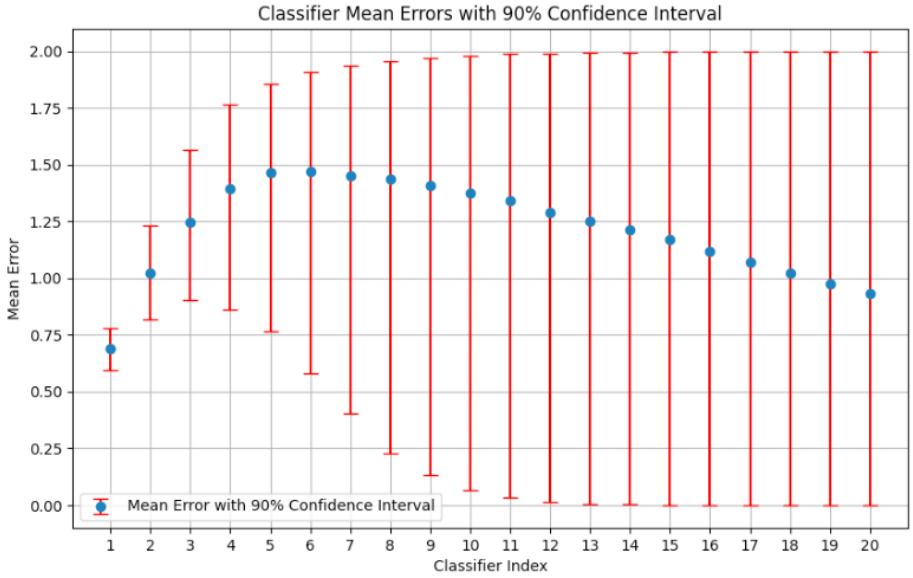
These findings, presented in Figure 5.7, emphasize that improving the accuracy of classifiers across all stages, not just the early ones, is vital for the effective performance of Bayesian updating. This highlights the importance of developing robust classifiers and using accurate likelihood estimates to accelerate convergence and enhance model reliability in practical applications.

5.5 Conclusion

In this chapter we studied the performance of two different multi-agent aggregation strategies in dynamic neural networks: a voting based approach, and a technique based on Bayesian updates. We employ synthetic experiments and study real Computer Vision problems in order to analyse these techniques in detail. We find that theoretically, if we had access to the true likelihood of classifiers, a Bayesian update based approach would perform better than voting. However, in practice the number of samples available to estimate the likelihood in real machine learning systems is limited, so we may still encounter situations where voting outperforms Bayesian updates. Therefore, in order to further enhance the potential of aggregation strategies in machine learning, we need to create mechanisms to better estimate the likelihood of classifiers (i.e., the probability of an agent having a certain output in any given real situation). Hence, we believe this work introduces a new and exciting challenge for the multi-agent community, with great potential impact in Computer Vision and Machine Learning.

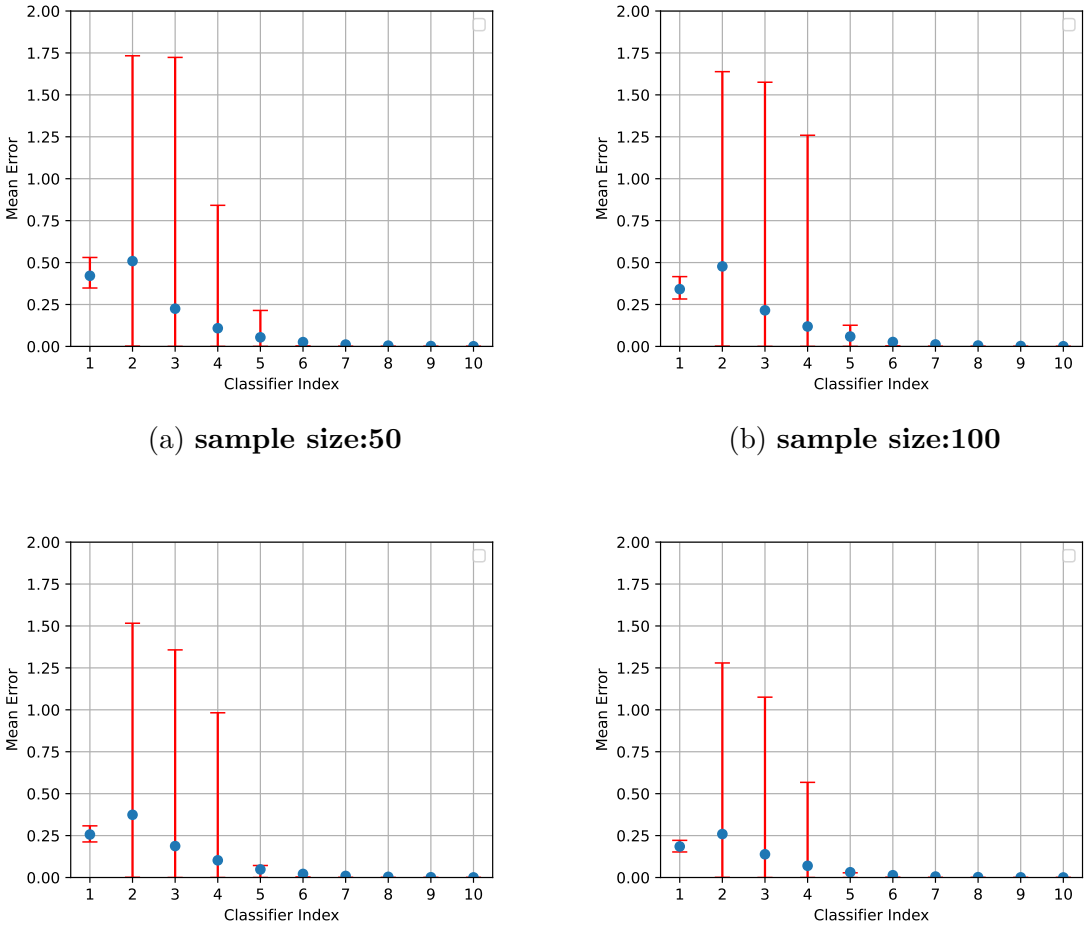


(a) **Simulation analysis of CIFAR-100 experiment setting**



(b) **Simulation analysis of ImageNet experiment setting**

Figure 5.4: **Comparison of synthetic experiments under CIFAR-100 (100 classes, 50 samples) and ImageNet (1,000 classes, 200 samples) settings.**



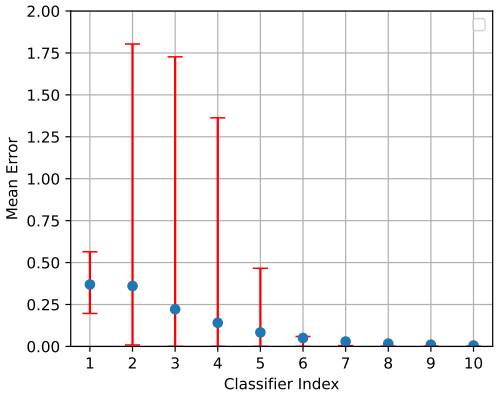
(a) **sample size:50**

(b) **sample size:100**

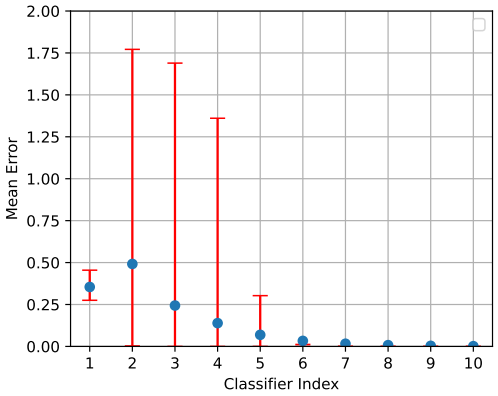
(c) **sample size:200**

(d) **sample size:400**

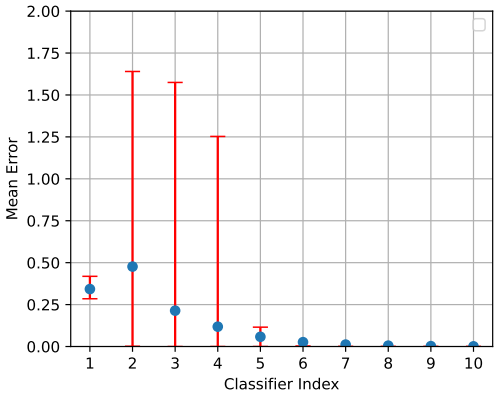
Figure 5.5: **Synthetic simulation analysis of the effect of sample size on bayesian posterior estimation**



