Neural Networks 192 (2025) 107970

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Full Length Article

Strategies at a glance: A comparative analysis of training techniques for optimizing early-exit deep neural networks

Haseena Rahmath P *, Kuldeep Chaurasia, Abhay Bansal

School of Computer Science Engineering and Technology, Bennett University, Plot Nos 8–11, TechZone II, Greater Noida, UP, India

ARTICLE INFO

Keywords:
Early-exit DNN
Multi-exit DNN
Training
Dynamic inference
Inference acceleration

ABSTRACT

Early-exit deep neural networks (DNNs) enable adaptive inference by allowing predictions at intermediate layers, thereby reducing computational cost. However, their performance is highly sensitive to the chosen training strategy—a factor that remains underexplored. This study presents the first systematic comparison of six prominent strategies—Joint, Separate, Branch-wise, Two-stage, Distillation-based, and Hybrid—across three architectures (MobileNet, ResNet, VGG) using core benchmarks (CIFAR-10 and CIFAR-100). To evaluate scalability and domain generalization, we extended experiments on ImageNet-100 and ChestX-ray14. For each setup, we assess convergence behavior, accuracy, overfitting, and training efficiency, supported by statistical validation via ANOVA and Tukey's HSD tests. Results reveal key trade-offs: Joint and Distillation-based strategies offer strong generalization but incur higher computational cost; Two-stage and Branch-wise are prone to overfitting at deeper exits; Separate training underperforms at early exits. In contrast, Hybrid strategies achieve the best balance of accuracy and efficiency. These insights offer practical guidance for optimizing early-exit DNNs under resource constraints and lay a principled foundation for future research on efficient training paradigms.

1. Introduction

Deep neural networks (DNNs) have achieved state-of-the-art results in tasks such as image classification, object detection, and language modeling (He et al., 2016; Krizhevsky et al., 2012; Liu et al., 2024, 2019, 2022; Zheng et al., 2024). However, their high computational demands hinder deployment on mobile and edge devices operating under stringent power and latency constraints.

Early-exit DNNs address this challenge by attaching intermediate classifiers that allow predictions at various depths, enabling adaptive inference—where easy inputs exit early, while only complex ones proceed deeper, thereby saving computation and energy (Belilovsky et al., 2019; Teerapittayanon et al., 2016). This paradigm has gained traction in edge-AI and distributed systems, where dynamic resource allocation is essential (Karpikova et al., 2023; Laskaridis et al., 2020; Moon et al., 2023). While prior studies have explored architectural innovations (Huang et al., 2018; Mullapudi et al., 2018; Xu et al., 2023) and refined exit policies (Jo et al., 2023; Marquez et al., 2018; Wang et al., 2019, 2024), the training strategy—which governs how exits learn and coordinate—remains underexamined.

Training early-exit DNNs introduces unique challenges that are absent in conventional single-exit models. Since predictions can occur at multiple depths, the model must coordinate learning across exits with varying levels of feature maturity. Although several training strategies have been proposed (Gao et al., 2023; Kaya et al., 2019; Li et al., 2019b; Phuong & Lampert, 2019; Sabet et al., 2021; Teerapittayanon et al., 2016), their relative effectiveness remains unclear. To date, no large-scale study has systematically compared the trade-offs between accuracy, efficiency, and overfitting across training strategies, architectures, and datasets.

To address this gap, we investigate six prominent training strategies—Joint, Separate, Branch-wise, Two-stage, Distillation-based, and Hybrid—on three canonical backbones (MobileNet, ResNet, VGG) using CIFAR-10 and CIFAR-100. To test scalability and robustness, we extend our evaluation to ImageNet-100 and ChestX-ray14. Each strategy is assessed on convergence behavior, exit-wise accuracy, overfitting, and resource efficiency, with findings validated through ANOVA and Tukey's HSD analyzis.

Our results reveal clear trade-offs: Joint and Distillation-based strategies generalize well but demand high resources; Two-stage and Branch-wise tend to overfit at deeper exits; Separate weakens early-exit performance. In contrast, Hybrid strategies—particularly *Pretrained-branch* (backbone pretraining followed by branch-wise fine-tuning)—achieve the best trade-off between accuracy and efficiency.

* Corresponding author.
E-mail addresses: haseenarahmath@gmail.com (H. Rahmath P), kuldeep@bennett.edu.in (K. Chaurasia), abhay.bansal@bennett.edu.in (A. Bansal).

https://doi.org/10.1016/j.neunet.2025.107970
Received 12 April 2025; Received in revised form 23 June 2025; Accepted 4 August 2025
Available online 8 August 2025

These findings provide practical guidance for training early-exit DNNs under deployment constraints and deepen our understanding of optimization dynamics in adaptive inference architectures.

1.1. Contributions

Our key contributions are summarized below:

1. **First comprehensive benchmark.** We conduct the first large-scale, controlled comparison of six training strategies for early-exit DNNs—Joint, Separate, Branch-wise, Two-stage, Distillation-based, and Hybrid—across three architectures (MobileNet, ResNet, VGG) and two benchmarks (CIFAR-10/100), using ANOVA and Tukey's HSD to assess performance, convergence, and efficiency.
2. **Scalability and robustness analyzis.** To assess generalization under diverse conditions, we extend our evaluation to ImageNet-100 (large-scale natural images) and ChestX-ray14 (medical imaging). Results confirm that performance trends hold across both scale and domain shifts.
3. **Unified trade-off insights.** We analyze how model depth and dataset complexity affect the trade-offs between accuracy, overfitting, and training cost, offering a deeper understanding of optimization dynamics in early-exit DNNs.
4. **Impact of hybrid training.** We evaluate three Hybrid variants—*Joint-branch*, *Joint–two-stage*, and *Pretrained-branch*—and compare their performance against individual strategies such as Joint, Branch-wise, and Two-stage.
5. **Actionable guidelines.** We propose a concise decision matrix to help practitioners select appropriate training strategies under resource, latency, and performance constraints, enabling informed deployment of early-exit DNNs in real-world scenarios.

1.2. Research questions

To guide our investigation, we pose the following research questions:

- **RQ1:** How do different training strategies influence convergence behavior, exit-wise accuracy, and generalization in early-exit DNNs?
- **RQ2:** What are the computational efficiency and training stability characteristics of each strategy, particularly in terms of training time, memory, and power usage?
- **RQ3:** Can hybrid training approaches effectively mitigate overfitting at deeper exits while preserving robust early-exit performance?

The remainder of this paper is organized as follows. Section 2 surveys prior work on early-exit architectures and training strategies. Section 3 describes the baseline network designs, while Section 4 details the six training strategies benchmarked in this study. Section 5 outlines our experimental setup, and Section 6 presents the core quantitative results. Section 7 presents additional evaluations on large-scale and cross-domain datasets. We discuss key findings in Section 8, outline limitations and future directions in Section 9, and conclude in Section 10.

2. Related work

Early-exit DNNs have gained significant attention since the introduction of BranchyNet by Teerapittayanon et al. (2016), which added intermediate classifiers to LeNet, ResNet, and AlexNet to reduce inference time and computational cost. Subsequent studies confirmed the efficiency gains of early exits, particularly in resource-constrained environments (Pacheco et al., 2021a; Rahmath P et al., 2023). Graph-based extensions have also been explored (Rahmath P et al., 2025) and applied in specialized domains such as hyperspectral imaging (Ding et al., 2025a,b; Rahmath P & Chaurasia, 2024; Rahmath P et al., 2024a). However, these efforts primarily emphasize spectral–spatial representation learning rather than the design or comparison of early-exit training strategies.

Early exits have been deployed across various domains. In computer vision, they support image classification (Ilhan et al., 2024; Jo et al., 2023; Xu et al., 2023), object detection (Yang et al., 2024), image synthesis (Karpikova et al., 2023), video analytics (Wang et al., 2024), and content generation (Moon et al., 2023). In natural language processing, early-exit architectures improve performance on language understanding (He et al., 2024), question answering (Garg & Moschitti, 2021), sentiment analyzis (Ilhan et al., 2024), and text classification (Liu et al., 2020). Their efficiency in distributed systems (Laskaridis et al., 2020; Pacheco et al., 2021b) makes them appealing for edge–cloud AI deployments.

2.1. Architectural enhancements and exit policies

Research on early-exit networks spans a variety of labels—cascaded networks, multi-exit networks, layer-skipping models, and dynamic inference networks (Rahmath P et al., 2024b). To maintain consistency, we use the term *early-exit DNNs* throughout this paper.

Most prior work has focused on three main dimensions: First, architectural improvements, such as adding lightweight branches or multi-scale paths (Huang et al., 2018; Mullapudi et al., 2018; Teerapittayanon et al., 2016); second, exit-policy design for dynamic decision-making during inference (Bolukbasi et al., 2017; Figurnov et al., 2017; Li et al., 2017; Lin et al., 2017; Veit & Belongie, 2018; Wang et al., 2019, 2018; Wu et al., 2018; Ying & Fragkiadaki, 2018); and third, inference-time optimization, including adaptive early stopping (Bolukbasi et al., 2017). While these directions significantly reduce computational overhead, they often overlook the role of robust training strategies, which are essential for stabilising intermediate exits and preventing degradation at shallow depths—an aspect that remains underexplored and is the focus of this study.

2.2. Training strategies for early-exit DNNs

The training methodology plays a pivotal role in determining how well intermediate exits generalize and how efficiently computation is utilized. Table 1 categorizes prior work by training strategy. We briefly summarize each strategy here (with formal details in Section 4).

Joint training simultaneously optimizes the backbone and all exits using a weighted sum of exit-wise losses (Kaya et al., 2019; Lee et al., 2015; Leontiadis et al., 2021; Li et al., 2019a; Pacheco & Couto, 2020; Teerapittayanon et al., 2016; Wang et al., 2019, 2020). Separate training treats each exit as an independent model trained in isolation (Sabet et al., 2021; Wang et al., 2019). Branch-wise training incrementally trains exits alongside the backbone in sequence, freezing earlier layers once trained (Belilovsky et al., 2019; Wang et al., 2017a). Two-stage training first trains the full backbone and then fits each exit with the backbone weights frozen (Berestizshevsky & Even, 2019; Kaya et al., 2019). Distillation-based training leverages deeper exits or the full backbone as teacher models to guide earlier exits via knowledge distillation (Li et al., 2019b; Phuong & Lampert, 2019). Hybrid training combines elements of the above approaches to balance generalization, efficiency, and stability (Pacheco et al., 2021b; Wang et al., 2020).

As illustrated in Table 1, Joint and Two-stage training are the most widely adopted. In contrast, Branch-wise and Distillation-based approaches remain underexplored, and existing works typically evaluate a single strategy in isolation. Recent surveys on early-exit DNNs (Han et al., 2021; Laskaridis et al., 2021; Matsubara et al., 2022; Rahmath P et al., 2024b; Scardapane et al., 2020) provide valuable overviews of architectures and exit policies but devote limited attention to training methodology—and none conduct empirical benchmarking of multiple strategies side-by-side.

This gap motivates our systematic evaluation of six training strategies, including hybrid variants, across both standard (CIFAR-10/100) and domain-specific (ImageNet-100, ChestX-ray14) datasets, with the

Table 1
Early-exit training strategies used in prior work.

Strategy	Representative references
Joint	Berestizshevsky and Even (2019), Kaya et al. (2019), Laskaridis et al. (2020), Lee et al. (2015), Leontiadis et al. (2021), Li et al. (2019a), Mullapudi et al. (2018), Pacheco et al. (2021a), Pacheco and Couto (2020), Rahmath P et al. (2025, 2023), Teerapittayanon et al. (2016), Wang et al. (2019, 2020), Yang et al. (2024)
Two-stage	Berestizshevsky and Even (2019), Kaya et al. (2019), Leontiadis et al. (2021), Panda et al. (2016, 2017), Xu et al. (2023)
Separate	Bolukbasi et al. (2017), Lee et al. (2015), Wang et al. (2019, 2024)
Branch-wise	Belilovsky et al. (2019), Hettinger et al. (2017), Wang et al. (2017a)
Distillation-based	He et al. (2024), Li et al. (2019b), Liu et al. (2020), Phuong and Lampert (2019)
Hybrid	Brock et al. (2017), Ilhan et al. (2024), Liu et al. (2020), Pacheco et al. (2021b), Sabet et al. (2021), Wang et al. (2020), Xin et al. (2021)

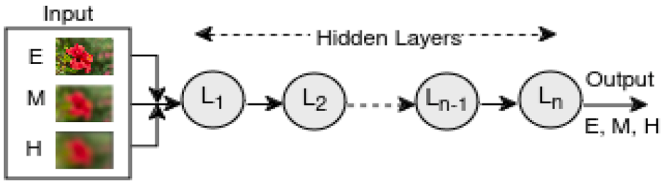


Fig. 1. A conventional DNN processes all inputs—regardless of difficulty—through the full stack of layers L_1 to L_n. This leads to inefficiencies for easy samples that could be confidently classified earlier.

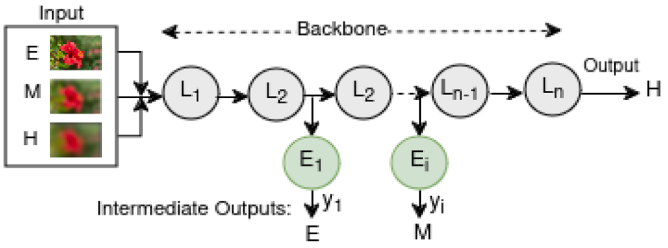


Fig. 2. An early-exit DNN inserts intermediate classifiers (exits) at multiple depths ($E_1 \ldots E_N$) along backbone layers ($L_1 \ldots L_n$). Each exit produces a partial prediction $(y_1, \ldots, y_N)$. Inputs of varying difficulty—easy (E), medium (M), hard (H)—depart at different depths, enabling dynamic inference.

goal of supporting principled early-exit deployment under real-world constraints.

3. Early-exit DNN architecture

Conventional deep neural networks (DNNs) operate as single-exit models, where each input traverses all layers sequentially before producing an output. As illustrated in Fig. 1, this one-size-fits-all computation results in unnecessary overhead—particularly for simpler inputs that could be classified earlier.

Early-exit DNNs address this inefficiency by inserting intermediate classifiers—referred to as exits—at multiple depths within the backbone. As shown in Fig. 2, each exit branch enables the model to make partial predictions, allowing easy inputs to exit early while more complex samples proceed deeper. This design supports adaptive inference, balancing accuracy and efficiency. A standard backbone—such as ResNet (He et al., 2016), MobileNet (Howard et al., 2017), VGG (Simonyan & Zisserman, 2015), or AlexNet (Krizhevsky et al., 2017)—is augmented with lightweight exit branches, denoted E_i. Each branch consists of a shallow classifier, typically composed of one or more convolutional or fully connected layers, followed by a softmax activation for prediction.

At inference time, each exit E_i produces a prediction y_i, and an exit policy determines whether to return it or continue to deeper layers. If the prediction confidence exceeds a threshold, the input exits early. Otherwise, computation proceeds until the final layer. While various static and dynamic exit policies exist, this paper focuses exclusively on the training strategies; exit policy design is deferred to related research.

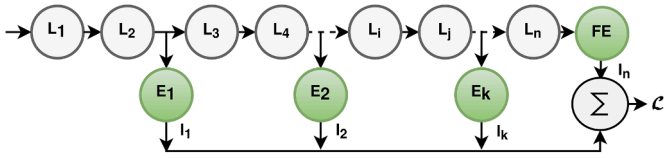


Fig. 3. Joint training strategy for early-exit DNNs. The backbone network (layers L_1 to L_n), including the final exit (FE), is trained jointly with all intermediate early-exit branches (E_1, E_2, …, E_i). Each exit produces a prediction loss (l_1, l_2, …, l_i, l_n), where l_n corresponds to the final exit. The total training loss $\mathcal{L}$ is computed as a weighted sum of these individual losses. This joint optimization encourages shared feature learning and simultaneous training across all exits.