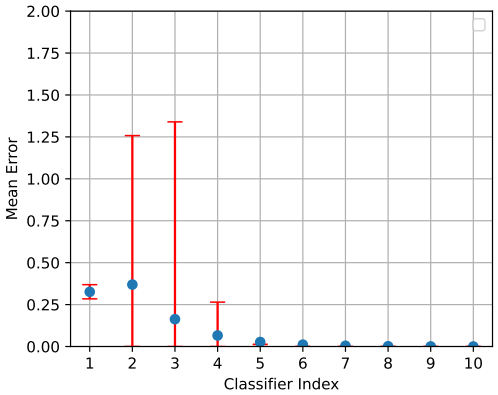
(a) **class size:10**



(b) **class size:50**

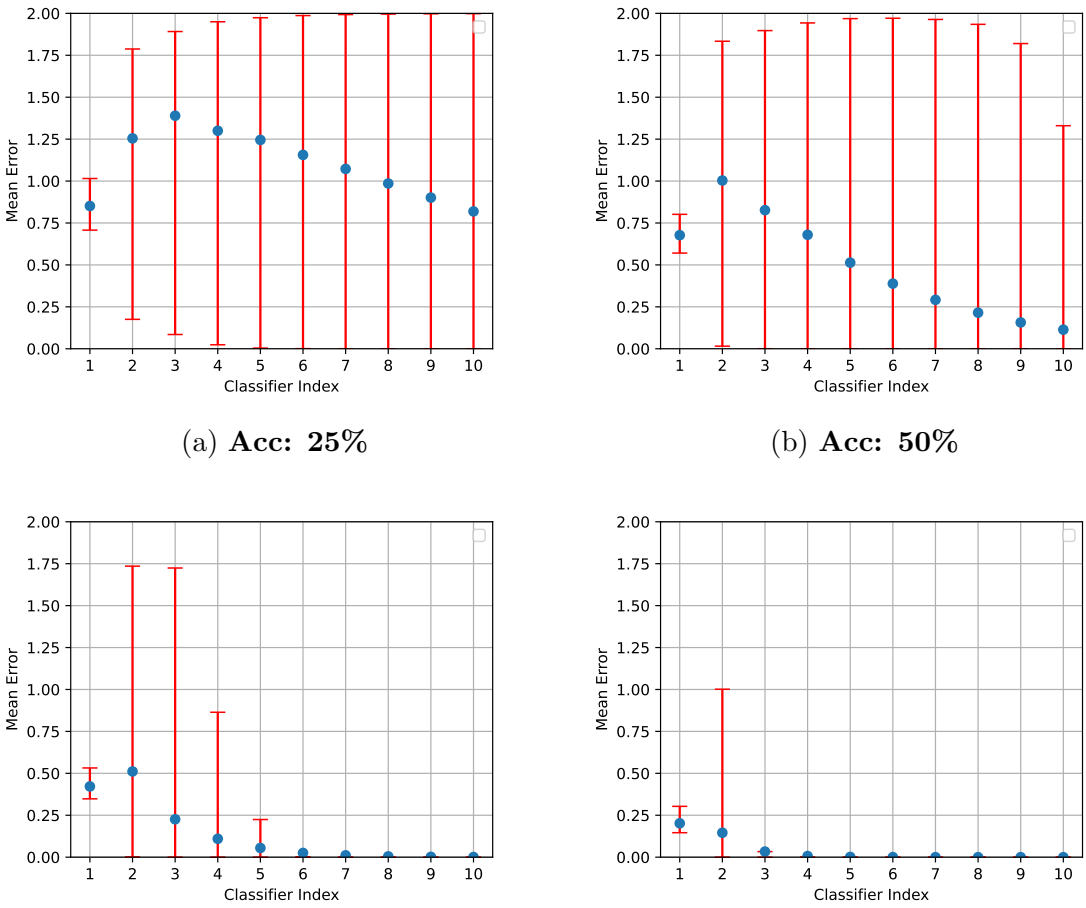


(c) **class size:100**



(d) **class size:500**

Figure 5.6: **Synthetic simulation analysis of the effect of class size on bayesian posterior estimation**



(a) **Acc: 25%** (b) **Acc: 50%**

(c) **Acc: 75%** (d) **Acc: 90%**

Figure 5.7: **Synthetic simulation analysis of the effect of classifier's accuracy on bayesian posterior estimation**

Chapter 6

Dynamic Policy–Value Monte Carlo Tree Search

Although dynamic early-exiting networks have been widely applied in many domains, their potential in deep reinforcement learning remains underexplored. Unlike conventional deep learning, deep reinforcement learning does not rely on a fixed dataset; instead, training data are generated through interactions between the model and the environment. This characteristic makes the application of dynamic early-exiting networks—whose architecture is inherently data-dependent—particularly interesting in this domain.

In this chapter, we turn to the domain of deep reinforcement learning and investigate the combination of dynamic early-exiting networks with Monte Carlo Tree Search (MCTS). We focus on the task of computer Go and propose Dynamic Policy-Value Monte Carlo Tree Search (DPV-MCTS) to investigate the effectiveness of dynamic early-exiting networks within the AlphaZero framework. We investigate DPV-MCTS across both the training stage and the inference stage. We find that DPV-MCTS achieves slightly better performance than the standard model while maintaining comparable training cost. During inference, DPV-MCTS achieves a substantial increase in win rate compared to the AlphaZero baseline.

6.1 Introduction

Many state-of-the-art game-playing AIs combine deep neural networks (DNNs) with Monte Carlo Tree Search (MCTS). This hybrid paradigm has achieved remarkable success in domains such as Go, chess, shogi, Atari, and Hex (Silver et al., 2016; Silver et al., 2017; Silver et al., 2018; Arneson et al., 2010).

The most notable milestone in this line of research is AlphaGo's victories over world-class human players. In 2016, AlphaGo defeated Lee Sedol, one of the strongest professional Go players, and in 2017 it went on to beat Ke Jie, the world's top-ranked player at the time (Silver et al., 2016). An important factor behind these achievements is the ability of deep neural networks to extract high-level representations from raw game states, providing powerful features for guiding the search process of MCTS.

Despite these impressive results, deep neural networks impose substantial computational demands, particularly in terms of GPU resources. For instance, in its matches against Lee Sedol, AlphaGo was reported to rely on a large-scale distributed system consisting of over one thousand CPUs together with about 250 GPUs (Silver et al., 2016). Although smaller neural networks can be used to make AlphaZero-style methods more lightweight, they are not always sufficient (Wu, 2019). Some complex positions still require larger networks to identify the correct move. In addition, there are some real-world scenarios, such as on mobile devices, where models must deliver strong performance under limited computational resources.

Current approaches typically rely on a fixed network architecture and vary only the number of simulations according to the available computational budget (Silver et al., 2016; Silver et al., 2017; Wu, 2019). While straightforward, this strategy is relatively basic and leaves substantial room for improvement.

As discussed in the previous chapter, dynamic early-exiting networks have attracted increasing attention in recent years in the field of deep learning, owing to their strong performance–efficiency trade-off. Specifically, dynamic early-exiting networks augment deep neural networks with intermediate classifiers, enabling

the model to adaptively choose classifiers at different depths according to input difficulty. Although dynamic early-exiting networks have been widely applied in areas such as computer vision (Huang et al., 2017b), natural language processing (Zhou et al., 2020), and multimodal learning (Fei et al., 2022), their exploration in deep reinforcement learning remains limited.

Unlike conventional deep learning tasks that rely on a fixed training dataset, deep reinforcement learning generates its training data through the model's own interactions with the environment. For example, the AlphaZero algorithm generates training samples via self-play. This characteristic makes studying dynamic early-exiting networks, which are inherently data dependent, particularly challenging in deep reinforcement learning.

In the training phase, the model must continuously interact with the environment to generate training data. Unlike conventional deep learning, where a fixed dataset is reused, this process greatly increases the computational cost. As a result, training cost becomes an important factor to consider. In AlphaZero, where training data are generated through self-play, the overall training speed directly affects the quality of the data.

In the inference phase, the characteristics of the AlphaZero algorithm also introduce important changes for dynamic early-exiting networks. In conventional deep learning tasks, dynamic early-exiting networks select different exit points for individual inputs, where exits with higher computational cost generally yield better performance. In AlphaZero, however, this raises a new question: does a larger network always guarantee better results? Under a fixed computational budget, is it more effective to use a large network for fewer searches, or a smaller network for more searches?

In this work, we explore the aforementioned challenges. First, in the training phase, we propose a knowledge distillation–based approach that leverages the high-dimensional features of a larger network to assist the training of a smaller one. Unlike conventional deep learning, we exploit the characteristics of deep

reinforcement learning by sampling directly from the model itself, which leads to more effective distillation. As a result, our training method enables Policy-Value MCTS networks with an early-exiting architecture to achieve better performance under the same training cost. In the inference phase, we focus on the AlphaZero setting. We find that dynamic early-exiting networks can achieve significant performance improvements under a fixed computational budget. The key idea is to use shallower networks, which require less computation, to allow for more simulations. This trade-off yields substantially better results compared with the baseline.

6.2 DPV-MCTS

Although Monte Carlo Tree Search (MCTS) with Policy–Value networks has achieved remarkable performance, its substantial computational cost still hinders widespread application. In this work, we focus on the deep early-exiting network component and propose a dynamic solution, Dynamic Policy–Value Monte Carlo Tree Search (DPV-MCTS). In the following sections, we present the details of our approach: Section 6.2.1 introduces the model architecture of DPV-MCTS, Section 6.2.2 describes the training methodology, and Section 6.2.3 explains the inference procedure.

6.2.1 DPV-MCTS architecture

We propose the DPV-MCTS architecture, which replaces the deep neural network in the traditional Policy–Value Monte Carlo Tree Search (PV-MCTS) framework with a dynamic early-exiting network. The overall architecture is illustrated in Figure 6.1. We adopt the ResNet backbone commonly used in state-of-the-art PV-MCTS frameworks. Our approach is general and can also be extended to other deep neural network architectures, such as Vision Transformers (Ju et al., 2025).

Specifically, We employ a ResNet as the deep neural network backbone. The

board state of Go game is taken as input, followed by an initial convolutional layer and a stack of residual blocks. Two output heads are then attached to produce the policy and value predictions. It is important to note that, as in AlphaZero, the policy and value networks share most of the backbone and differ only in the output layers, where an additional convolutional and fully connected layer are added. This design reflects the fact that policy and value estimation rely on largely shared representations. In DPV-MCTS, our deep neural network design also follows this principle. We place policy and value outputs at multiple points corresponding to different computational budgets; that is, additional policy and value heads are attached after different ResNet layers. We adopt a 15-layer ResNet as the backbone, and attach policy and value heads after the 6th, 9th, and 12th residual blocks.

It is worth noting that although our DPV–MCTS architecture introduces additional policy and value heads at intermediate layers, their computational overhead is negligible compared to that of the backbone network, i.e., the ResNet blocks. Detailed results are presented in the Experiments section.

6.2.2 Training of DPV-MCTS

Training DPV-MCTS differs substantially from conventional deep neural network training. Unlike traditional deep learning tasks that rely on a fixed dataset, deep reinforcement learning generates training data through interactions with the environment. In the case of the AlphaZero algorithm, for example, training data are obtained from self-play games. This leads to the first major challenge: the training process incurs a very high computational cost. For example, training AlphaZero required thousands of TPUs, incurring a computational cost on the order of tens of millions of dollars. In our DPV-MCTS architecture, networks at different depths provide distinct output heads. However, directly using each of them for self-play to generate training data would multiply the computational cost, which is prohibitive in the domain of deep reinforcement learning. Therefore, in our design, we use a single output head for self-play.

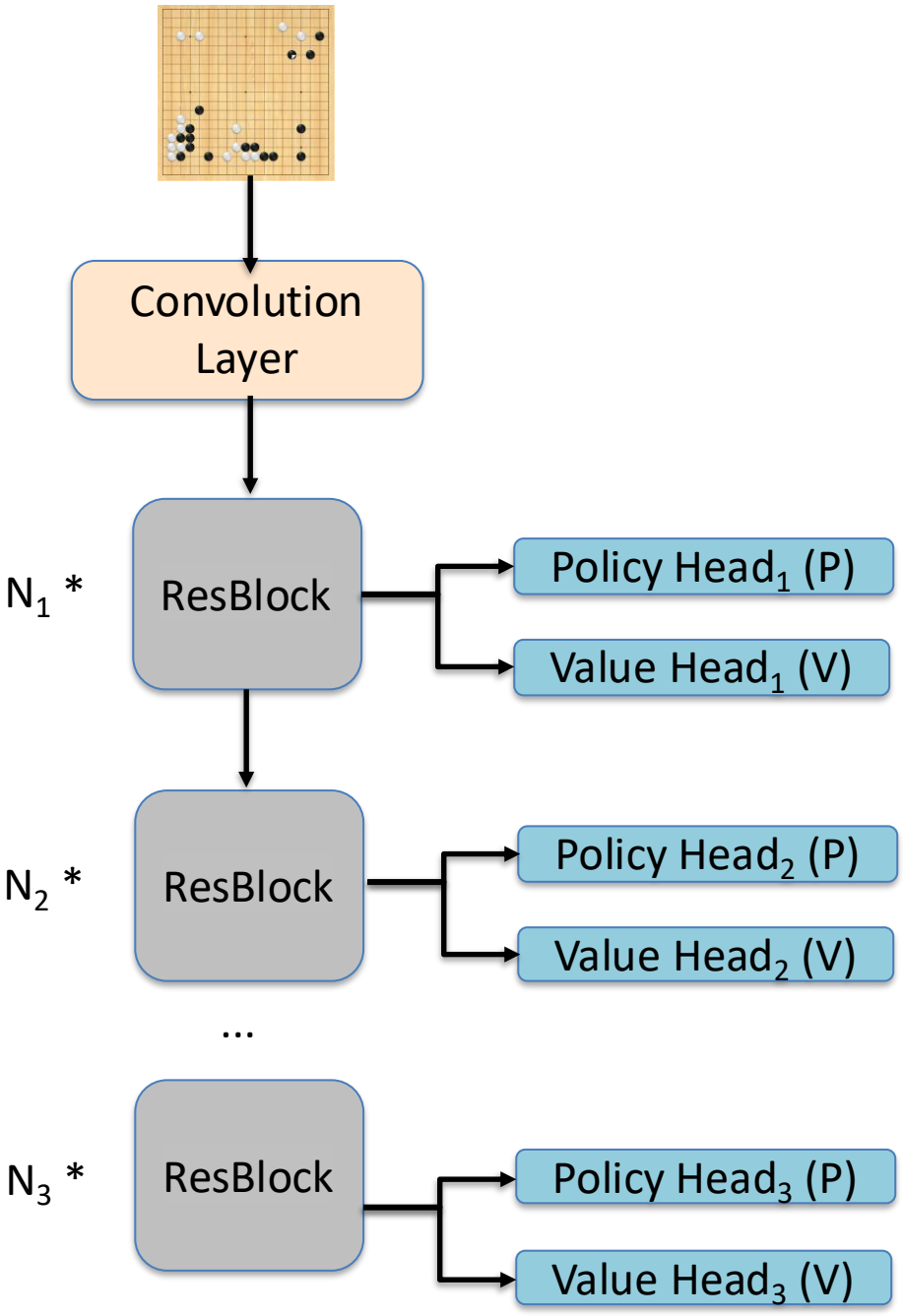


Figure 6.1: **DPV-MCTS model architecture**

Another notable distinction in AlphaZero is that the data generated through self-play are inherently influenced by the model distribution. We attempt to use different

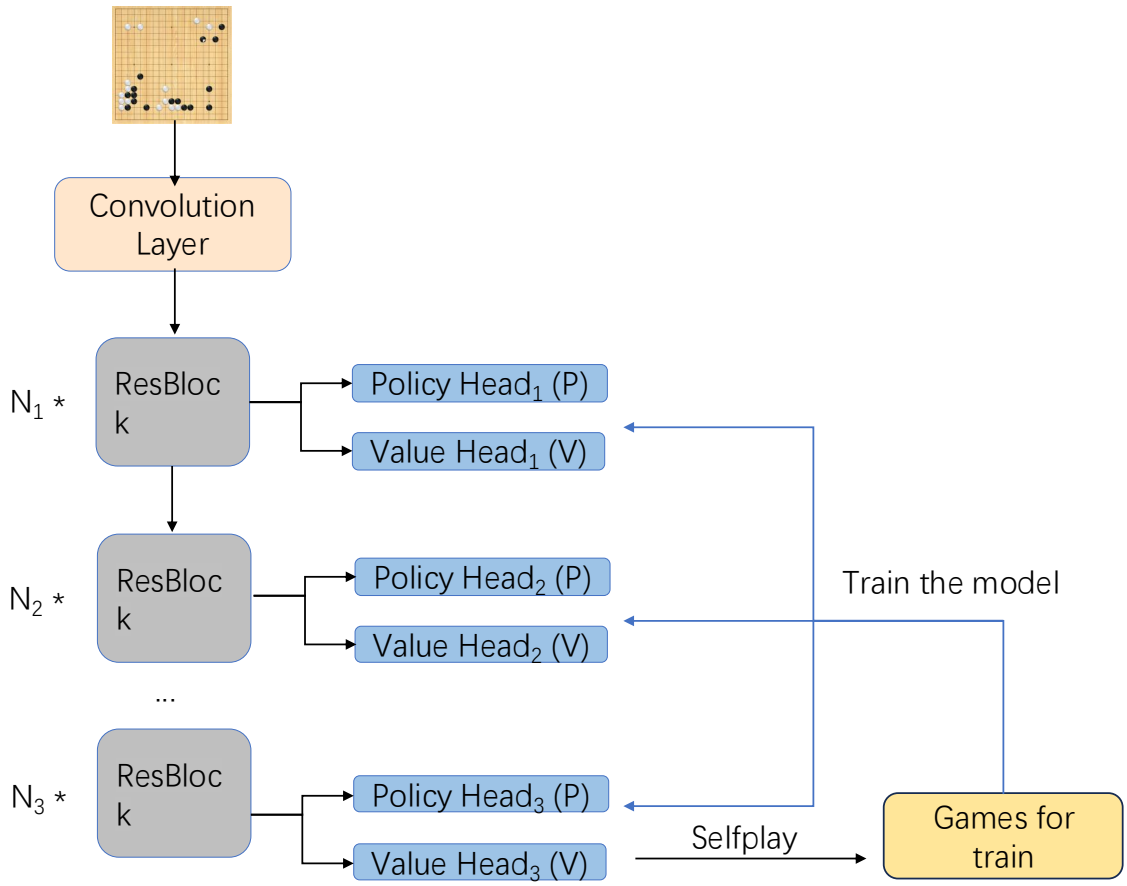


Figure 6.2: **Training Process of DPV-MCTS**

output heads for self-play at different stages of training. This design is motivated by the observation that smaller networks converge faster and, in the early stages of training, often yield better performance, thereby generating higher-quality self-play data. Moreover, employing smaller networks for self-play reduces the computational cost during training. However, we find that the model distributions associated with different output heads are not consistent, leading to discontinuities in the generated data and ultimately degrading training effectiveness. A more detailed discussion and analysis of this issue is provided in the Experiments section.

A corresponding phenomenon can be observed in the game of Go. In human play, the opening stage often follows well-established patterns, commonly referred

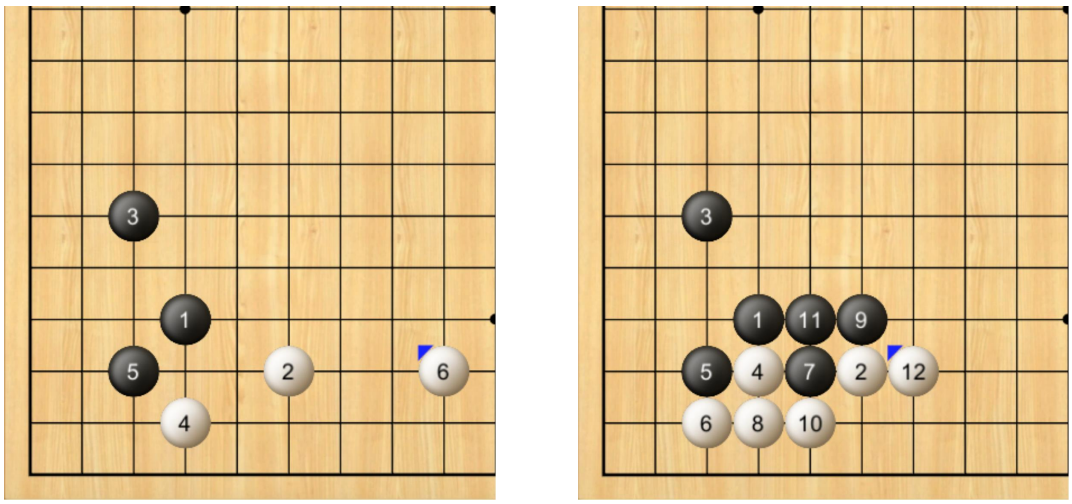


Figure 6.3: **Example of a new joseki learned by AlphaZero.** The patterns on the left illustrate traditional joseki developed through human play, while those on the right show AlphaZero's new understanding of the same openings.

to as joseki. The examples are shown in Figure 6.3. Similarly, during the training of AlphaZero, the model distribution develops its own opening patterns. Some of these align with traditional human joseki, while others reflect novel strategies introduced by AlphaZero itself, giving rise to new joseki in the era of AI-driven Go.