4. Early-exit training strategies

Early-exit DNNs depart from conventional DNN designs by enabling predictions at multiple depths, making effective training essential to fully exploit their efficiency and performance potential. Over time, various training strategies have been developed to optimize these architectures—each addressing distinct challenges and trade-offs. As outlined in the Introduction, these strategies can be broadly categorized into six types: Joint training, Separate training, Branch-wise training, Two-stage training, Distillation-based training, and Hybrid training. This section provides a unified overview of these approaches, outlining their conceptual foundations, practical algorithms, and implementation workflows. A deeper understanding of these strategies equips researchers and practitioners to make informed design choices for deploying early-exit DNNs across diverse real-world applications.

4.1. Joint training strategy

The Joint training strategy simultaneously optimizes the backbone network and all early-exit branches, ensuring that the entire model is learned in a unified manner. This approach enhances performance across all exits by allowing each early-exit branch (denoted E_1, E_2, …, E_N) to contribute to overall accuracy and efficiency. The backbone layers L_1, …, L_n and the final exit (FE) are trained jointly with the intermediate branches to minimize a comprehensive loss function $\mathcal{L}$ that balances optimization across exits. The total loss is defined as:

$$\mathcal{L} = \sum_{n=1}^{N+1} W_n \, l_n, \tag{1}$$

where N is the number of early-exit branches, l_n is the loss at the n^{th} exit (with $n = N + 1$ corresponding to the final exit), and W_n is a weighting factor that controls the relative importance of each exit. Each branch computes a standard cross-entropy loss:

$$l_n = -\frac{1}{C} \sum_{c \in C} y_c \, \log\left(\frac{\exp(\hat{y}_{n,c})}{\sum_{c' \in C} \exp(\hat{y}_{n,c'})} \right), \tag{2}$$

where y_c is the ground-truth indicator for class c, and $\hat{y}_{n,c}$ is the predicted logit at exit n. The weighting factor W_n is computed as:

$$W_n = \frac{c\, n}{\sum_{k=1}^{N+1} k}, \tag{3}$$

where c is a scaling coefficient. By tuning W_n, the model can prioritize certain exits while maintaining global performance.

During training, a forward pass generates predictions at all exits. Their respective losses are aggregated using the weights above, and the combined loss is backpropagated through both the backbone and all exit heads. This simultaneous optimization enforces consistent feature learning across depths, improving early-exit accuracy without compromising final-layer performance. Algorithm 1 outlines the training process, Fig. 3 illustrates the architecture, and a detailed flowchart is provided in Appendix Fig. 21.

Joint training is widely adopted in early-exit DNNs (Kaya et al., 2019; Lee et al., 2015; Leontiadis et al., 2021; Li et al., 2019a; Pacheco & Couto, 2020; Teerapittayanon et al., 2016; Wang et al., 2019, 2020) and serves as a fundamental baseline. However, careful tuning of loss weights is essential—if later exits dominate, earlier branches may underfit. When balanced appropriately, joint training supports diverse input complexities and enhances both robustness and computational efficiency.

Algorithm 1 Joint training for early-exit DNNs.

Input: Early-exit model $\mathcal{M}$ with N exit branches (flagged as `exit_branch`); training dataset $(\mathbf{X}, \mathbf{Y})$; maximum number of epochs `MaxEpoch`
Output: Trained model with jointly optimized backbone and all exits

Initialize model parameters with ImageNet-pretrained weights
for $epoch = 1$ to `MaxEpoch` **do**
  for each minibatch (X_b, Y_b) in training data **do**
    $H \leftarrow X_b$ ▷ Forward pass input
    $\mathcal{L} \leftarrow 0$, $n \leftarrow 1$
    for each layer l in model $\mathcal{M}$ **do**
      $H \leftarrow l(H)$ ▷ Propagate through layer
      if l is an `exit_branch` **then** ▷ If current layer is an exit
        $\hat{Y}_n \leftarrow l(H)$ ▷ Predict at exit n
        $loss_n \leftarrow \texttt{CrossEntropy}(\hat{Y}_n, Y_b)$
        $w_n \leftarrow \frac{c \cdot n}{\sum_{k=1}^{N} k}$ ▷ Optional weighting (default $c = 1$)
        $\mathcal{L} \leftarrow \mathcal{L} + w_n \cdot loss_n$
        $n \leftarrow n + 1$
      end if
    end for
    Backpropagate and update all parameters using total loss $\mathcal{L}$
  end for
end for

4.2. Separate training strategy

The separate-training strategy optimizes each early-exit branch independently, treating it as a standalone classifier. Each exit, along with the corresponding subset of backbone layers feeding into it, is trained in isolation—tailoring its learning objective to its specific depth in the network. Unlike joint training, which updates all exits simultaneously, separate training avoids gradient interference across branches.

Fig. 4 illustrates the approach: the first sub-model contains layers L_1, L_2 and exit E_1, while the second includes L_1, L_2, L_3, L_4 and exit E_2. Algorithm 2 details the procedure, and a step-by-step flowchart is shown in Appendix Fig. 22.

Algorithm 2 Separate training for early-exit DNNs.

Input: Backbone $\mathcal{B}$ (ImageNet-pretrained); exit heads $\{E_1, \ldots, E_N\}$ at layers $\{L_1, \ldots, L_N\}$; training data $(\mathbf{X}, \mathbf{Y})$; maximum epochs `MaxEpoch`
Output: Fully trained early-exit model $\mathcal{M}$

 # Build one sub-model per exit
for $n = 1$ to N **do**
  $\mathcal{M}_n \leftarrow \mathcal{B}_{1:L_n} \cup E_n$ ▷ Truncate backbone to L_n and attach E_n
end for
 # Train exits independently
for $n = 1$ to N **do**
  for $epoch = 1$ to `MaxEpoch` **do**
    for each minibatch (X_b, Y_b) **do**
      $\hat{Y}_n \leftarrow \mathcal{M}_n(X_b)$ ▷ Forward pass
      $\mathcal{L} \leftarrow \texttt{CrossEntropy}(\hat{Y}_n, Y_b)$
      Update parameters of $\mathcal{M}_n$ using $\mathcal{L}$
    end for
  end for
end for
Assemble $\{\mathcal{M}_1, \ldots, \mathcal{M}_N\}$ into final model $\mathcal{M}$

Separate training offers flexibility—each exit can specialise in a distinct feature hierarchy, making it suitable for multi-task or domain-specific scenarios. By isolating optimization, later exits cannot interfere with earlier ones. The trade-off is that feature sharing is lost, which may lead to inconsistencies across exits and reduced generalization at deeper layers. Nevertheless, separate training remains effective in many early-exit systems (Bolukbasi et al., 2017; Lee et al., 2015; Wang et al., 2019, 2024), particularly when per-exit customisation is desirable.

4.3. Branch-wise training strategy

The branch-wise training strategy follows a progressive, layer-wise schedule in which early-exit branches are optimized sequentially, rather than updating the full network simultaneously. The model incrementally builds its feature representations: previously trained layers are frozen, while newly added layers support deeper abstraction. Training begins with the first exit (E_1), optimizing that branch and the backbone layers up to E_1. Once converged, those layers are frozen, and the next exit (E_2) is trained—updating only the newly introduced layers. This process continues until the final exit is trained, with each stage minimising a standard loss (e.g., cross-entropy).

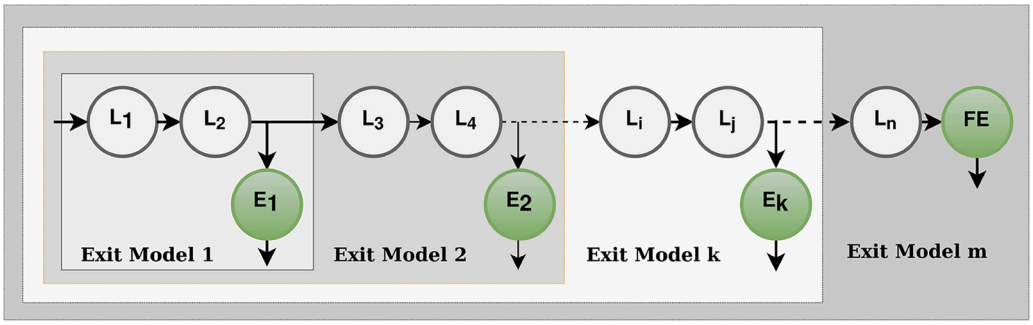


Fig. 4. Separate training strategy for early-exit DNNs. Each exit (E_1, E_2, …, E_N) is trained with its own sub-model comprising a prefix of backbone layers and the corresponding classifier head (e.g., L_1–L_2 + E_1, L_1–L_4 + E_2). The final exit (FE) is trained on the full backbone (L_1–L_n). No parameters are shared across exits during optimization, leading to independent learning in each branch.

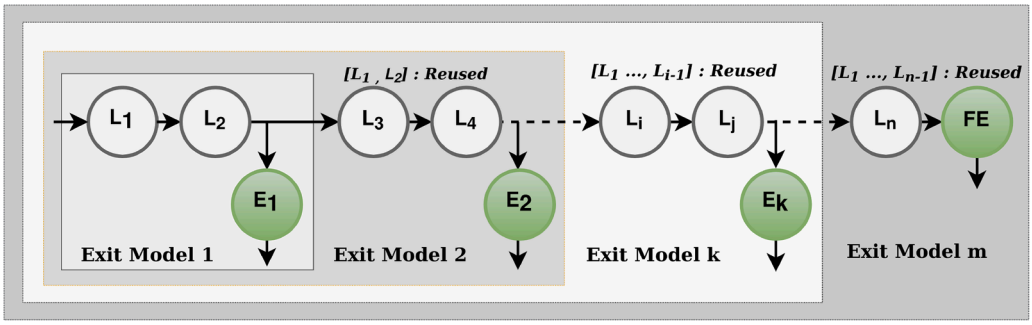


Fig. 5. Branch-wise training for early-exit DNNs. Training proceeds sequentially: layers L_1–L_2 and exit E_1 are optimized first and then frozen; subsequent layers (e.g. L_3, L_4) and exits (E_2, …, E_N) are added and trained in turn. Earlier representations are preserved throughout the process.

As illustrated in Fig. 5, optimization starts with layers L_1–L_2 and exit E_1; for exit E_2, the sub-model includes L_1–L_4 and E_2, where L_1–L_2 are kept frozen. Algorithm 3 presents the pseudocode, and a step-by-step flowchart is provided in Appendix Fig. 23.

This training schedule stabilises feature learning and reduces interference between shallow and deep exits. Compared to separate training—where each exit is optimized in complete isolation—branch-wise training reuses learned layers, saving computation and avoiding redundant updates. However, it has two key limitations: (i) frozen shallow layers cannot adapt to deeper exits, and (ii) errors in early exits may propagate forward. Still, branch-wise training proves effective in incremental learning and resource-constrained scenarios, where sequential updates are preferable to full-network optimization (Belilovsky et al., 2019; Hettinger et al., 2017; Wang et al., 2017a).

Algorithm 3 Branch-wise training for early-exit DNNs.

Input: Backbone $\mathcal{B}$ (ImageNet-pretrained); exits $\{E_1, \ldots, E_N\}$ at layers $\{L_1, \ldots, L_N\}$; training data $(\mathbf{X}, \mathbf{Y})$; maximum epochs `MaxEpoch`
Output: Trained early-exit model $\mathcal{M}$

 # Train one exit at a time, freezing earlier blocks
for $n = 1$ to N **do**
  $\mathcal{M}_n \leftarrow \mathcal{B}_{1:L_n} \cup E_n$ ▷ Truncate backbone, attach E_n
  if $n > 1$ **then**
    Initialize $\mathcal{M}_n$ with weights from $\mathcal{M}_{n-1}$
    Freeze layers up to L_{n-1} in $\mathcal{M}_n$
  end if
  for $epoch = 1$ to `MaxEpoch` **do**
    for each minibatch (X_b, Y_b) **do**
      $\hat{Y}_n \leftarrow \mathcal{M}_n(X_b)$
      $\mathcal{L} \leftarrow$ `CrossEntropy`$(\hat{Y}_n, Y_b)$
      Update *trainable* parameters of $\mathcal{M}_n$
    end for
  end for
  Save checkpoint of $\mathcal{M}_n$
end for
Merge $\{\mathcal{M}_1, \ldots, \mathcal{M}_N\}$ into final model $\mathcal{M}$

4.4. Two-stage training strategy

The two-stage training strategy divides optimization into two distinct phases, improving computational efficiency while preserving high final-exit accuracy. Unlike joint training, which updates all exits simultaneously, or separate training, which trains each exit from scratch, two-stage training *first* pre-trains the backbone and *then* fine-tunes the exit heads.

In Stage 1, the backbone is trained end-to-end—without any early exits—allowing it to learn robust feature representations. In Stage 2, the backbone weights are frozen, and each early-exit branch is optimized individually on the fixed features. This decoupled procedure eliminates redundant gradient computations and allows pretrained backbones to be reused. Fig. 6 illustrates the workflow, Algorithm 4 provides the corresponding pseudocode, and a detailed flowchart is included in Appendix Fig. 24.

Algorithm 4 Two-stage training for early-exit DNNs.

Input: Model $\mathcal{M}$ with N exits at layers $\{L_1, \ldots, L_N\}$; training data $(\mathbf{X}, \mathbf{Y})$; `MaxEpoch`
Output: Backbone and exits trained in two stages

Stage 1 - train backbone (final exit only)
for $epoch = 1$ to `MaxEpoch` **do**
  for each minibatch (X_b, Y_b) **do**
    $\hat{Y} \leftarrow \mathcal{M}(X_b)$ ▷ Forward to FE
    $\mathcal{L} \leftarrow$ `CrossEntropy`$(\hat{Y}, Y_b)$
    Back-propagate and update backbone parameters
  end for
end for
Freeze backbone weights
Stage 2 - train exits on frozen backbone
for $n = 1$ to N **do**
  $\mathcal{M}_n \leftarrow \mathcal{B}_{1:L_n} \cup E_n$
  for $epoch = 1$ to `MaxEpoch` **do**
    for each minibatch (X_b, Y_b) **do**
      $\hat{Y}_n \leftarrow \mathcal{M}_n(X_b)$
      $\mathcal{L} \leftarrow$ `CrossEntropy`$(\hat{Y}_n, Y_b)$
      Update parameters of E_n only
    end for
  end for
end for

Two-stage training substantially improves computational efficiency, as the backbone is optimized only once and its frozen features are reused across all exits. This reduces overall training time, making the approach attractive for large-scale applications or scenarios with limited training budgets. However, a key limitation is that the backbone is learned without explicitly accounting for intermediate exits. As a result, early-exit performance may lag behind that of joint or branch-wise training. The fixed feature maps may not align well with the needs of shallow classifiers, often resulting in higher loss at earlier exits.

Nevertheless, two-stage training remains an effective solution when final-exit accuracy is the primary goal. It is particularly useful for retrofitting pretrained networks—such as ResNet, MobileNet, or VGG—into early-exit architectures with minimal additional training. This strategy underpins several prior works (Berestizshevsky & Even, 2019; Kaya et al., 2019; Leontiadis et al., 2021; Panda et al., 2016, 2017).