As noted in MSDNet (Huang et al., 2017b), directly adding intermediate classifiers to deep neural network architectures such as ResNet can degrade overall performance. The training of intermediate classifiers interferes with the optimization of the deepest classifier, thereby preventing the network from reaching its best performance.

Based on the above challenges, we propose a deeply supervised training method for DPV-MCTS. The training procedure is illustrated in Figure 6.2. Specifically, self-play is conducted using the deepest output head to generate training data, which are then shared across all output heads for training. This approach bears some similarity

to knowledge distillation in training dynamic early-exiting network; however, in our case the self-play training data are sampled directly from the model itself, resulting in a more stable knowledge distillation. To further ensure the stability of the self-play training data, we assign greater weight to the loss of the final output head when aggregating the losses from all heads, thereby prioritizing its optimization and reinforcing the reliability of the generated data.

For each output head $k \in \{1, \ldots, K\}$, the policy network is trained with a cross-entropy loss

$$\mathcal{L}_{\text{policy}}^{(k)} = -\sum_{a} \pi_a \log p_a^{(k)}, \tag{6.1}$$

where π_a is the target policy distribution and $p_a^{(k)}$ is the predicted probability of action a at head k.

The value network at head k is trained with a mean squared error (MSE) loss

$$\mathcal{L}_{\text{value}}^{(k)} = \left(v^{(k)} - z\right)^2, \tag{6.2}$$

where $v^{(k)}$ is the predicted value and z is the ground-truth outcome.

The total loss is obtained by summing over all heads:

$$\mathcal{L} = \sum_{k=1}^{K} w_k \Big(\mathcal{L}_{\text{policy}}^{(k)} + \mathcal{L}_{\text{value}}^{(k)} \Big), \tag{6.3}$$

where w_k denotes the weight assigned to head k. In particular, we assign a larger weight only to the loss of the deepest head, while keeping the weights of the shallow heads unchanged. This design prioritizes the optimization of the final head and stabilizes the self-play training data.

6.2.3 Inference of DPV-MCTS

Due to the characteristics of MCTS-based deep reinforcement learning, our DPV-MCTS network differs significantly from conventional dynamic early-exiting networks during inference. Most importantly, in traditional dynamic early-exiting networks, deeper exits that require more computation generally provide better

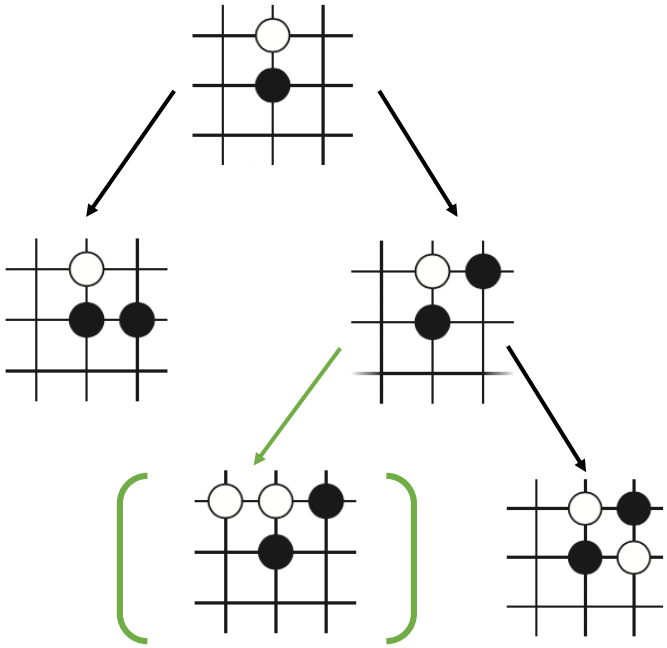


Figure 6.4: **MCTS in AlphaZero**

performance. In contrast, within the PV-MCTS framework, performance depends not only on the strength of the neural network but also on the number of simulations performed. This raises a key question: within the PV-MCTS framework, does a more powerful network always guarantee better performance?

In our experiments, we observe that under a limited computational budget, using a slightly weaker shallow classifier and allocating the saved computation to perform more simulations can significantly improve the win rate. We present this finding in more detail in the experimental section. This finding provides an important insight for the design of inference strategies in dynamic early-exiting networks for MCTS. In conventional deep learning tasks, the central idea of dynamic early-exiting is to use shallow networks for relatively simple inputs, thereby reducing computational cost. In the DPV-MCTS framework, however, we adapt the role of dynamic early-exiting networks. Given a fixed computational budget, the key question is how to allocate this budget across networks of different depths to achieve better overall performance.

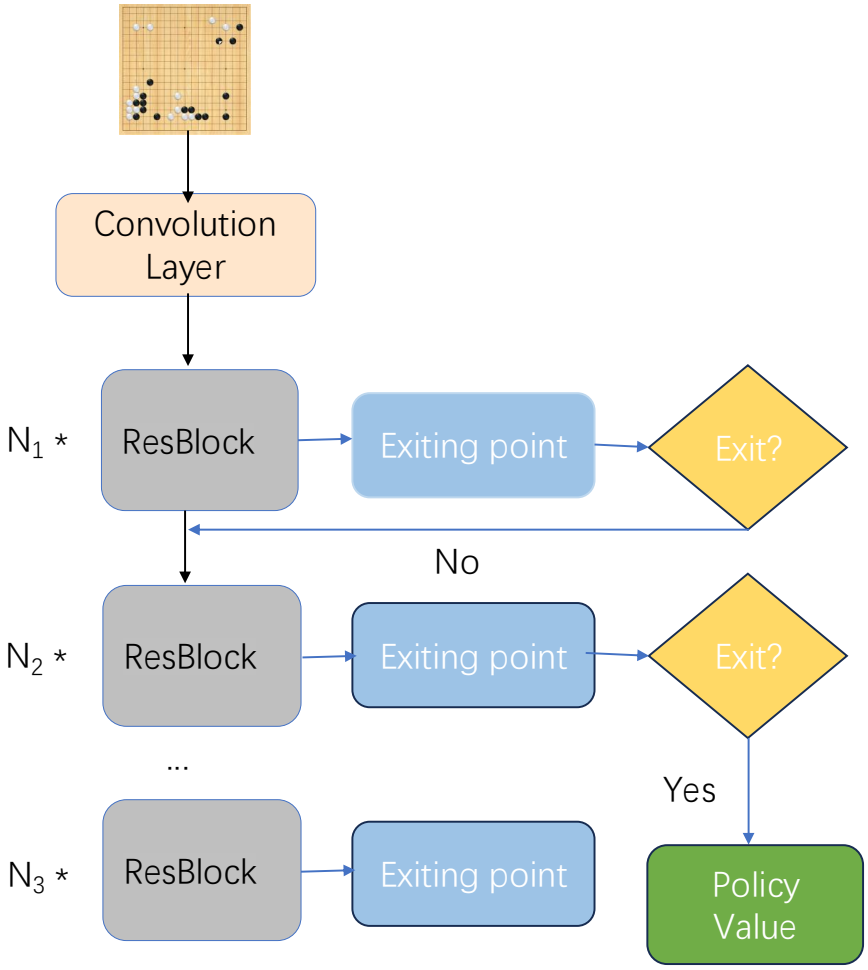


Figure 6.5: **Inference process of DPV-MCTS**

A straightforward and effective strategy is to determine the number of simulations assigned to networks of different depths directly according to the computational budget. We evaluate policy and value networks with different depths, along with their corresponding numbers of simulations under the current computational budget. Based on the outcomes of these matchups, we select the network depth that achieves the best overall performance.

Although this method demonstrates strong performance and highlights a novel application of dynamic early-exiting networks in the MCTS domain, we aim to further investigate strategies for dynamically selecting network depth according

to different game states. Within the PV-MCTS architecture, the importance of information varies across different nodes. As illustrated in Figure 6.4, MCTS employs a multi-armed bandit algorithm to select the most promising node for exploration. The newly explored node (i.e., the game state highlighted in green in the figure) is then evaluated by the value network, and the resulting estimate is propagated back to the root node.

Through this observation, we find that for the root node at the beginning of the search, the evaluations of many child nodes will be propagated back during subsequent explorations. As a result, any initial bias in the network can be corrected by the feedback from its descendants. In contrast, nodes located deeper in the search tree are typically explored later during simulations and often have fewer or even no child nodes. For these nodes, the accuracy of the network's direct evaluation becomes more critical.

Building on this property of MCTS, we design a dynamic method for selecting network depth, as shown in Figure 6.5. We use the maximum softmax probability as a measure of network confidence. For each input, the data first passes through the shallow layers of the network. At each exit point, the confidence is compared against a dynamic threshold to determine whether to exit.

Formally, the confidence at an exit point is defined as

$$c = \max_j p_j, \tag{6.4}$$

where p_j denotes the softmax probability of class j.

The dynamic threshold τ is given by

$$\tau = \tau_0 + \alpha \cdot r, \tag{6.5}$$

where τ_0 is a predefined base threshold, α is a weighting factor, and $r \in [0, 1]$ represents the proportion of computational resources that have already been consumed (i.e., the ratio of used to total computation budget). In this thesis we use a maximum number of FLOPS to set our computation budget, as is usually done in early-exiting network works.

As the search progresses, r increases, which raises the required confidence for early exiting. This encourages the use of more accurate network outputs in later stages of the search, since these evaluations are more likely to serve as the final returned values that influence the policy distribution.

The exit decision is made according to

$$\text{exit if } c \geq \tau, \quad \text{otherwise continue.} \tag{6.6}$$

6.3 Experiments

In this section, we demonstrate the effectiveness of our method on the game of Go with a 9×9 board size. We adopt the AlphaZero algorithm as our baseline. All methods are initialized with randomly initialized network weights and trained entirely through self-play data samples, without relying on any human game records.