4.5. Distillation-based training strategy

Distillation-based training transfers knowledge from a high-capacity teacher model to lightweight early-exit "students." Following initial backbone training in Stage 1, the teacher's soft predictions are used as additional supervision during Stage 2, where each exit is trained to align

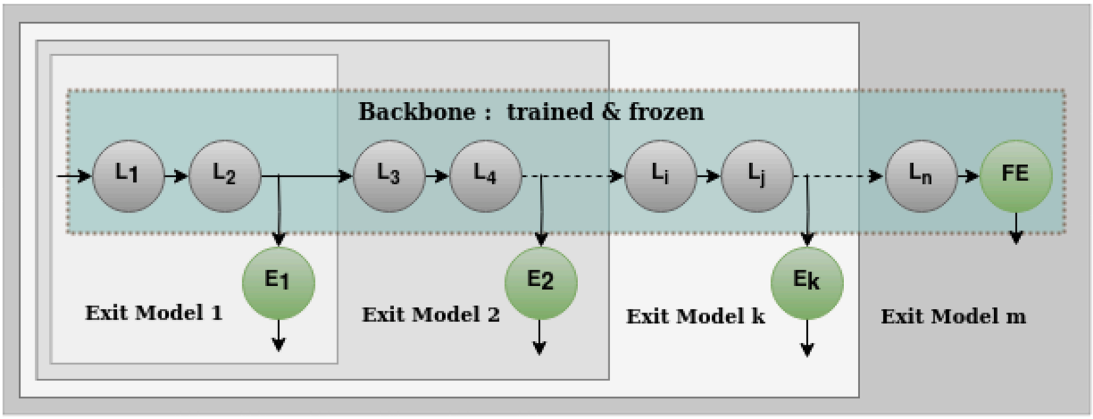


Fig. 6. Two-stage training for early-exit DNNs. Stage 1: the full backbone (layers L_1–L_n and the final exit, FE) is trained end-to-end. Stage 2: with backbone weights frozen, each early-exit branch (E_1, E_2, …, E_N) is trained independently—for instance, E_1 uses L_1–L_2, while deeper exits reuse progressively more layers. This modular workflow enables any pretrained backbone to be converted into an early-exit network with minimal retraining.

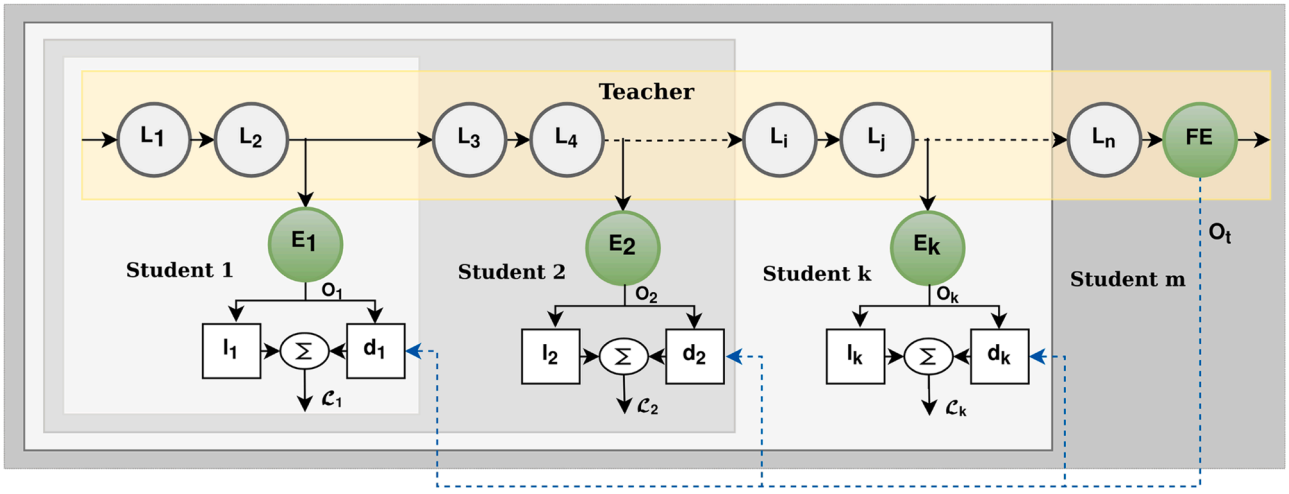


Fig. 7. Distillation-based training for early-exit DNNs. A pretrained teacher (layers L_1–L_n and final exit, FE) supplies soft targets $\mathcal{O}_t$ for each early exit (E_1, E_2, …, E_N). Each student output $\mathcal{O}_i$ is optimized using a weighted sum of cross-entropy loss l_i (ground truth) and distillation loss d_i (Kullback–Leibler divergence from the teacher).

with both the ground-truth labels and the teacher's softened output distribution.

The distillation loss is defined as:

$$\mathcal{L}_{\mathrm{D}} = \mathrm{KL}\left(\mathcal{O}_t \,\|\, \mathcal{O}_s\right) = \sum_i \mathcal{O}_t^i \log \frac{\mathcal{O}_t^i}{\mathcal{O}_s^i}, \tag{4}$$

where $\mathcal{O}_t$ and $\mathcal{O}_s$ are the softmax outputs of the teacher and student, respectively. Temperature scaling τ is applied to smooth the class probabilities. The total loss combines student supervision and distillation as follows:

$$\mathcal{L} = \alpha\, \mathcal{L}_{\mathrm{S}} + (1-\alpha)\, \mathcal{L}_{\mathrm{D}}, \tag{5}$$

where $\alpha \in [0, 1]$ controls the balance between direct supervision $\mathcal{L}_{\mathrm{S}}$ and knowledge transfer $\mathcal{L}_{\mathrm{D}}$. Smaller values of α place more emphasis on teacher guidance.

This method typically improves exit-wise accuracy and convergence compared to training exits independently, albeit with additional overhead from teacher pretraining. Nonetheless, distillation is among the most effective techniques for boosting early-exit performance, as demonstrated in several studies (Li et al., 2019b; Phuong & Lampert, 2019). Fig. 7 visualises the training procedure. Algorithm 5 provides the corresponding pseudocode, and a step-by-step flowchart is included in Appendix Fig. 25.

4.6. Hybrid training strategy

Hybrid training combines multiple optimization schemes within a single early-exit network to exploit their complementary strengths. A typical pipeline begins with a global training routine—most often joint training—followed by a local refinement phase using branch-wise fine-tuning, separate re-training, or distillation at individual exits. This layered schedule can improve exit-specific accuracy, albeit at the cost of additional training phases and hyperparameter tuning.

Variants of hybrid training are common in the literature. Freeze-Out (Brock et al., 2017) augments joint training with a branch-wise refinement stage; FastBERT (Liu et al., 2020) integrates separate training and distillation for language models; and Pacheco et al. (2021b) employ a joint–then–two-stage schedule for edge–cloud inference. Such pipelines enhance generalization at early exits while preserving final-exit performance. Pseudocode for the *Joint–branch* Hybrid variant is shown in Algorithm 6.

By pairing global optimization with exit-specific refinement, hybrid training typically achieves stronger overall performance than any single strategy. The trade-off is increased compute cost, sensitivity to hyperparameters, and added implementation complexity. Nevertheless, when robust early-exit accuracy is essential, hybrid training provides a flexible and effective solution.

5. Experiments

This section outlines the experimental setup for evaluating early-exit DNNs trained under different strategies.

5.1. Model architectures

We implemented early-exit variants of three widely used CNN backbones: MobileNet (Howard et al., 2017), ResNet (He et al., 2016), and VGG (Simonyan & Zisserman, 2015). These architectures were selected for their complementary properties—MobileNet for lightweight efficiency, ResNet for residual learning, and VGG for hierarchical depth. Their early-exit counterparts—E-MobileNet, E-ResNet, and E-VGG—include three strategically positioned exits to support adaptive inference: enabling early predictions for simple inputs while allowing deeper refinement for harder cases. Specifically, E-MobileNet includes exits at the 8th, 14th, and 24th layers; E-ResNet (based on ResNet-50) places exits at the 10th, 22nd, and 40th layers; and E-VGG (based on VGG-16) inserts them at the 5th, 9th, and 13th layers. All backbones are initialized with ImageNet-pretrained weights.

Each exit branch is designed to provide efficient and stable early predictions. The architecture begins with a MaxPooling2D layer for spatial downsampling, followed by a Conv2D layer with 128 filters (reduced to 64 at the first exit in E-MobileNet), Batch Normalisation, and ReLU activation. Earlier exits include a second MaxPooling2D layer, whereas deeper exits retain full spatial resolution. Outputs are flattened or globally pooled (GlobalAveragePooling2D for E-MobileNet), then passed through a 128-unit Dense layer, Dropout, and Batch Normalisation. A final softmax layer produces class probabilities.

Algorithm 5 Distillation-based training for early-exit DNNs.

Input: Early-exit model $\mathcal{M}$ with N exits; training data $(\mathbf{X}, \mathbf{Y})$; temperature τ; balance weight α; `MaxEpoch`
Output: Trained model $\mathcal{M}$

Stage 1 - train teacher (final exit)
for $epoch = 1$ to `MaxEpoch` **do**
  for each minibatch (X_b, Y_b) **do**
    $\hat{Y}_T \leftarrow \mathcal{M}_T(X_b)$
    $\mathcal{L}_T \leftarrow \texttt{CrossEntropy}(\hat{Y}_T, Y_b)$
    Back-propagate; update all parameters
  end for
end for
Stage 2 - distil each early exit
for $n = 1$ to N **do**
  $\mathcal{M}_n \leftarrow \mathcal{B}_{1:L_n} \cup E_n$
  for $epoch = 1$ to `MaxEpoch` **do**
    for each minibatch (X_b, Y_b) **do**
      $\hat{Y}_S \leftarrow \mathcal{M}_n(X_b)$
      $\hat{Y}_T^{\tau} \leftarrow \texttt{softmax}(\mathcal{M}_T(X_b)/\tau)$
      $\hat{Y}_S^{\tau} \leftarrow \texttt{softmax}(\hat{Y}_S/\tau)$
      $\mathcal{L}_S \leftarrow \texttt{CrossEntropy}(\hat{Y}_S, Y_b)$
      $\mathcal{L}_D \leftarrow \tau^2\, \texttt{KL}(\hat{Y}_T^{\tau} \,\|\, \hat{Y}_S^{\tau})$
      $\mathcal{L} \leftarrow \alpha\, \mathcal{L}_S + (1-\alpha)\, \mathcal{L}_D$
      Update parameters of $\mathcal{M}_n$ w.r.t. $\mathcal{L}$
    end for
  end for
end for

Algorithm 6 Joint–branch hybrid training for early-exit DNNs.

Input: Early-exit model $\mathcal{M}$ with N exits; training set $(\mathbf{X}, \mathbf{Y})$; epochs $\texttt{MaxEpoch}_1$ (phase 1) and $\texttt{MaxEpoch}_2$ (phase 2)
Output: Backbone jointly optimized; exits refined branch-wise

Phase 1 - joint optimization
for $epoch = 1$ to $\texttt{MaxEpoch}_1$ **do**
  for each minibatch (X_b, Y_b) **do**
    $\mathcal{L} \leftarrow 0, n \leftarrow 1$
    for each layer l in $\mathcal{M}$ **do**
      $H \leftarrow l(H)$
      if l is an `exit_branch` **then**
        $\hat{Y}_n \leftarrow l(H)$; $loss_n \leftarrow \texttt{CrossEntropy}(\hat{Y}_n, Y_b)$
        $w_n \leftarrow \frac{c\,n}{\sum_{k=1}^{N} k}$; $\mathcal{L} \leftarrow \mathcal{L} + w_n\, loss_n$
        $n \leftarrow n + 1$
      end if
    end for
    Back-propagate; update all parameters w.r.t. $\mathcal{L}$
  end for
end for
Phase 2 - branch-wise refinement
for $n = 1$ to N **do**
  Freeze backbone layers up to L_n; define sub-model $\mathcal{M}_n = \mathcal{B}_{1:L_n} \cup E_n$
  for $epoch = 1$ to $\texttt{MaxEpoch}_2$ **do**
    for each minibatch (X_b, Y_b) **do**
      $\hat{Y}_n \leftarrow \mathcal{M}_n(X_b)$
      $\mathcal{L}_n \leftarrow \texttt{CrossEntropy}(\hat{Y}_n, Y_b)$
      Update *only* the parameters of E_n
    end for
  end for
end for

5.2. Datasets

Our evaluation spans *two core benchmarks*—CIFAR-10 and CIFAR-100—and *two targeted extensions* that probe scalability and domain transfer: ImageNet-100 and ChestX-ray14.

CIFAR-10 / CIFAR-100. Each dataset comprises 60,000 color images at 32×32 resolution; CIFAR-10 includes 10 object classes, while CIFAR-100 covers 100 fine-grained categories (Krizhevsky et al., 2009). This pair enables evaluation of generalization from coarse to fine labels under consistent image statistics.

ImageNet-100. To evaluate scalability to high-resolution, large-scale natural images, we adopt ImageNet-100—a 100-class subset of ILSVRC-2012 widely used in contrastive learning research (Chun-Hsiao Yeh, 2022; Tian et al., 2020). It contains approximately 130,000 training and 5000 validation images at 224×224 resolution. MobileNet-V2 is assessed on this benchmark to measure performance under constrained computational budgets.

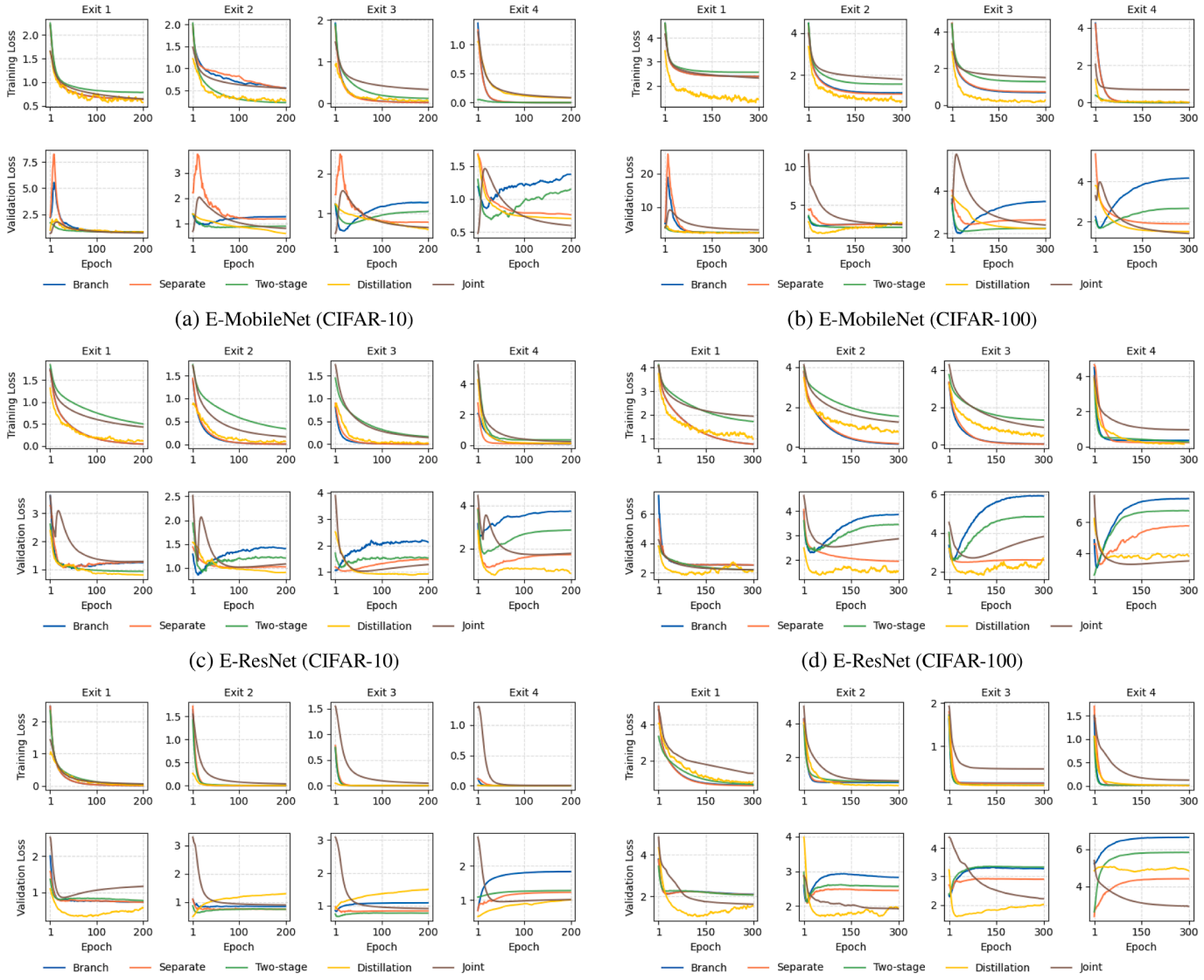
ChestX-ray14. To evaluate domain transferability and robustness in medical imaging, we include the ChestX-ray14 dataset (Wang et al., 2017b). We simplify the task to binary classification (pneumonia vs. normal) and adopt an 80/20 train–test split. All images are resized to 224×224 and channel-wise normalised. This setup emulates real-world clinical inference under resource limitations.