In our experiments, the baseline network follows the AlphaZero design with a backbone consisting of 15 residual blocks. Two output heads are attached at the end of the backbone: a policy head that produces a probability distribution over all legal moves, and a value head that predicts the expected game outcome.

In our proposed DPV-MCTS architecture, we extend this baseline by introducing intermediate early exits. Specifically, additional policy and value heads are attached after the 6th, 9th, and 12th residual blocks. These intermediate exits enable the network to generate predictions at multiple depths, providing the flexibility to dynamically trade-off between computational cost and predictive accuracy during inference.

We report the computational cost of different exit points in our DPV-MCTS network in terms of floating-point operations (FLOPs). Earlier exits require substantially fewer computations compared to the baseline final exit at the 15th residual block, demonstrating the efficiency of the dynamic early-exiting design. As shown in Table 6.1, Exit 0 (after the 6th block) incurs approximately 2.90×10^8 FLOPs, which corresponds to only 40.3% of the baseline cost. Exit 1 (after the

Table 6.1: Computational cost (FLOPs) of different exits in DPV-MCTS. The final exit at the 15th ResNet block is used as the 100% baseline.

Exit	Position (ResNet Block)	FLOPs	Relative Cost
Exit 0	6	2.90×10^8	40.3%
Exit 1	9	4.33×10^8	60.2%
Exit 2	12	5.77×10^8	80.1%
Exit 3	15	7.20×10^8	100% (baseline)

9th block) requires 4.33×10^8 FLOPs (60.2%), while Exit 2 (after the 12th block) consumes 5.77×10^8 FLOPs (80.1%). Finally, the full baseline network with Exit 3 (after the 15th block) requires 7.20×10^8 FLOPs, serving as the 100% reference point.

Next, we present the performance of our method during the training phase and the inference phase separately. We evaluate the performance of our model by measuring its winning rate in game matches, with each comparison consisting of 500 games.

6.3.1 Training Process Experiments

During training, our goal is to obtain a dynamic early-exiting variant of the Policy–Value MCTS network without incurring additional training cost. This constraint is critical in the AlphaZero framework. Training data are generated through self-play, which makes the process computationally expensive. To address this, we explore several strategies during training. Our goal is for the DPV-MCTS model to achieve performance comparable to the baseline under the same number of self-play games.

Specifically, training begins after the generation of 3,000 self-play games. Thereafter, for every additional 500 games, the model is trained for 1,000 steps. After each training cycle, we compare the updated model with the previous version.

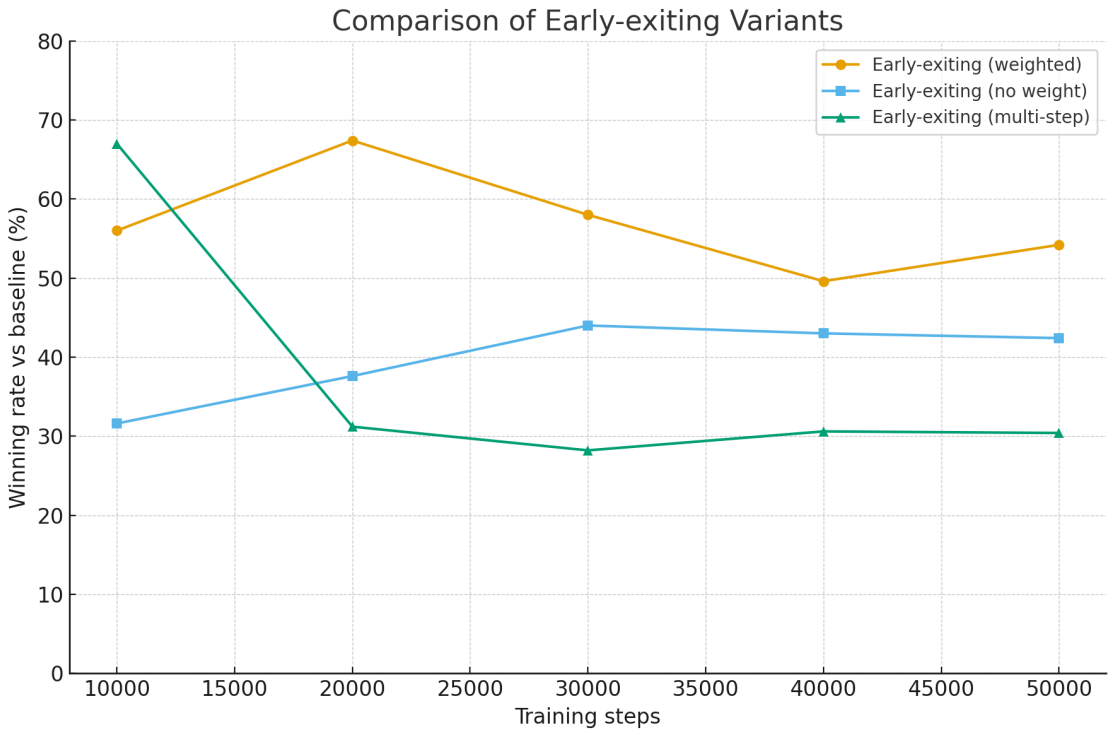


Figure 6.6: Winning rate against the baseline (i.e., AlphaZero with a shared 15-layer ResNet backbone followed by a policy head and a value head) across different training steps for three variants of early-exiting training: weighted training, no-weight training, and multi-step training. The experiments are conducted under a budget of 100 simulations for the full network.

If the new model achieves a winning rate above 55%, its parameters replace the old ones. Otherwise, the parameters of the previous model are retained.

We evaluate the trained model using its deepest exit. The comparison is made against a baseline model trained for the same number of steps, namely a Policy–Value MCTS with a 15-layer ResNet backbone. Performance is measured by the winning rate. We focus on the deepest exit because it usually reflects the maximum potential performance of a dynamic early-exiting network.

From the experimental results in Figure 6.7, we observe that our training method (weighted training) achieves performance comparable to the baseline. In most cases, it even provides slight improvements. These results show that our approach

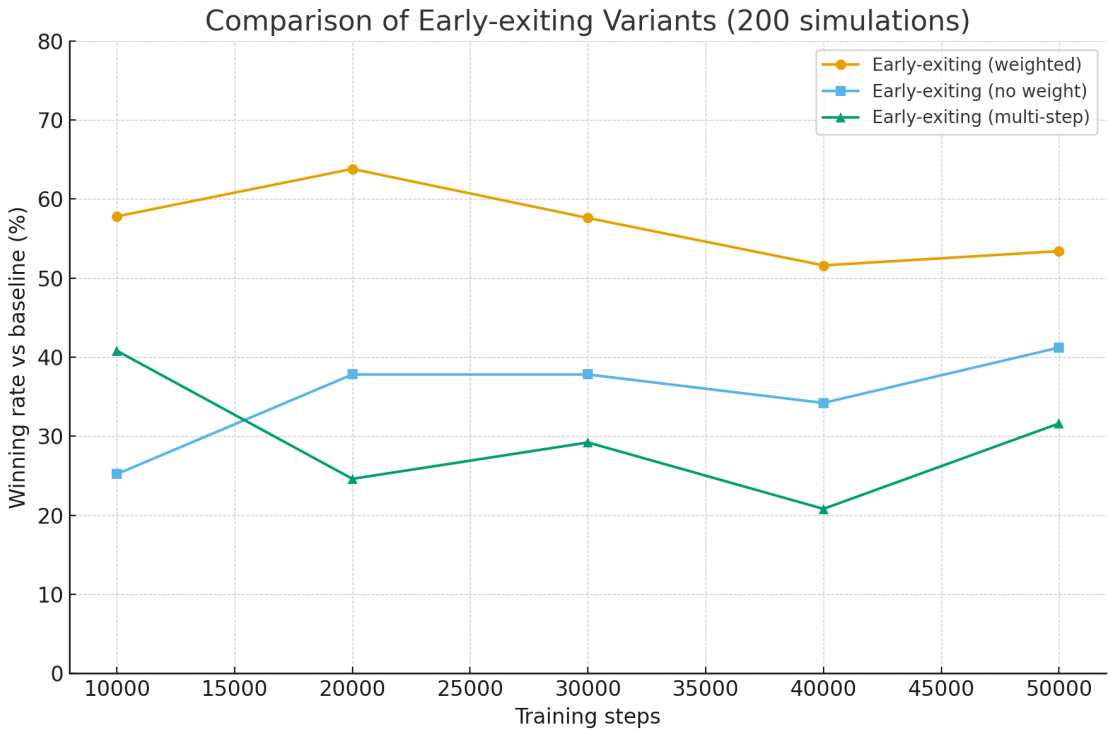


Figure 6.7: Winning rate against the baseline (i.e., AlphaZero with a 15-layer ResNet backbone) across different training steps for three variants of early-exiting training: weighted training, no-weight training, and multi-step training. The experiments are conducted under a budget of 200 simulations for the full network.

effectively alleviates the suboptimal training performance observed when directly applying a dynamic early-exiting architecture as the backbone network.

We also compare our approach with alternative training methods. The results show that if the gradients of the deepest network are not assigned greater weight, the training performance degrades significantly, shown as the blue line. This indicates that emphasizing the deepest exit is crucial for maintaining the quality of the self-play game data.

In addition, we compare our approach with a multi-step training baseline that generates self-play training data using different exits at different stages. Specifically, the training process is evenly divided into four intervals, with each interval using a different exit point. We refer to this training strategy as multi-step, shown as

the green line. The results show that when the exit used to generate training data is switched, the model's performance drops significantly. This degradation occurs because switching exits changes the model distribution, which in turn alters the distribution of the training data and leads to instability in training. Therefore, when training DPV-MCTS networks, self-play should be conducted using a single exit.

In Figure 6.6, we present the results with 100 simulations, while Figure 6.7 shows the results with 200 simulations, to demonstrate the stability of our performance across different simulation counts.