In summary, CIFAR-10 and CIFAR-100 assess model behavior on low-resolution natural images, while ImageNet-100 and ChestX-ray14 test scalability to large-scale vision and generalization to out-of-distribution (medical) domains.

5.3. Experimental setup

All strategies share a unified configuration. Optimizer settings mirror best-practice values reported in the backbone and early-exit literature: E-ResNet and E-VGG use SGD (momentum 0.9) with initial learning rates 0.1 and 0.01, respectively (He et al., 2016; Kaya et al., 2019; Simonyan & Zisserman, 2015); E-MobileNet adopts Adam at 0.001 following Howard et al. (2017).

Epoch budgets are 200 for CIFAR-10, 300 for CIFAR-100, and 50 for ChestX-ray14 (early-stopping patience = 10). For ImageNet-100, we follow Tian et al. (2020): Mobile-Net-V2 is trained for 100 epochs with SGD (momentum 0.9, weight-decay 10^{-4}), an initial learning rate of 0.05 (scaled linearly to batch size 64), and cosine decay. Images are resized to



(a) E-MobileNet (CIFAR-10)

(b) E-MobileNet (CIFAR-100)

(c) E-ResNet (CIFAR-10)

(d) E-ResNet (CIFAR-100)

(e) E-VGG (CIFAR-10)

(f) E-VGG (CIFAR-100)

Fig. 8. Exit-wise training and validation loss curves for E-MobileNet, E-ResNet, and E-VGG on CIFAR-10 and CIFAR-100. Each subplot shows how loss varies across exits during training, enabling comparison of convergence behavior and stability across architectures and datasets.

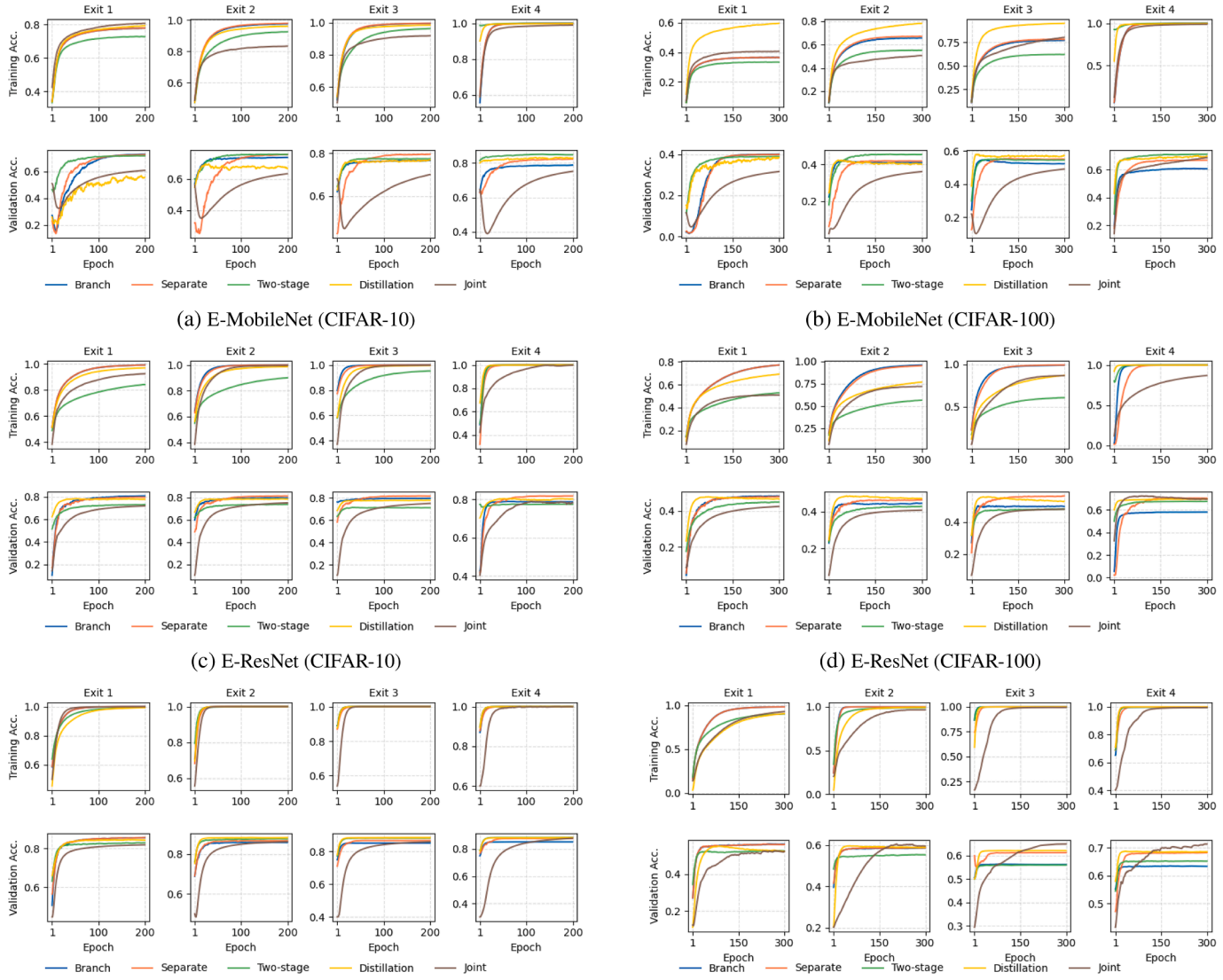
256×256; training augmentation is random–resized– crop to 224×224 plus horizontal flip, and validation uses a 224×224 centre crop.

All experiments run on an NVIDIA Tesla V100 (16 GB) GPU and an Intel Xeon E5-2698 CPU. CIFAR data are loaded via `tensorflow.keras.datasets`; ImageNet-100 and ChestX-ray14 are read from JPEG files with identical channel normalisation. Batch size is 128 for CIFAR and ChestX-ray14, and 64 for ImageNet-100. We fix `seed` as 42 and enable deterministic behavior in TensorFlow 2.14; all weights are ImageNet-initialized unless noted otherwise. No augmentation beyond mean–std normalisation is applied to CIFAR or ChestX-ray14. GPU power and memory are sampled at 1 Hz with `nvidia-smi/pynvml`, while CPU package power and memory are fetched via Intel RAPL and `psutil`. All reported figures are the mean over each complete training run (five repetitions).

5.4. Training methodologies

We evaluate six early-exit training strategies:

- **Joint training.** The backbone and all exits are optimized simultaneously using a weighted cross-entropy loss. Loss weights decrease linearly from 1.0 at the final exit to 0.3 at the earliest exit to encourage balanced supervision.
- **Separate training.** Each exit is treated as an independent classifier, trained with its own optimizer and learning schedule. No weights are shared across exits, and backbone parameters are isolated per branch.
- **Branch-wise training.** Exit branches are optimized sequentially. Once the first exit converges, its associated backbone layers are frozen; each subsequent exit is then trained on the partially frozen network, avoiding interference with earlier exits.
- **Two-stage training.** The backbone is first trained to convergence without any exits. Its weights are then frozen, and each exit is independently trained on these fixed feature representations.
- **Distillation-based training.** A teacher model (the trained backbone) is used to supervise exit branches. Each student exit is trained with a weighted combination of cross-entropy loss and KL-divergence from the teacher's softened outputs. Temperature scaling



(a) E-MobileNet (CIFAR-10)

(b) E-MobileNet (CIFAR-100)

(c) E-ResNet (CIFAR-10)

(d) E-ResNet (CIFAR-100)

(e) E-VGG (CIFAR-10)

(f) E-VGG (CIFAR-100)

Fig. 9. Exit-wise training and validation accuracy curves for E-MobileNet, E-ResNet, and E-VGG on CIFAR-10 and CIFAR-100. Each plot shows how accuracy evolves at different exits, helping assess generalization and exit-wise overfitting behavior across models and datasets.

($T = 3$) and a balancing factor ($\alpha = 0.1$) modulate the trade-off between hard labels and soft supervision.

- **Hybrid training.** We evaluate three hybrid schemes that combine global and local optimization phases while maintaining low implementation overhead:

(1) *Joint-branch*—joint training followed by branch-wise fine-tuning (Brock et al., 2017);

(2) *Joint–two-stage*–joint training followed by freezing the backbone and training exits independently (Barinov et al., 2023; Pacheco et al., 2021a);

(3) *Pretrained-branch*—a pretrained backbone is reused and exit branches are optimized using the branch-wise schedule.

The first two variants are drawn from prior literature, where they have demonstrated improved performance across both vision and edge inference tasks. We additionally introduce the *Pretrained-branch* variant for its practicality, minimal tuning overhead, and relevance to deployment scenarios where backbone weights are fixed—such as under regulatory, hardware, or training-cost constraints. These hybrids were prioritised over curriculum-based or meta-learning variants due to their simplicity, generality, compatibility with off-the-shelf pretrained backbones, and low implementation complexity.

Collectively, they span a representative spectrum of coordination strategies without requiring domain-specific scheduling or policy design.

5.5. Evaluation metrics

We assess early-exit training strategies using three categories of metrics:

- **Performance Evaluation:** Cross-entropy loss and classification accuracy are measured on training and validation sets. Key indicators include convergence speed, exit-wise accuracy, and training stability across epochs.
- **Training Dynamics:** Total and per-exit training time are tracked, alongside epoch-wise durations, to assess computational efficiency. Convergence is defined in terms of the epoch achieving minimum validation loss, peak accuracy, and accuracy stabilisation.

- **Resource Utilization:** GPU/CPU memory usage and power consumption are monitored per exit. For Separate, Branch-wise, Two-stage, and Distillation-based strategies, memory is aggregated across exit runs, while peak power usage is reported. For Joint training, resources are recorded directly during unified training. To reduce variance, all metrics are averaged across training epochs and aggregated over five independent runs.

5.6. Statistical analysis

We employed one-way ANOVA and Tukey's HSD post-hoc tests to assess the statistical significance of differences across training strategies. ANOVA was performed independently for each metric—accuracy, loss, training time, memory usage, and power consumption—across all models and datasets. A result of $p < 0.05$ was considered statistically significant, indicating that at least one strategy differed meaningfully from the others for that metric.

To identify specific pairwise differences, we applied Tukey's HSD post-hoc test, which evaluates which strategy pairs differ significantly ($p < 0.05$). Results are visualised using directed graphs, where an arrow from Strategy A → Strategy B indicates that A significantly outperforms B on the corresponding metric. This statistical framework provides robust validation of observed trends and enables nuanced comparisons of generalization, efficiency, and training stability. It directly supports the research questions posed in Section 1.2 and enhances the interpretability of the experimental findings.

Assumption Validation: Prior to conducting ANOVA, we verified all necessary assumptions across model–dataset– metric–strategy combinations. Independence was ensured by executing five separate trials per strategy using distinct random seeds. Normality of residuals was tested using the Shapiro–Wilk test, and the assumption was satisfied ($p > 0.05$) in the majority of cases (e.g., Joint: $p = 0.8794$; Distillation-based: $p = 0.6980$). A few isolated violations—primarily involving the Branch strategy—were observed (e.g., $p = 0.01$ for E-MobileNet–CIFAR10 training time). Given that ANOVA is robust to mild deviations from normality when group sizes are equal, these cases were not considered to invalidate the analyzis. Homogeneity of variances was assessed using Levene's test, which confirmed variance equality across all tested configurations (e.g., $p = 0.7462$ for E-MobileNet–CIFAR10), thereby justifying the application of the ANOVA framework.

6. Results

We present a comprehensive evaluation of early-exit training strategies across multiple architectures and datasets. Our analyzis addresses three key aspects: (1) convergence behavior, exit-wise accuracy, and generalization; (2) computational efficiency and training stability; and (3) the effectiveness of hybrid training in mitigating overfitting. All findings are statistically validated using one-way ANOVA and Tukey's HSD.

6.1. How do different training strategies influence convergence behavior, exit-wise accuracy, and generalization in early-exit DNNs? (RQ1)

We begin by analyzing the effects of training strategies on loss convergence, accuracy, and generalization across all model–dataset combinations.

6.1.1. Loss convergence

Fig. 8 presents exit-wise training and validation loss curves. Fig. 10a, b, e, f, i, and j illustrate the overall convergence trends across training strategies for each model and dataset.

Convergence speed. Distillation-based training consistently achieves the fastest convergence, typically within 50 epochs, followed by Joint training (approximately 75 epochs). Separate training shows fast early convergence, especially in E-MobileNet and E-VGG, though it fluctuates at later exits in E-ResNet. Two-stage training converges more gradually across models and is especially stable on CIFAR-100. Branch-wise training is the slowest and least stable, with late convergence and signs of overfitting in deeper exits.

Exit-wise performance. At Exits 1 and 2, Distillation-based and Joint training achieve the lowest training and validation loss across all models and datasets. Separate training performs reasonably well, occasionally matching Two-stage in early exits. Branch-wise consistently yields higher loss in these stages, particularly in E-ResNet and E-VGG. At Exit 3, Distillation-based and Joint continue to lead. Two-stage training typically outperforms Separate and Branch-wise, especially on CIFAR-100, where it maintains lower loss and better stability. Branch-wise again shows pronounced spikes in E-ResNet (CIFAR-100), while Separate begins to diverge more strongly at this depth. At the final exit (Exit 4), Distillation-based training maintains the lowest overall loss, followed by Joint and Two-stage. Separate performs moderately well but exhibits greater instability in E-ResNet and E-VGG on CIFAR-100. Branch-wise records the highest final-exit loss across most configurations.

Validation loss and stability. Distillation-based and Joint training show the most stable validation loss across exits and datasets, with minimal divergence from training loss. Two-stage training is also notably stable—particularly in later exits—exhibiting lower volatility than Separate or Branch-wise, especially on CIFAR-100. Separate training stabilises in shallower exits but shows degradation at deeper exits in E-ResNet. Branch-wise training displays the most pronounced instability and overfitting, with severe validation spikes at Exit 3 and Exit 4 in both E-MobileNet and E-ResNet (CIFAR-100).

6.1.2. Accuracy trends

Fig. 9 presents exit-wise training and validation accuracy curves, while Figs. 10c, d, g, h, k, and l illustrate overall accuracy convergence across models and datasets.

Convergence speed and overall accuracy. Distillation-based training achieves the highest and fastest accuracy gains across all exits and models. Joint training follows closely, with smooth improvements and minimal overfitting. Separate training performs competitively in both E-MobileNet and E-VGG, consistently outperforming Two-stage and Branch-wise, especially in CIFAR-100. Branch-wise training shows the slowest convergence and lowest final accuracy across datasets. Two-stage training is more stable than Branch-wise, but underperforms Separate in most cases—particularly in deeper models and complex tasks like CIFAR-100.