6.3.2 Inference Process Experiments

We also conducted experiments with DPV-MCTS during the inference phase. For comparison, we used the deepest classifier of DPV-MCTS as the baseline, so as to rule out advantages gained specifically during training. In addition, our training method can also be applied independently as an auxiliary tool to accelerate the training process.

As a first step, we evaluated the performance of various shallow exit points in comparison with the deepest exit point of the network. The results are shown in Table 6.2. We present the performance of different exits under the same number of simulations. The results show that, although fewer network parameters are used, the performance decreases only slightly. For example, exit 0 uses only about 40% of the parameters of the full network, yet it still achieves a win rate of around 40%, which is close to the performance of the full model. This finding is consistent with the experimental results reported in the previous chapters. It is worth noting that under different numbers of simulations, shallower exits sometimes even slightly outperform the deepest exit. For example, this can be observed when the number of simulations is 100. In our experiments, we trained the models and evaluated their improvements over previous baselines using 200 simulations.

However, when constrained to the same computational budget, we found that

allocating resources to a larger number of simulations with shallower exits leads to a significant improvement in performance. In particular, compared with the deepest network, selecting an appropriate shallow exit achieved a win rate exceeding 70%.

Although selecting an appropriate shallow exit yields significant performance gains, we further explore more fine-grained strategies for exit point selection. We found that directly applying dynamic early-exiting networks, where confidence is measured by the softmax output with a predefined threshold, does not yield further improvements. This is because, under limited computational budgets, the number of simulations is typically small, and in such cases the simulation count has a substantial impact on performance. By analyzing the PUCT formula used in the MCTS of AlphaZero, we observe that network accuracy becomes increasingly important in later simulations. At the beginning of the search, even if the initial distribution predicted by the network is biased, subsequent simulations can correct it. In contrast, the network outputs assigned to leaf nodes in the final stages of the search directly determine the returned values and have no opportunity to be revised. Our dynamic thresholding method achieves further improvements compared with selecting the single best-performing exit point.

As shown in Table 6.3, dynamically selecting exit points provides additional performance gains compared with choosing a single best-performing exit. However, relying solely on softmax confidence with a fixed threshold for exit selection does not lead to further improvement. This is because, when the number of simulations is small, the benefit from simulations outweighs the performance of the model. Using a shallow network for more simulations is therefore much more effective than using a deeper network with fewer simulations. Our comparisons show that simply relying on a fixed threshold to decide whether to use deeper networks performs poorly under a limited computational budget. In this case, the gain from a more accurate model is smaller than the gain from more simulations with a slightly weaker model. By contrast, our method dynamically adjusts the threshold (in equation 6.5) based on the computation budget used so far and achieves much better results than a fixed-

Table 6.2: Performance comparison of shallow exits when playing against the deepest output under equal numbers of simulations and equal computational budgets.

Simulation	**Setting**	**Exit 0**	**Exit 1**	**Exit 2**
100	Same number of simulations	40.0%	54.0%	52.8%
	Same computation budgets	73.4%	70.6%	58.4%
200	Same number of simulations	36.8%	47.8%	47.4%
	Same computation budgets	67.4%	72.0%	60.2%

Table 6.3: Performance comparison of dynamic exit selection methods when playing against the single best-performing exit under different computation budgets. Simulation count represents the computational budget used by the full network with the corresponding number of simulation.

Simulation Count	**Dynamic**	**Dynamic (w/o dynamic threshold)**
100	52.6%	46.6%
200	53.4%	47.2%

threshold strategy. Under the AlphaZero framework, early nodes can still be refined by subsequent simulations, and additional simulations often bring larger potential gains at the beginning. In contrast, simulations performed at later stages are more likely to determine the final outcome and therefore require more accurate predictions. Accordingly, the threshold is adjusted dynamically to balance potential gains and computational cost across different stages.

In practice, we can select the strongest method under a given computational budget through match play. For example, when resources are very limited, we could fix the shallowest classifier to allow more simulations. For dynamic selection, we determine the best parameter settings through self-play evaluation.

6.4 Conclusion

In this chapter, we propose DPV-MCTS, extending the application of dynamic early-exiting networks to the domain of deep reinforcement learning. Specifically, we replace the deep neural network in PV-MCTS with a dynamic early-exiting architecture. On the training side, we introduce a deeply supervised learning–based method to improve optimization. On the inference side, we propose a strategy for selecting network depth under a fixed computational budget, which achieves strong performance. Furthermore, by integrating the multi-armed bandit algorithm within MCTS, we design a dynamic confidence-threshold adjustment mechanism that adaptively selects network depth, leading to additional performance gains.

Chapter 7

Conclusions and Future Works

7.1 Summary of Contributions

This thesis focuses on the fundamental problems of dynamic early-exiting networks. These foundational issues are highly influential and have the potential to extend to other dynamic neural network architectures. We investigate the problem of gradient conflicts among different classifiers during training, the collaboration among classifiers during inference, and the application of dynamic early-exiting networks in Monte Carlo Tree Search.

7.1.1 Damping unnecessary gradient during training process of dynamic early-exiting networks

In Chapter 4, we investigate the problem of gradient conflicts among different classifiers during the training of dynamic early-exiting networks. Unlike existing approaches that primarily focus on balancing gradient trade-offs, our method addresses the necessity of gradients by introducing a mechanism to damp unnecessary ones, thereby significantly improving training performance.

Specifically, we propose a damping loss by introducing an additional damping neuron in the fully connected layer of each classifier. When a classifier already

achieves a high softmax score on the current data sample, the damping neuron receives a larger gradient, which suppresses unnecessary updates. Conversely, when the classifier performs poorly and the softmax score is low, only a small gradient is assigned to the damping neuron, thereby limiting its effect at this stage.

To further improve gradient allocation, we design the power-sqrt loss, which redistributes gradients by emphasizing classifiers that perform relatively better than others. This joint consideration of all classifiers enables a more effective damping mechanism and enhances the overall training process. In addition, we utilize the values of the damping neuron to assign dynamic weights across classifiers, showing that our approach is compatible with current methods.

7.1.2 Multi-Agent Collaboration in Dynamic Early-Exiting Networks During Inference

In Chapter 5, we explore the collaboration among different classifiers in dynamic early-exiting networks. Since classifiers at different exits are based on subnetworks of varying parameter sizes, those with larger capacity typically achieve stronger performance. The mainstream inference strategy, however, is to adopt the output of the deepest classifier as the final prediction. This approach disregards the outputs of earlier classifiers, even though these subnetworks—despite being weaker—still provide useful information. From a Bayesian perspective, such observations should contribute to improving the final decision.

Building on this observation, we propose a Bayesian updating–based collaboration method. In this approach, we estimate the likelihood of each classifier's output for different labels by counting and analyzing results from a validation set, and then apply Bayesian updating to integrate these likelihoods. However, synthetic experiments reveal that when the validation set is limited, likelihood estimation becomes unreliable, which in turn restricts the effectiveness of Bayesian updating in certain scenarios.

To overcome this limitation, we introduce a conformal prediction–based method

to enhance likelihood estimation. Conformal prediction has demonstrated strong performance in large-scale deep learning tasks, and we leverage its more accurate confidence scores to refine the confidence estimates within the Bayesian framework.

In addition, we propose a logit voting strategy as an alternative. Although this method uses less information and may have lower theoretical potential than Bayesian updating, it performs well in practice and serves as a practical fallback when reliable likelihood estimation is difficult to obtain.

7.1.3 Dynamic Early-Exiting Networks in Monte Carlo Tree Search

In Chapter 6, we investigate the application of dynamic early-exiting networks in the domain of deep reinforcement learning. As a technique that adaptively adjusts network depth according to individual data samples, dynamic early-exiting becomes particularly appealing in reinforcement learning, where no fixed training dataset exists and training data are generated through continuous interaction between the model and the environment. In this chapter, we focus on the task of computer Go and introduce the DPV-MCTS architecture.

We address these challenges from both the training and inference perspectives. In the training phase, we propose a knowledge distillation–based method in which the high-dimensional representations learned by a deeper network are used to guide the training of shallower networks. Unlike conventional deep learning, our approach leverages the unique characteristics of deep reinforcement learning by sampling data directly from the model, resulting in more effective distillation. This strategy allows policy–value MCTS networks with an early-exiting architecture to achieve stronger performance under the same training cost.

In the inference phase, we further build on the properties of the AlphaZero algorithm and propose a dynamic network selection strategy that adapts to a fixed computational budget. Experimental results demonstrate that this method yields significant performance improvements compared with the baseline.

7.2 Future Works

In this section, we discuss future research directions that can be further extended from the work presented in this thesis.

7.2.1 Extending the Damping Training Mechanism to Multi-Task Learning

Our damping training mechanism has been extensively analyzed both experimentally and theoretically within the context of dynamic early-exiting networks. This idea also holds the potential to be extended to related research areas, such as multi-task learning.

In the domain of multi-task learning, the question of whether a gradient is necessary also arises. However, unlike in dynamic early-exiting networks, where all classifiers are trained on the same task and share the same dataset—making it relatively straightforward to determine the degree of damping—multi-task learning involves gradients originating from different tasks. This diversity introduces additional challenges in defining the necessity of gradients.

In future work, we plan to investigate this issue further, with the aim of extending the damping training mechanism to the domain of multi-task learning.

7.2.2 Leveraging Advanced DNN Confidence Estimation for More Stable Bayesian Likelihoods

In studying the collaboration among different classifiers in dynamic early-exiting networks, we identified the accurate estimation of Bayesian likelihoods as a major challenge. This issue becomes particularly critical in complex tasks, such as ImageNet classification, which involves 1,000 categories. The difficulty is closely tied to the problem of confidence estimation in deep neural networks, for which no lightweight and reliable solutions currently exist (Papamarkou et al., 2024).

A potential alternative is to use Bayesian neural networks (Blundell et al., 2015; Gal et al., 2016). However, Bayesian networks require sampling from the model distribution, such as Markov Chain Monte Carlo (MCMC) (Neal, 1993), which greatly increases training cost. In dynamic early-exiting networks, the target applications are often computationally demanding, making training cost a critical concern.