Exit-wise accuracy trends. At Exits 1 and 2, Distillation-based and Joint training deliver the highest accuracy, enabling reliable early exits. Separate training is close behind and often surpasses Two-stage at these points. Branch-wise consistently lags, especially in deeper networks. At Exit 3, Separate continues to outperform Two-stage in most models (E-MobileNet, E-VGG), with Two-stage slightly recovering only in E-ResNet. Branch-wise continues to degrade and performs worst across all architectures. At the final exit (Exit 4), Distillation and Joint still lead. Separate training remains robust and generally outperforms Two-stage and Branch-wise, particularly in E-VGG and E-MobileNet. Two-stage occasionally narrows the gap but remains slightly weaker. Branch-wise records the lowest final-exit accuracy in all configurations.

Validation accuracy and generalization. Distillation-based and Joint training exhibit the strongest generalization, with consistently aligned training and validation curves across all models and exits. Separate training shows some variability at early exits but stabilises effectively by Exit 3; however, it exhibits instability at later exits in deeper models like E-ResNet on CIFAR-100. Two-stage training maintains stable training behavior overall but suffers slightly reduced accuracy in deeper exits,

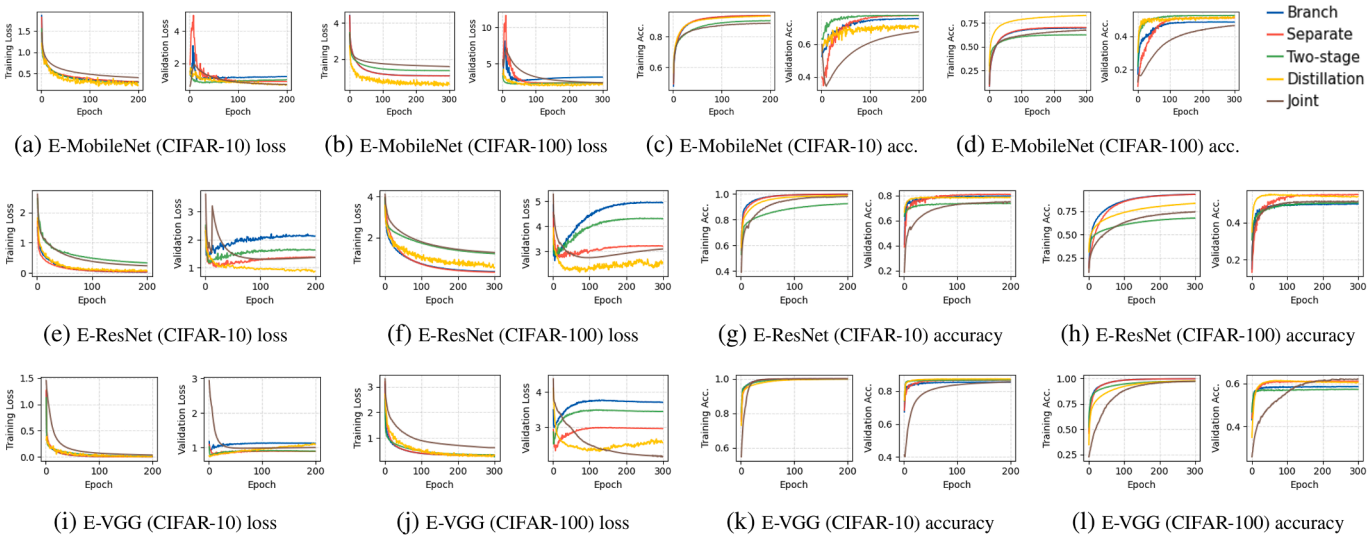


Fig. 10. Overall training loss and classification accuracy trends for E-MobileNet, E-ResNet, and E-VGG on CIFAR-10 and CIFAR-100. The plots summarize model-wide performance progression, capturing how effectively each architecture converges during training across datasets.

Table 2
ANOVA F-statistics and p-values for training and validation loss/accuracy across strategies. Significant differences ($p < 0.05$) indicate meaningful variation.

Model (Dataset)	Loss		Accuracy	
	Train F/p	Val F/p	Train F/p	Val F/p
E-MobileNet (CIFAR-10)	824.24 / 6.9e−22	3562.42 / 3.2e−28	130.30 / 5.0e−14	839.66 / 5.8e−22
E-MobileNet (CIFAR-100)	36332.91 / 2.7e−38	34305.20 / 4.8e−38	1536.00 / 1.4e−24	417.14 / 5.9e−19
E-ResNet (CIFAR-10)	5482.75 / 4.3e−30	25434.05 / 9.5e−37	240.20 / 1.3e−16	341.46 / 4.2e−18
E-ResNet (CIFAR-100)	40268.30 / 9.6e−39	122183.05 / 1.4e−43	2082.46 / 6.8e−26	141.04 / 2.4e−14
E-VGG (CIFAR-10)	347.92 / 3.5e−18	1525.94 / 1.5e−24	4.20 / 1.2e−2	79.62 / 5.4e−12
E-VGG (CIFAR-100)	11231.30 / 3.3e−33	46075.22 / 2.5e−39	626.72 / 1.0e−20	72.21 / 1.3e−11

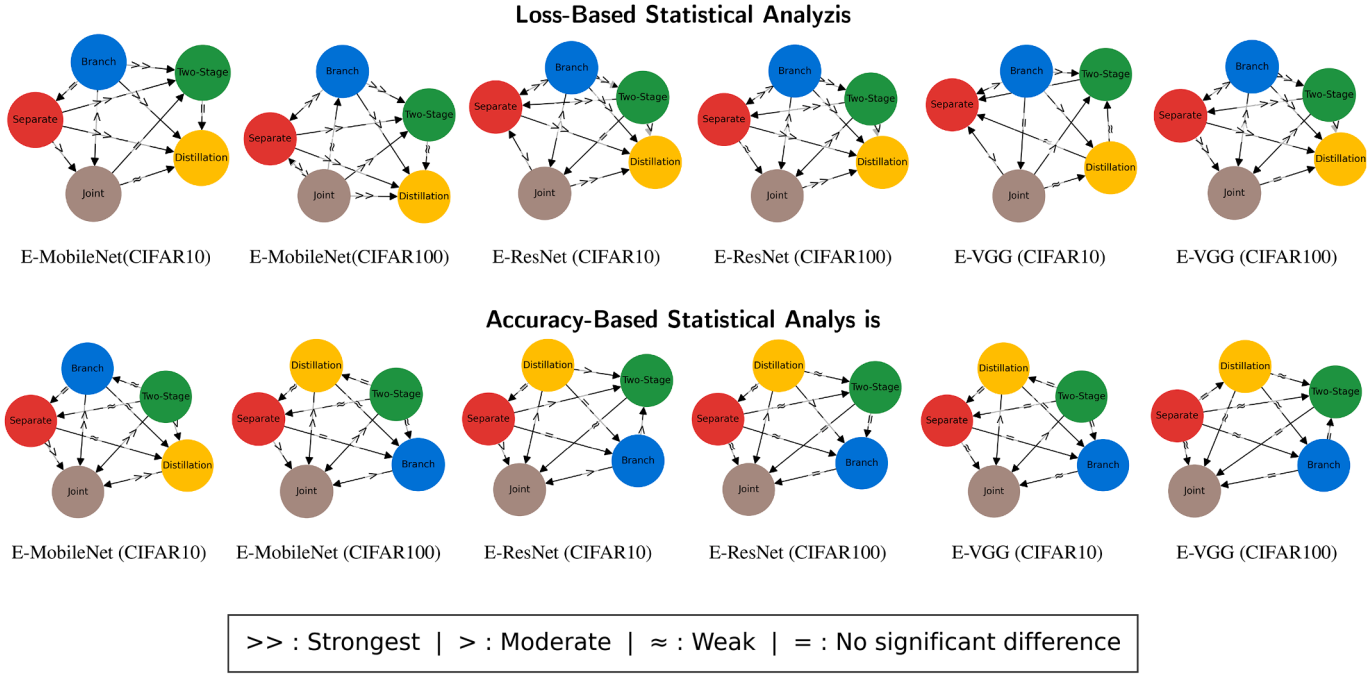


Fig. 11. Directed graph visualization of pairwise statistical comparisons using Tukey's HSD test across different training strategies. The top row depicts validation loss and the bottom row shows validation accuracy. Each node represents a training strategy. A solid arrow from Strategy A to Strategy B indicates that A has significantly higher loss (top) or higher accuracy (bottom) than B ($p < 0.05$). Edge labels denote the mean difference, as explained in the legend.

Table 3
Exit-wise training trends and strategy behaviors across models/datasets. Abbreviations: D = Distillation-based, J = Joint, T = Two-stage, S = Separate, B = Branch-wise.

Metric	Exit-wise trend	Strategy observations
Min Loss Epoch	Increases with exit depth; CIFAR-10 stabilizes faster than CIFAR-100.	Best: D, followed by J. S stabilizes reasonably; T and B struggle, particularly on E-VGG and E-ResNet (CIFAR-100).
Peak Accuracy Epoch	Occurs later at deeper exits; slower for E-VGG and E-ResNet on CIFAR-100.	Fastest: D > J. T converges moderately and smoothly. S is slower but consistent. B is slowest and highly erratic.
Overfitting Epoch	Later exits overfit more, especially in CIFAR-100.	Most overfitting: B > T > S. Best resistance: D and J.
Saturation Epoch	Early exits stabilize quickly; later exits vary more in deep models.	Smooth convergence: D, J, T. S fluctuates moderately. B remains unstable at deeper exits.
Generalization Gap	Small in CIFAR-10; widens significantly at deeper exits in CIFAR-100.	Narrowest: D. Moderate: J and T. Widest: S and B.
Validation Stability	Degrades at deeper exits; worst in E-ResNet and E-VGG (CIFAR-100).	Most stable: D and J. T remains fairly stable post-freeze. S is moderately stable; B is highly unstable.
Final Exit Accuracy	Increases with depth; lower in complex models/datasets.	Highest: D > J > T. S is competitive on CIFAR-10. Lowest: B on CIFAR-100.

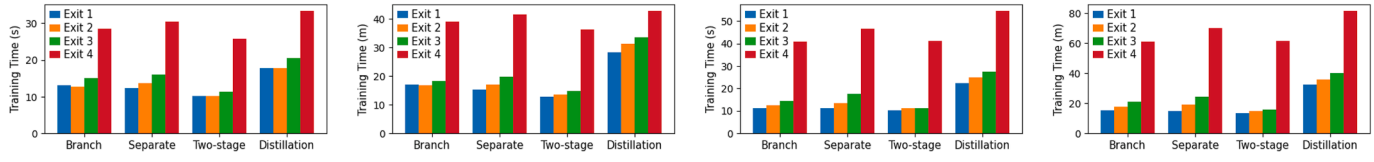


Fig. 12. Exit-wise training time for E-MobileNet and E-ResNet on CIFAR-10 and CIFAR-100. Each group of bars corresponds to a training strategy, and within each group, individual bars are color-coded by exit branch (e.g., E_1, E_2, …), as shown in the legend. The figure highlights how training time accumulates across exits and varies depending on the model, dataset, and training strategy.

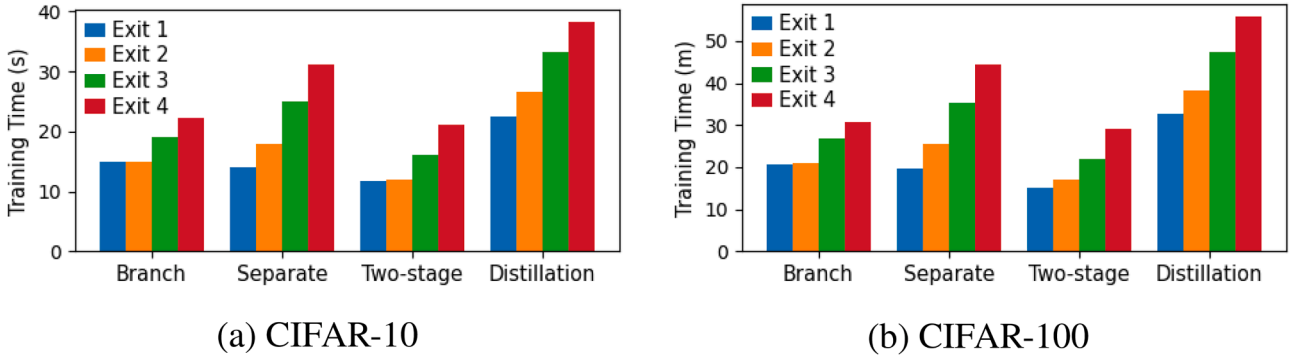


Fig. 13. Exit-wise training time comparison for E-VGG on CIFAR-10 and CIFAR-100. Each bar represents the training time for a specific exit under different training strategies. Blue and red colors indicate results for CIFAR-10 and CIFAR-100, respectively. Training strategies are shown along the x-axis, with grouped bars representing different exits.

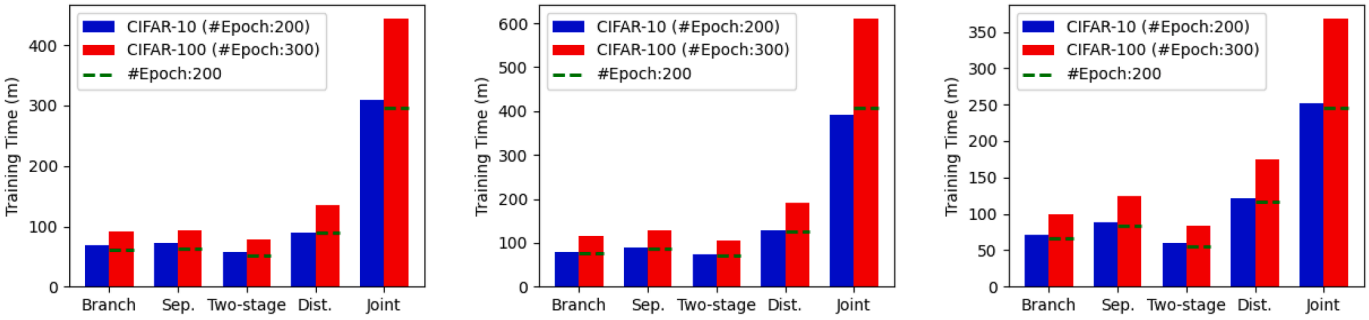
particularly in E-ResNet and E-VGG on CIFAR-100. Branch-wise training displays the most severe overfitting, with sharp train–validation divergence in later exits across datasets—most notably in E-MobileNet and E-ResNet on CIFAR-100.

6.1.3. RQ1 summary

Training strategies show distinct effects on convergence behavior, accuracy, and generalization across exits, models, and datasets. Distillation-based and Joint training remain the most reliable overall: Distillation converges fastest and consistently generalizes best, while Joint training achieves stable and accurate predictions with minimal overfitting across all exits. Separate training performs competitively and often surpasses Two-stage and Branch-wise—particularly in E-MobileNet and E-VGG on CIFAR-100. It maintains strong accuracy at early exits and retains robustness at later ones, though some instability emerges in E-ResNet at deeper exits.