As a result, although Bayesian updating holds strong potential, its performance in large-scale tasks remains limited. In future work, we plan to further investigate lightweight and accurate confidence estimation techniques for deep neural networks to help mitigate this challenge.

7.2.3 Exploring Bayesian Updating Methods in Deep Reinforcement Learning

In this thesis, while exploring the problem of collaboration among multiple classifiers in dynamic early-exiting networks, we observed that in deep learning, the combination of limited validation sets and complex task categories often introduces substantial noise into the likelihood distribution estimated from samples. In contrast, deep reinforcement learning provides a potential solution, as likelihoods can be sampled directly from the model itself. We plan to investigate this direction further in future work.

7.2.4 Exploring Fast Training in AlphaZero

In this thesis, we explored training from scratch using the AlphaZero framework. Later engineering advances introduced many techniques to improve AlphaZero's strength. For example, KataGo (Wu, 2019) can create special training scenarios. On a 9×9 board, it may start with five stones placed and give a very large komi, such as 80 points. This forces the model to win only by completely killing the opponent. Such settings strengthen aggressive play. These engineering techniques are complex.

It is difficult to adapt early-exiting to all of them. Therefore, it is valuable to take a model trained with these techniques as a base, and then further train a dynamic early-exiting network architecture.

References

Addad, Youva, Alexis Lechervy, and Frédéric Jurie (2025). "Balancing Accuracy and Efficiency in Budget-Aware Early-Exiting Neural Networks". In: *International Conference on Pattern Recognition*. Springer, pp. 17–31.

Angelopoulos, Anastasios N and Stephen Bates (2021). "A gentle introduction to conformal prediction and distribution-free uncertainty quantification". In: *arXiv preprint arXiv:2107.07511*.

Angelopoulos, Anastasios Nikolas, Stephen Bates, Michael Jordan, and Jitendra Malik (n.d.). "Uncertainty Sets for Image Classifiers using Conformal Prediction". In: *International Conference on Learning Representations*.

Arneson, Broderick, Ryan B Hayward, and Philip Henderson (2010). "Monte Carlo tree search in Hex". In: *IEEE Transactions on Computational Intelligence and AI in Games* 2.4, pp. 251–258.

Bajpai, Divya Jyoti and Manjesh Hanawal (2024). "CeeBERT: Cross-Domain Inference in Early Exit BERT". In: *Findings of the Association for Computational Linguistics ACL 2024*, pp. 1736–1748.

Blundell, Charles, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra (2015). "Weight uncertainty in neural network". In: *International conference on machine learning*. PMLR, pp. 1613–1622.

Bolukbasi, Tolga, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama (2017). "Adaptive neural networks for efficient inference". In: *International Conference on Machine Learning*. PMLR, pp. 527–536.

Carmo Alves, Matheus Aparecido do, Amokh Varma, Yehia Elkhatib, and Leandro Soriano Marcolino (2024). "It Is Among Us: Identifying Adversaries in Ad-hoc Domains using Q-valued Bayesian Estimations." In: *AAMAS*, pp. 472–480.

Chen, Xinshi, Hanjun Dai, Yu Li, Xin Gao, and Le Song (2020). "Learning to stop while learning to predict". In: *International conference on machine learning*. PMLR, pp. 1520–1530.

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei (2009). "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, pp. 248–255.

Ding, Chuntao, Zhichao Lu, Shangguang Wang, Ran Cheng, and Vishnu Naresh Boddeti (2023a). "Mitigating task interference in multi-task learning via explicit task routing with non-learnable primitives". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7756–7765.

Ding, Tiffany, Anastasios Angelopoulos, Stephen Bates, Michael Jordan, and Ryan J Tibshirani (2023b). "Class-conditional conformal prediction with many classes". In: *Advances in neural information processing systems* 36, pp. 64555–64576.

Dosovitskiy, Alexey (2020). "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929*.

Elbayad, Maha, Jiatao Gu, Edouard Grave, and Michael Auli (2020). "Depth-adaptive Transformer". In: *ICLR 2020-Eighth International Conference on Learning Representations*, pp. 1–14.

Fei, Zhengcong, Xu Yan, Shuhui Wang, and Qi Tian (2022). "Deecap: Dynamic early exiting for efficient image captioning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12216–12226.

Gal, Yarin and Zoubin Ghahramani (2016). "Dropout as a bayesian approximation: Representing model uncertainty in deep learning". In: *international conference on machine learning*. PMLR, pp. 1050–1059.

Gao, Chao, Ryan Hayward, and Martin Müller (2017). "Move prediction using deep convolutional neural networks in Hex". In: *IEEE Transactions on Games* 10.4, pp. 336–343.

Gao, Xiangxiang, Yue Liu, Tao Huang, and Zhongyu Hou (2023). "PF-BERxiT: Early exiting for BERT with parameter-efficient fine-tuning and flexible early exiting strategy". In: *Neurocomputing* 558, p. 126690.

Gelly, Sylvain, Levente Kocsis, Marc Schoenauer, Michele Sebag, David Silver, Csaba Szepesvári, and Olivier Teytaud (2012). "The grand challenge of computer Go: Monte Carlo tree search and extensions". In: *Communications of the ACM* 55.3, pp. 106–113.

Ghiasi, Golnaz, Barret Zoph, Ekin D Cubuk, Quoc V Le, and Tsung-Yi Lin (2021). "Multi-task self-training for learning general representations". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 8856–8865.

Gong, Cheng, Yao Chen, Qiuyang Luo, Ye Lu, Tao Li, Yuzhi Zhang, Yufei Sun, and Le Zhang (2024). "Deep Feature Surgery: Towards Accurate and Efficient Multi-exit Networks". In: *European Conference on Computer Vision*. Springer, pp. 435–451.

Grubb, Alex and Drew Bagnell (2012). "Speedboost: Anytime prediction with uniform near-optimality". In: *Artificial Intelligence and Statistics*. PMLR, pp. 458–466.

Guo, Xiaolin, Fang Dong, Dian Shen, Zhaowu Huang, and Jinghui Zhang (2024). "Resource-efficient dnn inference with early exiting in serverless edge computing". In: *IEEE Transactions on Mobile Computing*.

Han, Song, Huizi Mao, and William J Dally (2015). "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding". In: *arXiv preprint arXiv:1510.00149*.

Han, Yizeng, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang (2021). "Dynamic neural networks: A survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.11, pp. 7436–7456.

Han, Yizeng, Yifan Pu, Zihang Lai, Chaofei Wang, Shiji Song, Junfeng Cao, Wenhui Huang, Chao Deng, and Gao Huang (2022). "Learning to weight samples for dynamic early-exiting networks". In: *European conference on computer vision*. Springer, pp. 362–378.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

He, Yang, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang (2019). "Filter pruning via geometric median for deep convolutional neural networks acceleration". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4340–4349.

Hehn, Thomas M, Julian FP Kooij, and Fred A Hamprecht (2020). "End-to-end learning of decision trees and forests". In: *International Journal of Computer Vision* 128.4, pp. 997–1011.

Helmbold, David (2009). "All-moves-as-first heuristics in monte-carlo go". In.

Howard, Andrew G, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam (2017). "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861*.

Hu, Yuzheng, Ruicheng Xian, Qilong Wu, Qiuling Fan, Lang Yin, and Han Zhao (2024). "Revisiting scalarization in multi-task learning: A theoretical perspective". In: *Advances in Neural Information Processing Systems* 36.

Huang, Chen, Simon Lucey, and Deva Ramanan (2017a). "Learning policies for adaptive tracking with deep feature cascades". In: *Proceedings of the IEEE international conference on computer vision*, pp. 105–114.

Huang, Gao, Danlu Chen, Tianhong Li, Felix Wu, Laurens Van Der Maaten, and Kilian Q Weinberger (2017b). "Multi-scale dense networks for resource efficient image classification". In: *arXiv preprint arXiv:1703.09844*.

Huang, Gao, Zhuang Liu, Geoff Pleiss, Laurens Van Der Maaten, and Kilian Q Weinberger (2019). "Convolutional networks with dense connectivity". In: *IEEE transactions on pattern analysis and machine intelligence* 44.12, pp. 8704–8716.

Huang, Shih-Chieh, Broderick Arneson, Ryan B Hayward, Martin Müller, and Jakub Pawlewicz (2013). "MoHex 2.0: a pattern-based MCTS Hex player". In: *International Conference on Computers and Games*. Springer, pp. 60–71.

Hubara, Itay, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio (2016). "Binarized neural networks". In: *Advances in neural information processing systems* 29.

Jiang, Yikun, Huanyu Wang, Lei Xie, Hanbin Zhao, Hui Qian, John Lui, et al. (2024). "D-llm: A token adaptive computing resource allocation strategy for large language models". In: *Advances in Neural Information Processing Systems* 37, pp. 1725–1749.

Jin, Qing, Linjie Yang, and Zhenyu Liao (2020). "Adabits: Neural network quantization with adaptive bit-widths". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2146–2156.

Ju, Yan-Ru, Tai-Lin Wu, Chung-Chin Shih, and Ti-Rong Wu (Aug. 2025). "Bridging Local and Global Knowledge via Transformer in Board Games". In: *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*. Ed. by James Kwok. Main Track. International Joint Conferences on Artificial Intelligence Organization, pp. 7446–7454. DOI: 10.24963/ijcai.2025/828. URL: https://doi.org/10.24963/ijcai.2025/828.

Jung, Sangil, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi (2019). "Learning to quantize deep networks by optimizing quantization intervals with task loss". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4350–4359.

Kaya, Yigitcan, Sanghyun Hong, and Tudor Dumitras (2019). "Shallow-deep networks: Understanding and mitigating network overthinking". In: *International conference on machine learning*. PMLR, pp. 3301–3310.

Kim, Hoki, Woojin Lee, and Jaewook Lee (2021). "Understanding catastrophic overfitting in single-step adversarial training". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 9, pp. 8119–8127.

Kocsis, Levente and Csaba Szepesvári (2006). "Bandit based monte-carlo planning". In: *European conference on machine learning*. Springer, pp. 282–293.