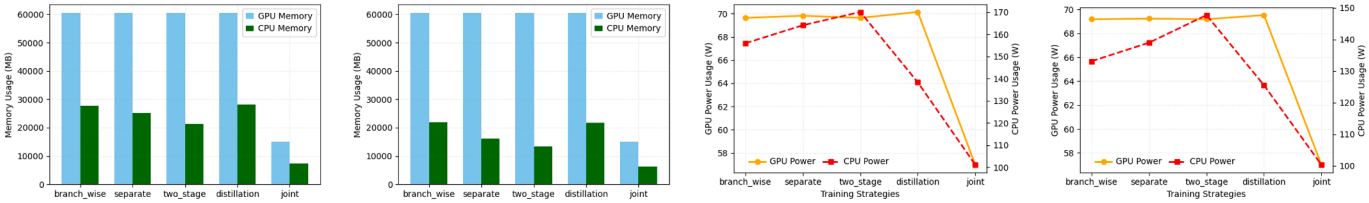
Branch-wise training yields the weakest results across all models and datasets, with the lowest final-exit accuracy and severe validation divergence—especially in CIFAR-100. Two-stage training is more stable than Branch-wise and sometimes matches Separate in shallow exits, but it underperforms at later exits in CIFAR-100, particularly in E-VGG and E-ResNet. Performance differences across strategies are modest on CIFAR-10 but grow significantly on CIFAR-100, where fine-grained classification increases sensitivity to training dynamics and highlights the benefits of stable, globally coordinated optimization.

Statistical insight. ANOVA results (Table 2) confirm statistically significant performance differences across training strategies ($p < 0.05$). These effects are most pronounced in deeper models like E-ResNet and E-MobileNet on CIFAR-100, while E-VGG on CIFAR-10 is comparatively less sensitive. Tukey's HSD post-hoc analyzis (Fig. 11) ranks Distillation-based training highest, with statistically significant margins over all other strategies—especially Branch-wise and Two-stage. Separate training closely approaches Joint in several configurations but shows slightly higher variance. Overall, Distillation-based and Joint training emerge as the most dependable strategies across datasets, offering the best trade-offs between convergence, accuracy, and generalization.



(a) E-MobileNet (b) E-ResNet (c) E-VGG

Fig. 14. Training time for early-exit models E-MobileNet, E-ResNet, and E-VGG on CIFAR-10 and CIFAR-100 under various training strategies. Blue and red bars represent total training time on CIFAR-10 (200 epochs) and CIFAR-100 (300 epochs), respectively. The dashed green line marks the training time for 200 epochs.



(a) CIFAR-10 (Memory) (b) CIFAR-100 (Memory) (c) CIFAR-10 (Power) (d) CIFAR-100 (Power)

Fig. 15. GPU and CPU memory usage and power consumption for E-MobileNet models trained on CIFAR-10 and CIFAR-100. Bars represent memory usage during inference, while markers denote average GPU and CPU power consumption. Results are reported for all five training strategies, highlighting resource overheads associated with each method.

6.2. What are the computational efficiency and training stability characteristics of each strategy, particularly in terms of training time, memory, and power usage? (RQ2)

This section evaluates the trade-offs between computational cost, training stability, and resource efficiency across different strategies. We assess epoch-wise dynamics, training time, memory usage, and power consumption.

6.2.1. Epoch-wise training dynamics

Table 3 summarizes key convergence patterns. Distillation-based and Joint training converge fastest, reaching Min Loss and Peak Accuracy Epochs earlier than others. Branch-wise and Two-stage begin to overfit sooner, particularly at deeper exits. Joint and Two-stage training reach the Saturation Epoch earlier, while Separate training lags. The Generalization Gap is narrowest in Distillation, widening notably in Branch-wise and Separate training. Validation Stability is highest for Distillation-based and Joint, whereas Branch-wise exhibits erratic patterns at later exits. Final Exit Accuracy is led by Distillation, followed closely by Joint training, with Branch-wise performing worst, especially on CIFAR-100.

Key insight. Distillation-based and Joint training optimize performance efficiently, while Branch-wise, Two-stage, and Separate training (with ResNet and VGG on CIFAR-100) struggle with stability and overfitting.

6.2.2. Training time analysis

Figs. 12 and 13 present exit-wise training time trends, while Fig. 14 illustrates overall training time variations.

Exit-wise. Training time increases with exit depth due to added computation. Branch-wise and Separate training are faster at early exits but scale poorly, especially Separate, which trains each exit independently. Two-stage training remains moderate via partial weight sharing, while Distillation training is costlier due to the student-teacher setup. CIFAR-100 requires longer training times, particularly at deeper exits, reflecting their greater feature complexity. Among architectures, E-MobileNet scales gradually, while E-ResNet and E-VGG exhibit sharp time increases at later exits due to network depth.

Overall training time. Total training time varies across strategies. Joint training takes the longest due to simultaneous optimization of all exits, followed by Distillation, which incurs teacher-student overhead. Separate training requires more time than Two-stage and Branch-wise, with Two-stage being the most efficient, closely followed by Branch-wise. Among architectures, E-ResNet and E-VGG require significantly more time than E-MobileNet, reflecting the impact of network depth. In CIFAR-100, training overhead is more pronounced due to the complexity of learning fine-grained features.

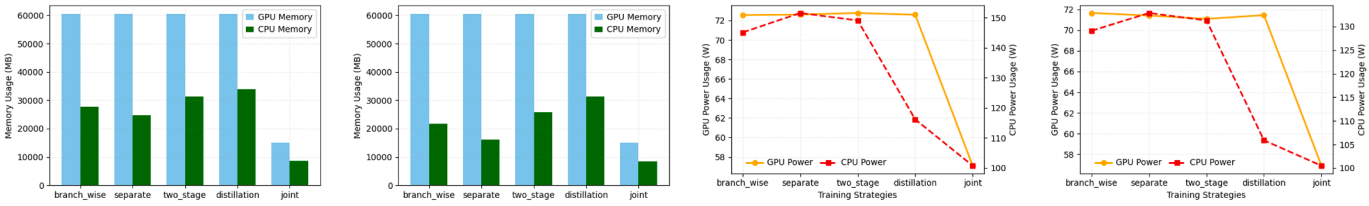
Key insight. Two-stage training achieves an optimal trade-off between efficiency and accuracy. Joint and Distillation yield better performance but demand longer training.

6.2.3. Resource utilization

This section analyze GPU/CPU memory and power consumption (Figs. 15–17) across datasets and models.

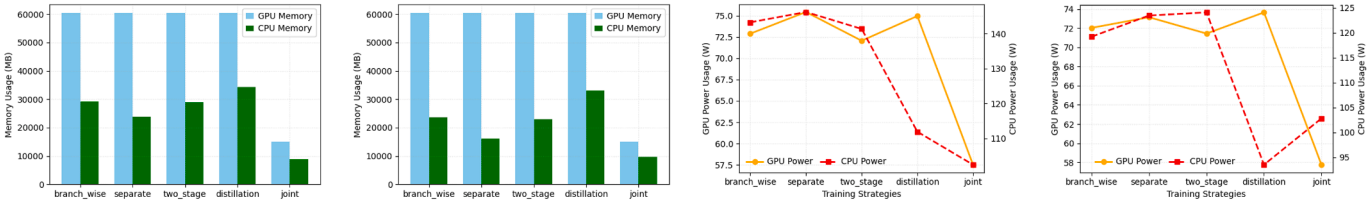
Memory consumption. Figs. 15a, b, 16a, b, 17a, and b illustrate GPU and CPU memory usage across strategies. Joint training minimizes GPU/CPU memory via shared optimization. Separate and Branch-wise are memory-heavy due to independent exit tuning. Distillation has the highest memory footprint due to dual-model learning. Two-stage training balances efficiency, reducing GPU memory. E-ResNet and E-VGG demand more memory than E-MobileNet, particularly for CIFAR-100.

Power consumption. Figs. 15c, d, 16c, d, 17c, and d present the GPU and CPU power consumption across different training strategies and datasets. Joint training is the most energy-efficient overall. Distillation demands the highest GPU power but maintains moderate CPU usage. Branch-wise, Separate, and Two-stage training show similar GPU power



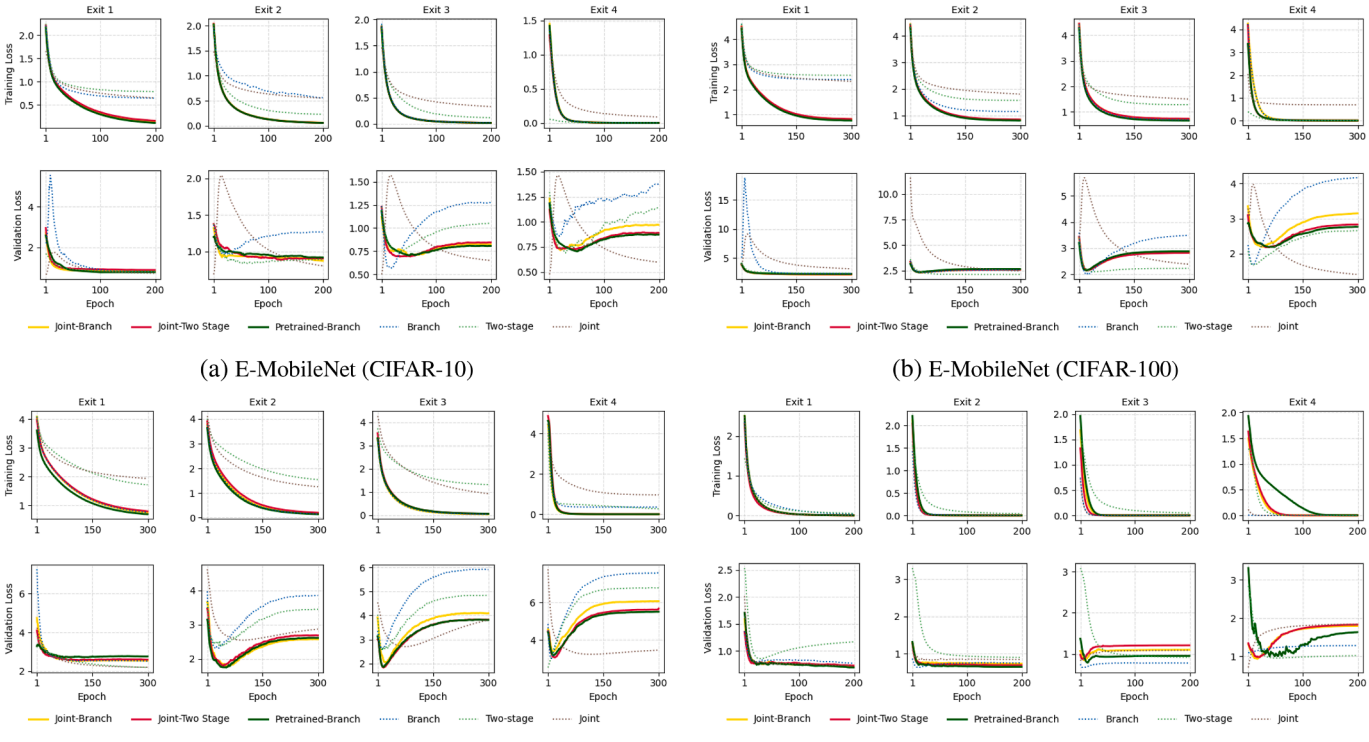
(a) CIFAR-10 (Memory) (b) CIFAR-100 (Memory) (c) CIFAR-10 (Power) (d) CIFAR-100 (Power)

Fig. 16. GPU/CPU memory usage and power consumption for E-ResNet on CIFAR-10 and CIFAR-100 across training strategies. Bars show peak memory; markers indicate average power.



(a) CIFAR-10 (memory) (b) CIFAR-100 (memory) (c) CIFAR-10 (power) (d) CIFAR-100 (power)

Fig. 17. GPU/CPU memory usage and power consumption for E-VGG on CIFAR-10 and CIFAR-100. Bars represent peak memory; markers show average power usage.



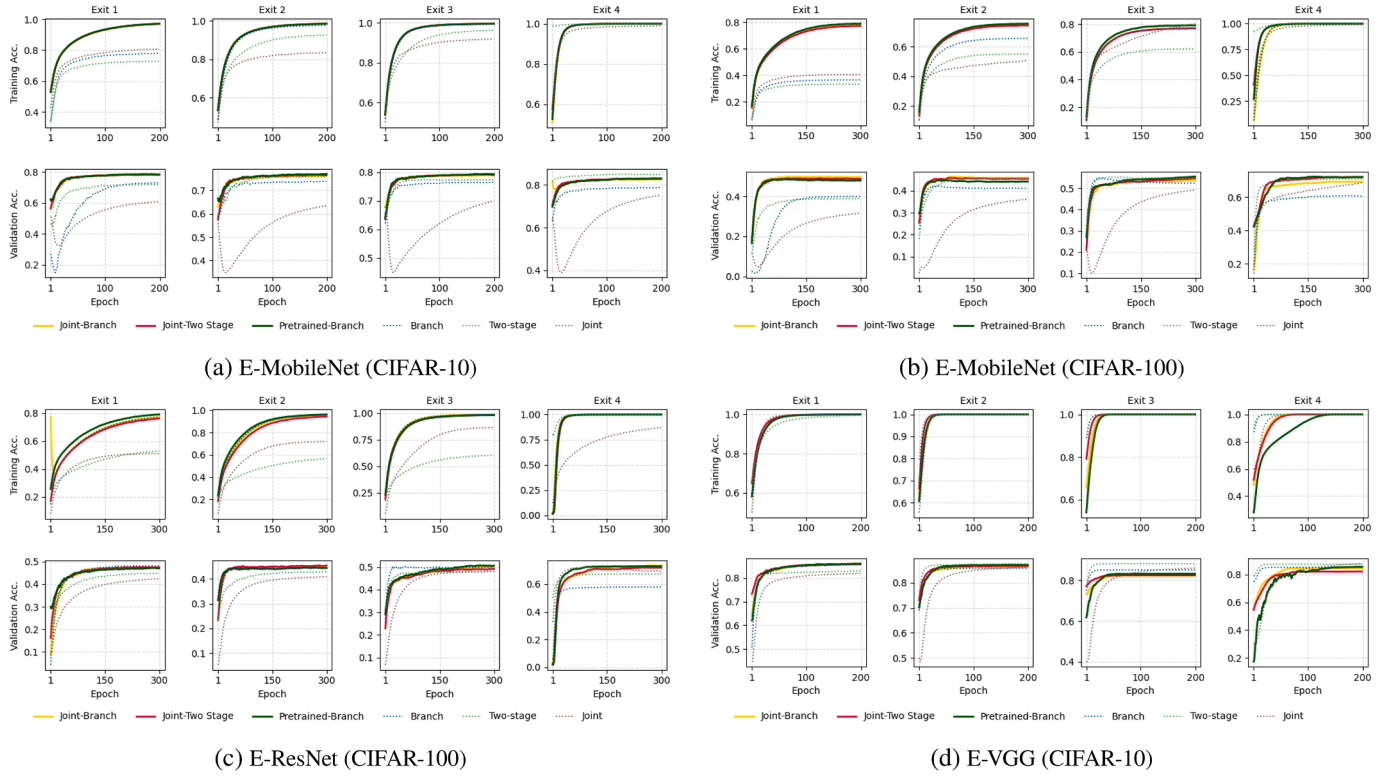
(a) E-MobileNet (CIFAR-10) (b) E-MobileNet (CIFAR-100)

(c) E-ResNet (CIFAR-100) (d) E-VGG (CIFAR-10)

Fig. 18. Exit-wise training and validation loss curves for Hybrid training applied to E-MobileNet, E-ResNet, and E-VGG on CIFAR-10 and CIFAR-100. Each curve tracks the loss trend across exits, highlighting stability and generalization under the Hybrid strategy.

Table 4
ANOVA F-statistics and p-values across all models, datasets, and metrics. Results for GPU and CPU include both power and memory.

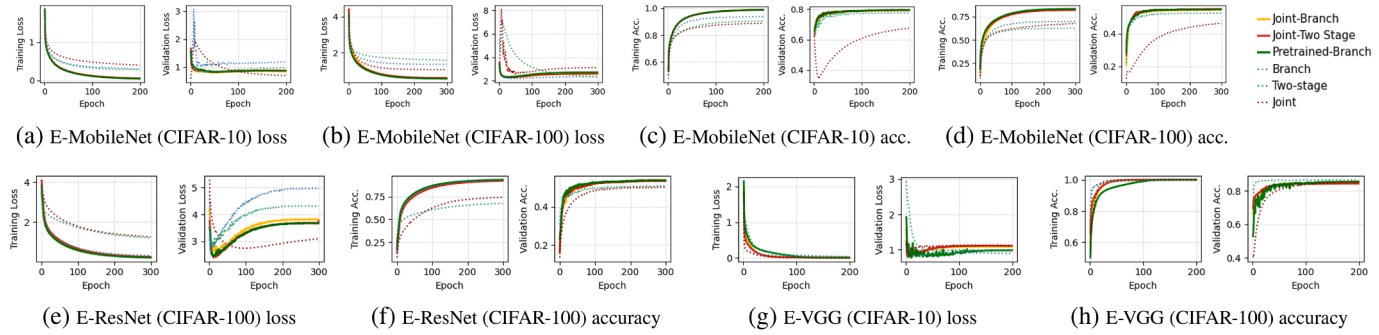
Model (Dataset)	Time F	Time p	GPU Power F/p	GPU Memory F/p	CPU Power F/p	CPU Memory F/p
E-MobileNet (CIFAR10)	2.16E+09	4.8E−86	3.66E+05 / 2.5E−48	2.17E+10 / 4.64E−96	2.25E+06 / 3.18E−56	3.92E+09 / 1.24E−88
E-MobileNet (CIFAR100)	1.93E+09	1.5E−85	1.52E+05 / 1.6E−44	9.09E+09 / 2.78E−92	6.88E+05 / 4.52E−51	9.08E+08 / 2.82E−82
E-ResNet (CIFAR10)	6.69E+09	6.0E−91	7.81E+05 / 1.27E−51	4.16E+10 / 6.9E−99	3.28E+06 / 7.37E−58	9.83E+09 / 1.28E−92
E-ResNet (CIFAR100)	5.66E+09	3.2E−90	2.53E+05 / 9.97E−47	1.39E+10 / 3.95E−94	8.58E+05 / 4.95E−52	2.63E+09 / 6.81E−87
E-VGG (CIFAR10)	1.56E+09	1.3E−84	6.84E+05 / 4.79E−51	2.92E+10 / 2.39E−97	1.86E+06 / 2.14E−55	8.18E+09 / 8.01E−92
E-VGG (CIFAR100)	8.17E+08	8.1E−82	1.56E+05 / 1.28E−44	6.91E+09 / 4.30E−91	3.00E+05 / 1.83E−47	1.56E+09 / 1.27E−84



(a) E-MobileNet (CIFAR-10) (b) E-MobileNet (CIFAR-100)

(c) E-ResNet (CIFAR-100) (d) E-VGG (CIFAR-10)

Fig. 19. Exit-wise training and validation accuracy curves for Hybrid training applied to E-MobileNet, E-ResNet, and E-VGG on CIFAR-10 and CIFAR-100. Each plot shows performance across exits (E_1 to E_4), illustrating how accuracy evolves throughout the network depth.



(a) E-MobileNet (CIFAR-10) loss (b) E-MobileNet (CIFAR-100) loss (c) E-MobileNet (CIFAR-10) acc. (d) E-MobileNet (CIFAR-100) acc.

(e) E-ResNet (CIFAR-100) loss (f) E-ResNet (CIFAR-100) accuracy (g) E-VGG (CIFAR-10) loss (h) E-VGG (CIFAR-100) accuracy

Fig. 20. Overall training loss and accuracy trends for Hybrid training applied to E-MobileNet, E-ResNet, and E-VGG across CIFAR-10 and CIFAR-100. Results reflect model performance aggregated across exits, showing improved convergence behavior under the hybrid strategy.

profiles, though their CPU power usage is higher due to sequential processing. E-MobileNet remains the most efficient architecture, whereas E-ResNet and E-VGG incur significantly higher power costs—particularly on CIFAR-100.

Key insight. Joint training is ideal for constrained environments. Distillation yields stronger performance at higher cost. Two-stage provides the best power-performance balance among lightweight alternatives.

6.2.4. RQ2 summary

The results reveal clear trade-offs between training efficiency, resource consumption, and training stability across strategies. wo-Stage training emerges as the fastest in terms of total training time, completing model optimization significantly quicker than all other strategies. However, this speed comes with moderate power and memory demands. Joint training is the most resource-efficient overall. Despite being the slowest to train, it consistently minimises GPU/CPU memory usage and power consumption through full-model optimization with shared gradients. This makes it particularly well-suited for power-constrained settings or low-memory environments.

Distillation-based training shows moderate training time but is the most resource-intensive strategy, incurring the highest GPU and CPU memory and power usage. Its superior accuracy comes at substantial computational cost, limiting its practicality in constrained deployments. Branch-wise and Separate training strategies are less efficient overall. Both suffer from early overfitting and high memory consumption, particularly Separate training due to its fully independent exit-wise optimization. While they complete training faster than Joint, their instability and overhead make them less attractive under deployment constraints.

At the architectural level, E-MobileNet is the most efficient across all datasets in terms of memory and power, while E-ResNet and E-VGG incur significantly higher costs—especially on CIFAR-100, where fine-grained classification leads to heavier computational load.

Statistical insight. ANOVA results (Table 4) confirm statistically significant differences across training strategies for all efficiency metrics

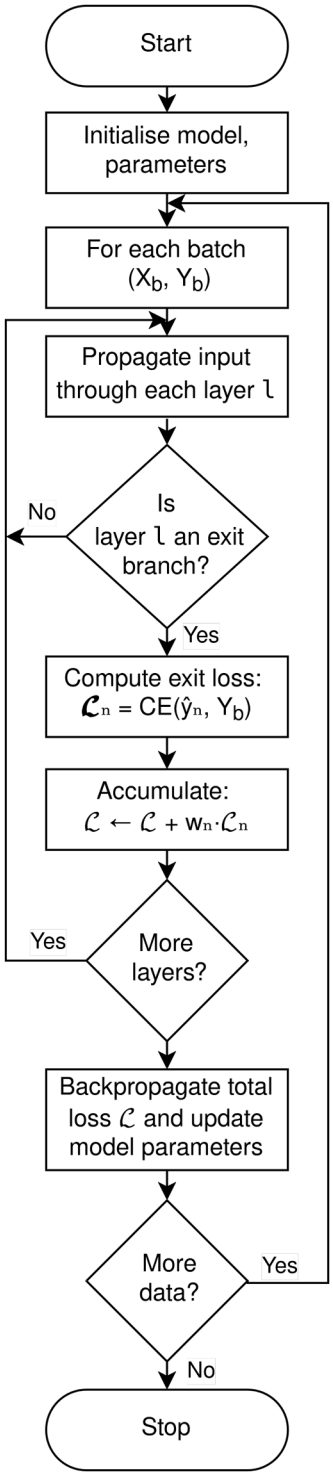


Fig. 21. Joint training.

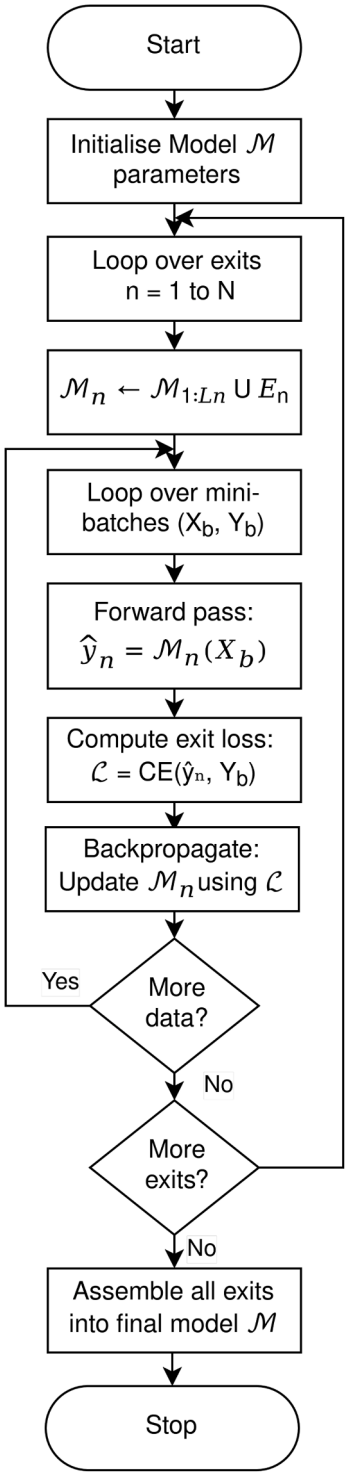


Fig. 22. Separate training.

($p < 0.001$), including training time, GPU/CPU power, and memory usage. Tukey's HSD post-hoc analyzis (Table 5) reveals that Two-stage training achieves the shortest training time, significantly outperforming all other strategies in terms of speed. In contrast, Joint training, while the slowest in terms of total time, ranks highest in GPU/CPU memory and power efficiency. Distillation-based training shows moderate training time but incurs the highest resource usage, ranking lowest in power and memory efficiency. Branch-wise and Separate training fall in the middle for training time but perform poorly in power and memory metrics. These findings establish Two-stage as the fastest, Joint as the most resource-efficient, and Distillation as the most resource-intensive strategy overall.

6.3. *Can hybrid training approaches effectively mitigate overfitting at deeper exits while preserving robust early-exit performance? (RQ3)*

This section evaluates Hybrid training strategies designed to improve convergence, stability, and generalization in early-exit DNNs. Three hybrid configurations—*Joint-branch*, *Joint-two-stage*, and

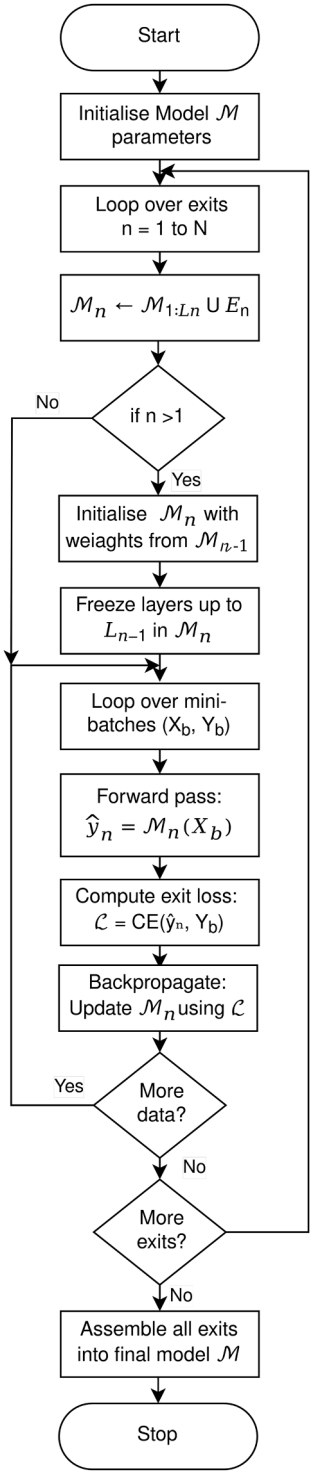


Fig. 23. Branch-wise training.

Pretrained-branch—are compared against their corresponding standalone strategies (Branch, Two-stage, and Joint) across multiple architectures and datasets. Figs. 18 and 19 visualise exit-wise loss and accuracy trends, while Fig. 20 summarises overall performance.

6.3.1. Key findings

1. **Accelerated Convergence:** Hybrid strategies improve loss reduction rates across exits. *Joint-branch* and *Joint-two-stage* converge faster than their standalone counterparts, particularly in E-MobileNet and E-ResNet. *Pretrained-branch* exhibits smooth convergence due to backbone pretraining.
2. **Reduced Overfitting at Later Exits:** All hybrids mitigate the validation loss spikes observed at Exits 3 and 4 in standard strategies. *Pretrained-branch* provides the strongest regularisation, significantly reducing overfitting.
3. **Validation Stability and Accuracy Retention:** *Pretrained-branch* maintains the most stable validation accuracy, especially in E-MobileNet and E-VGG. *Joint-branch* and *Joint-two-stage* also improve late-exit accuracy, with pronounced benefits on CIFAR-100.
4. **Architecture-Specific Sensitivity:** Exit 4 remains the most overfit-prone point, particularly in E-VGG. *Pretrained-branch* best mitigates this trend, while *Joint-branch* still shows mild overfitting in E-MobileNet and E-ResNet.
5. **Dataset Complexity and Hybrid Adaptability:** CIFAR-100 models require 250+ epochs for generalization, whereas CIFAR-10 stabilises earlier (180–200 epochs). Hybrid strategies adapt well to dataset complexity, but E-ResNet benefits from extended training to achieve full generalization.

Table 5
Tukey HSD summaries ($p < 0.05$) for efficiency metrics (lower is better: time, memory, power). Strategy abbreviations: J = Joint, D = Distillation, B = Branch, S = Separate, T = Two-stage. Symbol interpretation: $A \gg B$ means A has a significantly higher (i.e., worse) mean than B.

Model (Dataset)	HSD Summary
Time	
All	J >> D >> B > S > T
GPU Memory	
E-MobileNet (CIFAR10/100)	J >> B = D > S > T
E-ResNet (CIFAR100)	J >> D > B > T > S
E-VGG (CIFAR10/100)	J >> B = D = S = T
GPU Power	
E-MobileNet (CIFAR10)	J >> D >> B = S > T
E-MobileNet (CIFAR100)	J >> D >> S ≈ B = T
E-ResNet (CIFAR10/100)	J >> D > S > B ≈ T
E-VGG (CIFAR10/100)	J >> D > S ≈ B ≈ T
CPU Memory	
E-MobileNet (CIFAR10)	J >> D ≈ B > S >> T
E-MobileNet (CIFAR100)	J >> B ≈ D > S >> T
E-ResNet/VGG (CIFAR10/100)	J >> S > B > T > D
CPU Power	
E-MobileNet (CIFAR10)	J >> D >> B ≈ S > T
E-MobileNet (CIFAR100)	J >> D > B ≈ S > T
E-ResNet (CIFAR10/100)	J >> D >> B ≈ S > T
E-VGG (CIFAR10)	J >> D >> B ≈ S > T
E-VGG (CIFAR100)	J >> D >> S > B ≈ T

Table 6
Performance of non-early-exit baseline models on CIFAR-10 and CIFAR-100. All metrics are averaged over 5 runs.