Kouris, Alexandros, Stylianos I Venieris, Stefanos Laskaridis, and Nicholas Lane (2022). "Multi-exit semantic segmentation networks". In: *European Conference on Computer Vision*. Springer, pp. 330–349.

Krizhevsky, Alex, Geoffrey Hinton, et al. (2009). "Learning multiple layers of features from tiny images". In.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25.

LeCun, Yann, John Denker, and Sara Solla (1989). "Optimal brain damage". In: *Advances in neural information processing systems* 2.

Li, Hao, Hong Zhang, Xiaojuan Qi, Ruigang Yang, and Gao Huang (2019). "Improved techniques for training adaptive deep networks". In: *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1891–1900.

Lin, Ji, Yongming Rao, Jiwen Lu, and Jie Zhou (2017). "Runtime neural pruning". In: *Advances in neural information processing systems* 30.

List, Christian and Robert E Goodin (2001). "Epistemic democracy: Generalizing the Condorcet jury theorem". In.

Liu, Bo, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu (2021). "Conflict-averse gradient descent for multi-task learning". In: *Advances in Neural Information Processing Systems* 34, pp. 18878–18890.

Liu, Lanlan and Jia Deng (2018). "Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1.

Loaiza-Ganem, Gabriel, Brendan Leigh Ross, Jesse C Cresswell, and Anthony L Caterini (2022). "Diagnosing and fixing manifold overfitting in deep generative models". In: *arXiv preprint arXiv:2204.07172*.

Mangrulkar, Sourab, Ankith MS, and Vivek Sembium (2022). "Be3r: Bert based early-exit using expert routing". In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 3504–3512.

Marcolino, Leandro Soriano (2016). *Three fundamental pillars of decision-centered teamwork*. University of Southern California.

Marcolino, Leandro Soriano, Albert Xin Jiang, and Milind Tambe (2013). "Multi-agent team formation: Diversity beats strength?" In: *IJCAI*. Vol. 13.

Marcolino, Leandro Soriano, Haifeng Xu, Albert Xin Jiang, Milind Tambe, and Emma Bowring (2014). "Give a hard problem to a diverse team: Exploring large action spaces". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 28. 1.

Meronen, Lassi, Martin Trapp, Andrea Pilzer, Le Yang, and Arno Solin (2024). "Fixing overconfidence in dynamic neural networks". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 2680–2690.

Müller, Martin (2002). "Computer go". In: *Artificial Intelligence* 134.1-2, pp. 145–179.

Neal, Radford M (1993). "Probabilistic inference using Markov chain Monte Carlo methods". In.

Niu, Li, Yan Hong, Junyan Cao, and Liqing Zhang (2024). "Progressive painterly image harmonization from low-level styles to high-level styles". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 5, pp. 4352–4360.

Papamarkou, Theodore, Maria Skoularidou, Konstantina Palla, Laurence Aitchison, Julyan Arbel, David Dunson, Maurizio Filippone, Vincent Fortuin, Philipp Hennig, José Miguel Hernández-Lobato, et al. (2024). "Position: Bayesian Deep Learning is Needed in the Age of Large-Scale AI". In: *International Conference on Machine Learning*. PMLR, pp. 39556–39586.

Peng, Haoran and Leandro Soriano Marcolino (2025a). "Decision-centered Teamwork in Dynamic Early Exiting Networks". In: *In submission*.

— (2025b). "Dynamic Policy-Value Monte Carlo Tree Search". In: *In submission*.

Peng, Haoran, Leandro Soriano Marcolino, Bryan M. Williams, and Erickson R. Nascimento (2025c). "Diminishing Non-important Gradients for Training Dynamic Early-Exiting Networks". In: *In submission*.

Phuong, Mary and Christoph H Lampert (2019). "Distillation-based training for multi-exit architectures". In: *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1355–1364.

Prechelt, Lutz (2002). "Early stopping-but when?" In: *Neural Networks: Tricks of the trade*. Springer, pp. 55–69.

Sandler, Mark, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen (2018). "Mobilenetv2: Inverted residuals and linear bottlenecks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520.

Schuster, Tal, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler (2022). "Confident adaptive language modeling". In: *Advances in Neural Information Processing Systems* 35, pp. 17456–17472.

Shafer, Glenn and Vladimir Vovk (2008). "A tutorial on conformal prediction." In: *Journal of Machine Learning Research* 9.3.

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587, pp. 484–489.

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. (2018). "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419, pp. 1140–1144.

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. (2017). "Mastering the game of go without human knowledge". In: *nature* 550.7676, pp. 354–359.

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556*.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1, pp. 1929–1958.

Sun, Yi, Jian Li, and Xin Xu (2022). "Meta-GF: Training dynamic-depth neural networks harmoniously". In: *European Conference on Computer Vision*. Springer, pp. 691–708.

Świechowski, Maciej, Tomasz Tajmajer, and Andrzej Janusz (2018). "Improving hearthstone ai by combining mcts and supervised learning algorithms". In: *2018 IEEE conference on computational intelligence and games (CIG)*. IEEE, pp. 1–8.

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich (2015). "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

Tambe, Milind (1997). "Towards Flexible Teamwork". In: *Journal of Artificial Intelligence Research* 7, pp. 83–124.

Tang, Shengkun, Yaqing Wang, Zhenglun Kong, Tianchi Zhang, Yao Li, Caiwen Ding, Yanzhi Wang, Yi Liang, and Dongkuan Xu (2023). "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*, pp. 10781–10791.

Teerapittayanon, Surat, Bradley McDanel, and Hsiang-Tsung Kung (2016). "Branchynet: Fast inference via early exiting from deep neural networks". In: *2016 23rd international conference on pattern recognition (ICPR)*. IEEE, pp. 2464–2469.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention is all you need". In: *Advances in neural information processing systems* 30.

Veit, Andreas and Serge Belongie (2018). "Convolutional networks with adaptive inference graphs". In: *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–18.

Vovk, Vladimir, Alexander Gammerman, and Glenn Shafer (2005). *Algorithmic learning in a random world*. Vol. 29. Springer.

Wang, Xin, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez (2018). "Skipnet: Learning dynamic routing in convolutional networks". In: *Proceedings of the European conference on computer vision (ECCV)*, pp. 409–424.

Wang, Yulin, Rui Huang, Shiji Song, Zeyi Huang, and Gao Huang (2021). "Not all images are worth 16x16 words: Dynamic transformers for efficient image recognition". In: *Advances in neural information processing systems* 34, pp. 11960–11973.

Wei, Hongxin, Renchunzi Xie, Hao Cheng, Lei Feng, Bo An, and Yixuan Li (2022). "Mitigating neural network overconfidence with logit normalization". In: *International conference on machine learning*. PMLR, pp. 23631–23644.

Wołczyk, Maciej, Bartosz Wójcik, Klaudia Bałazy, Igor T Podolak, Jacek Tabor, Marek Śmieja, and Tomasz Trzcinski (2021). "Zero time waste: Recycling predictions in early exit neural networks". In: *Advances in Neural Information Processing Systems* 34, pp. 2516–2528.

Wu, David J (2019). "Accelerating self-play learning in go". In: *arXiv preprint arXiv:1902.10565*.

Xin, Derrick, Behrooz Ghorbani, Justin Gilmer, Ankush Garg, and Orhan Firat (2022). "Do current multi-task optimization methods in deep learning even help?" In: *Advances in neural information processing systems* 35, pp. 13597–13609.

Xin, Ji, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin (2020). "DeeBERT: Dynamic Early Exiting for Accelerating BERT Inference". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2246–2251.

Xin, Ji, Raphael Tang, Yaoliang Yu, and Jimmy Lin (2021). "BERxiT: Early exiting for BERT with better fine-tuning and extension to regression". In: *Proceedings of the 16th conference of the European chapter of the association for computational linguistics: Main Volume*, pp. 91–104.

Xu, Yangyang, Yibo Yang, and Lefei Zhang (2023). "Multi-task learning with knowledge distillation for dense prediction". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 21550–21559.

Yang, Bohong, Lin Wang, Hong Lu, and Youzhao Yang (2020a). "Learning the game of go by scalable network without prior knowledge of komi". In: *IEEE Transactions on Games* 12.2, pp. 187–198.

Yang, Le, Yizeng Han, Xi Chen, Shiji Song, Jifeng Dai, and Gao Huang (2020b). "Resolution adaptive networks for efficient inference". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2369–2378.

Yang, Le, Haojun Jiang, Ruojin Cai, Yulin Wang, Shiji Song, Gao Huang, and Qi Tian (2021). "Condensenet v2: Sparse feature reactivation for deep networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3569–3578.

Yang, Le, Ziwei Zheng, Jian Wang, Shiji Song, Gao Huang, and Fan Li (2023). "Adadet: An adaptive object detection system based on early-exit neural

networks". In: *IEEE Transactions on Cognitive and Developmental Systems* 16.1, pp. 332–345.

Yu, Jianglai, Lei Zhang, Dongzhou Cheng, Wenbo Huang, Hao Wu, and Aiguo Song (2024). "Improving Human Activity Recognition with Wearable Sensors through BEE: Leveraging Early Exit and Gradient Boosting". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering*.

Yu, Tianhe, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn (2020). "Gradient surgery for multi-task learning". In: *Advances in Neural Information Processing Systems* 33, pp. 5824–5836.

Yue, Yang, Yulin Wang, Bingyi Kang, Yizeng Han, Shenzhi Wang, Shiji Song, Jiashi Feng, and Gao Huang (2024). "DeeR-VLA: Dynamic Inference of Multimodal Large Language Models for Efficient Robot Execution". In: *arXiv preprint arXiv:2411.02359*.

Zhang, Xiangyu, Xinyu Zhou, Mengxiao Lin, and Jian Sun (2018). "Shufflenet: An extremely efficient convolutional neural network for mobile devices". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6848–6856.

Zhou, Wangchunshu, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei (2020). "Bert loses patience: Fast and robust inference with early exit". In: *Advances in Neural Information Processing Systems* 33, pp. 18330–18341.