Model	Dataset	Accuracy (%)	Training Time (min)	GPU Power (W)	CPU Power (W)
MobileNet	CIFAR-10	85.68	30.09	70.14	128.5
MobileNet	CIFAR-100	60.75	40.19	69.20	117.5
ResNet	CIFAR-10	77.32	45.29	73.07	109.8
ResNet	CIFAR-100	63.00	67.57	72.19	107.2
VGG	CIFAR-10	88.25	29.41	75.52	105.8
VGG	CIFAR-100	66.94	42.87	73.61	100.4

6.3.2. RQ3 Summary

Pretrained-branch emerges as the most stable hybrid strategy, effectively reducing validation loss fluctuations and preserving accuracy at deeper exits—especially in E-MobileNet and E-VGG. Joint-based hybrids (*Joint-branch*, *Joint-two-stage*) accelerate convergence and improve generalization, particularly on CIFAR-100. However, hybrid effectiveness is architecture-sensitive: E-MobileNet responds well to early convergence,

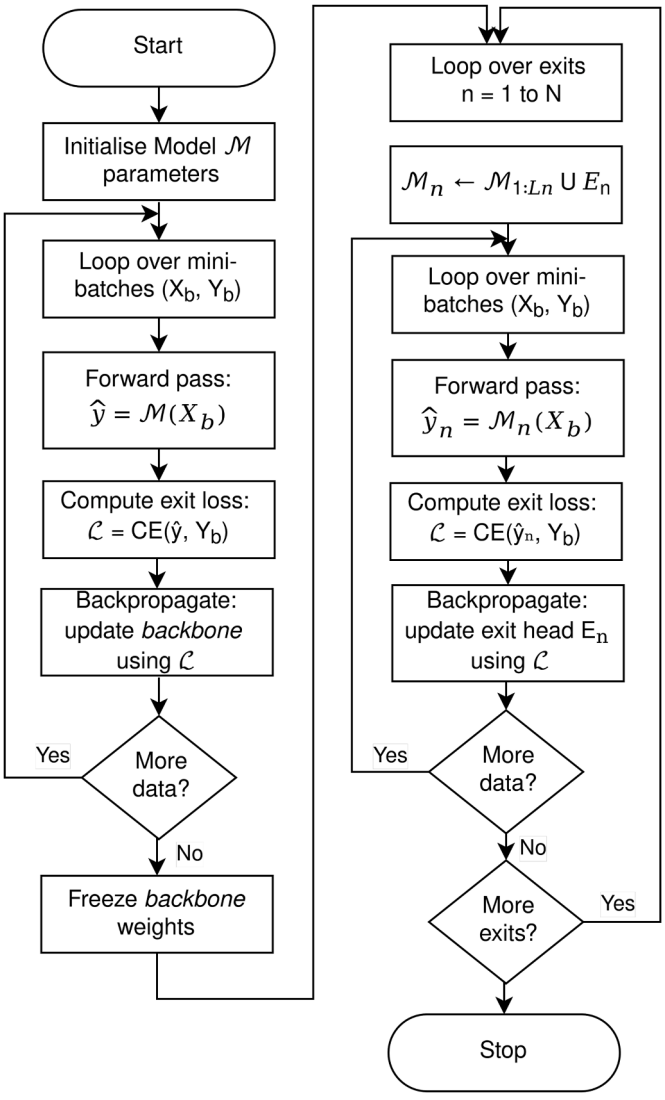


Fig. 24. Two-stage training.

Table 7
Exit-wise accuracy and resource usage on ChestX-ray14 (ResNet-18) and ImageNet-100 (MobileNet-V2); five-run averages.

Strategy	ChestX-ray14 – ResNet-18								ImageNet-100 – MobileNet V2							
	Accuracy (%)				Loss Trend	Time(min)	Power (W)		Accuracy (%)				Loss Trend	Time (min)	Power (W)	
	E_1	E_2	E_3	E_4			GPU	CPU	E_1	E_2	E_3	E_4			GPU	CPU
No-early-exit	–	–	–	91.0	–	44.6	72.7	117.9	–	–	–	66.1	–	75.4	65.7	120.2
Joint	83.5	88.9	90.7	91.3	Stable	47.3	73.2	115.6	48.5	60.9	66.3	67.4	Stable	79.2	64.2	118.1
Separate	79.0	85.4	86.2	87.6	Unstable	58.8	70.1	112.3	44.0	56.7	63.8	65.2	Unstable	89.6	68.0	125.0
Branch-wise	76.5	84.3	86.0	86.5	Divergent	60.9	69.4	110.7	40.2	52.3	59.7	64.1	Divergent	94.1	60.2	112.1
Two-stage	81.8	86.6	88.5	88.2	Moderate	51.2	71.2	113.2	46.8	59.5	65.4	66.2	Moderate	82.5	63.1	116.5
Distillation-based	80.2	87.5	88.2	90.9	Stable	52.8	71.7	113.5	47.1	60.1	65.8	66.9	Stable	85.7	63.4	117.2
Joint-branch	82.6	88.4	89.7	91.9	Stable	49.9	72.4	114.1	49.3	61.6	67.0	68.3	Stable	83.4	63.9	117.6
Joint-two-stage	83.1	88.6	90.3	91.6	Stable	50.7	72.1	114.4	50.0	62.0	67.4	68.2	Stable	84.0	63.6	117.3
Pretrained-branch	82.8	88.9	89.8	91.4	Stable	53.3	70.9	113.0	46.2	60.0	65.5	67.0	Stable	81.8	62.8	116.0

while E-ResNet requires prolonged training for full benefit. In summary, hybrid training improves robustness and mitigates overfitting in early-exit networks, but success depends on careful alignment between strategy, architecture, and dataset complexity.

6.4. Contextual baseline: standard non-early-exit models

While our primary focus is on evaluating training strategies for early-exit DNNs, this section provides a contextual reference by benchmarking standard full-depth models. Specifically, we assess baseline versions of MobileNet, ResNet, and VGG trained end-to-end on CIFAR-10 and CIFAR-100 without any intermediate exits or adaptive inference mechanisms. Table 6 reports the average validation accuracy, training time, and power consumption over five independent runs. These conventional architectures follow a fixed-depth computation path and lack dynamic exit capability, applying uniform computational effort across all inputs.

Although non-early-exit models deliver competitive final-exit accuracy, they do so at the cost of inflexible resource usage. In contrast, early-exit architectures enable conditional computation by routing easier inputs through shallower branches—potentially reducing latency and

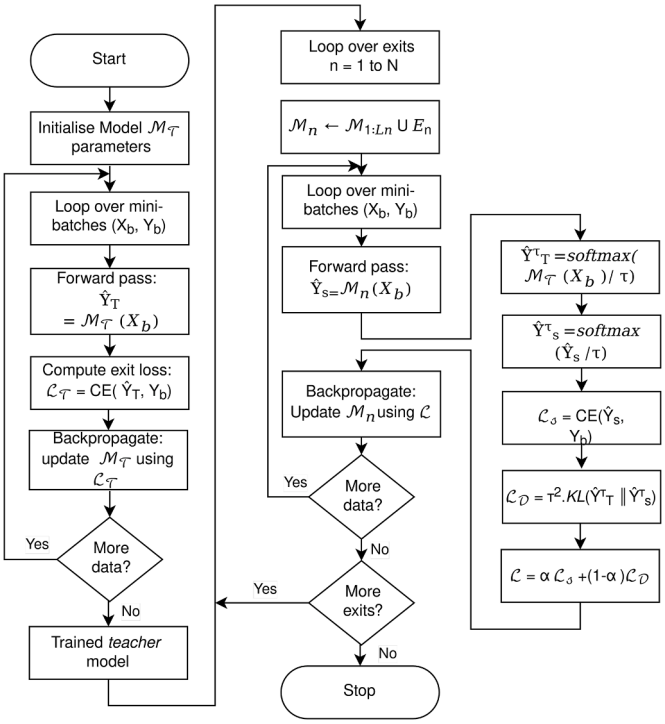


Fig. 25. Distillation-based training.

Table 8
Comparison of training strategies and their characteristics.

Strategy	Training time	Resource usage	Convergence behavior	Accuracy	Generalization strength	Overfitting risk	Severity
Joint	Moderate–High	Low	Smooth and stable	High	Strong	Moderate	Mild
Separate	High	High	Unstable beyond Exit-2	Moderate	Moderate (early)	High	Severe
Branch-wise	High	High	Slow and unstable	Low	Weak	Very High	Severe
Two-stage	Lowest	Moderate	Gradual and stable	Moderate	Moderate	Moderate	Moderate
Distillation	High	High	Fastest and stable	High	Strong	Low	Low
Hybrid	High	High	Fast and more stable	Highest	Strongest	Low–Moderate	Low

energy consumption. The performance trends presented earlier demonstrate how various training strategies can unlock these benefits while balancing accuracy, efficiency, and training stability.

7. Additional evaluation on large-scale and cross-domain datasets

To evaluate whether the trends observed on CIFAR-10/100 hold under more challenging conditions, we extend our experiments to two additional benchmarks: (1) a binary pneumonia-vs-normal classification task from ChestX-ray14 using E-ResNet-18, and (2) ImageNet-100, a 100-class subset of ILSVRC-2012, using E-MobileNet-V2. Each model includes four early exits, and all reported values are averaged over five independent runs.

Table 7 presents exit-wise accuracy, training time, and average GPU/CPU power consumption. Across both datasets, the relative ordering of training strategies largely mirrors that observed in CIFAR-10/100. On ChestX-ray14, final-exit accuracy peaks at 91.9 % using *Joint-branch* training, with other hybrid strategies—*Joint-two-stage* and *Pretrained-branch*—closely following. Joint and Distillation-based training maintain high accuracy at both early and final exits, confirming their generalization strength. In contrast, Separate and Branch-wise training underperform: they converge slowly, overfit deeper exits, and trail by up to 7–8 percentage points at early exits (E_1–E_2), while consuming more wall-clock time.

On ImageNet-100, *Joint-two-stage* again yields the highest final accuracy (68.2 %), marginally outperforming *Joint-branch* and *Pretrained-branch*. Early-exit accuracy is notably lower overall—due to ImageNet's higher complexity and resolution—but consistent trends persist: Separate and Branch-wise remain the weakest, and hybrids close the gap to Joint while improving efficiency.

Training time and power consumption trends also reinforce earlier findings. Joint and Distillation incur higher resource usage but achieve strong performance. Hybrid strategies offer balanced trade-offs: they recover most of Joint's accuracy while reducing training time by 5–10 min and average GPU power by 1–2 W. Among lightweight strategies, *Pretrained-branch* offers the best compromise between early-exit performance and cost, whereas Two-stage is most efficient in wall-clock time. Separate and Branch-wise are consistently the least efficient, with longer training durations and higher CPU load.

Overall, we observe that training strategies that optimize the full backbone—either directly (Joint) or via a teacher (Distillation)—consistently deliver superior accuracy and generalization. Hybrid strategies achieve near-optimal performance with better efficiency. In contrast, approaches that treat exits in isolation (Separate, Branch-wise) are less reliable across domains, architectures, and resolutions.

8. Discussion

This study systematically evaluates the trade-offs among early-exit training strategies in terms of convergence, generalization, and resource efficiency across diverse architectures and datasets. Tables 8 and 9 summarise core characteristics, and ANOVA with Tukey's HSD confirms

Table 9
Strengths and weaknesses of different training strategies.

Strategy	Strengths	Weaknesses
Joint	Promotes holistic optimization via shared gradients across exits. Enables efficient training with low memory and power usage. Demonstrates strong generalization and stable convergence.	Requires careful loss-weight balancing; may underfit early exits without tuning. Scalability challenges for deeper or more complex models.
Branch-wise	Allows independent exit-wise training. Reduces redundant computation per stage and enables partial reuse of pretrained layers.	Frozen shallow layers limit adaptability. Poor generalization at deeper exits due to absence of coordinated learning.
Separate	Maximizes per-exit flexibility and modular design. Useful for debugging or task-specific adaptation.	No inter-exit coordination leads to instability. Computationally inefficient due to repeated training and backbone re-initialization.
Two-stage	Highly time-efficient and compatible with pretrained backbones. Requires fewer gradient updates. Offers stable performance for shallower exits.	Final-exit accuracy can suffer due to fixed backbone. Early exits may receive misaligned features, reducing generalization.
Distillation-based	Boosts early-exit performance via soft supervision from a strong teacher. Balances accuracy and stability.	Adds overhead via dual-model training. Sensitive to teacher quality and requires tuning of distillation parameters.
Hybrid	Combines global and local strategies to improve convergence, accuracy, and robustness. Reduces overfitting at deeper exits.	Requires additional tuning and extended training. Implementation complexity increases due to multi-phase optimization.

statistically significant differences across strategies ($p < 0.001$), models, and datasets ($p < 0.05$).

Distillation-based training consistently delivers strong performance across exits, with fast convergence and high final-exit accuracy—even on large-scale and cross-domain datasets. However, its student–teacher formulation introduces added computational overhead in terms of training time, GPU/CPU power, and memory usage. Joint training, in contrast, achieves near-equivalent accuracy while being the most computationally efficient strategy. Its shared optimization reduces resource usage and ensures stable convergence, particularly on deeper networks like E-VGG and cross-domain settings like ChestX-ray14.

Two-stage training emerges as the most time-efficient strategy, providing acceptable generalization with minimal training cost. It is especially effective for scenarios prioritizing training speed or decoupled optimization. However, its fixed backbone limits alignment with shallow exit requirements, leading to reduced accuracy at early exits—particularly in high-resolution datasets like ImageNet-100.

Separate and Branch-wise training are the least reliable. Both strategies exhibit degraded performance at deeper exits due to lack of coordinated learning. In Separate training, reinitializing and retraining each exit in isolation leads to inconsistencies and early overfitting. In Branch-wise training, the frozen backbone restricts feature adaptation for deeper exits. These limitations are particularly severe in deeper architectures such as E-ResNet and E-VGG on CIFAR-100, where feature hierarchies are critical for generalization. Tukey's HSD analyzis confirms their significantly weaker performance across accuracy and stability metrics.

Hybrid training offers a balanced solution by combining the benefits of global and local optimization. *Pretrained-branch* achieves the most stable validation behavior, while *Joint-branch* and *Joint-two-stage* maintain high accuracy and faster convergence without excessive overhead. These hybrids significantly reduce overfitting at later exits and adapt well to complex datasets like CIFAR-100 and ChestX-ray14. Notably, *Joint-two-stage* consistently achieves the best final-exit accuracy on ImageNet-100, highlighting the practical value of hybrid schedules even under high-resolution and large-scale scenarios.

Among architectures, E-MobileNet demonstrates the highest efficiency across time and power metrics, making it a suitable backbone for resource-constrained deployment. E-ResNet and E-VGG offer higher accuracy ceilings but incur greater cost, especially on CIFAR-100. These findings are consistent across both clean and domain-specific datasets, as validated by additional evaluation in Section 7.

To contextualize the efficiency–accuracy trade-offs of early-exit training, we also benchmarked standard DNN backbones without early exits (Section 6.4, Table 6). While these models offer strong final accuracy, they incur uniformly high resource usage across inputs. In contrast, early-exit networks—when trained with appropriate strategies like Joint or Hybrid—achieve comparable or superior accuracy with significantly lower training time, power, and memory usage. These results reinforce the practical value of adaptive inference over fixed-depth computation, especially for energy-aware or latency-sensitive deployments.

Although our experiments were conducted on high-end GPUs (Tesla V100) and CPUs (Xeon), the findings have direct implications for edge deployment. In particular, Two-stage and *Pretrained-branch* strategies are attractive for embedded platforms due to their reduced memory footprint and decoupled exit training. These approaches avoid full backpropagation and complex teacher networks, making them well-suited for deployment on hardware such as Jetson Nano or Raspberry Pi. While profiling on such platforms is beyond the scope of this study, future work could benchmark real-world energy, latency, and performance to guide deployment decisions.

8.1. Guidelines for selecting training strategies in deployment

Choosing the right training strategy depends on deployment priorities—whether maximizing accuracy, minimizing computational overhead, or striking a balance between the two. Based on our empirical findings and statistical analyzis, we recommend the following practical guidelines:

- Distillation-based training is ideal when accuracy and generalization are top priorities. However, it demands the highest memory and power consumption, making it less suitable for resource-constrained environments.
- Joint training minimizes GPU/CPU memory and power usage through shared optimization, making it a strong candidate for deployment under limited resources. It slightly trails Distillation in accuracy and requires the longest training time.
- Two-stage training offers the fastest training time and moderate generalization, making it well-suited for rapid prototyping or low-resource setups where training efficiency is critical.
- Separate training provides flexibility for task-specific exits but suffers in deeper networks and complex datasets due to lack of coordinated optimization.
- Branch-wise training is generally prone to instability and overfitting, but can be effective when fine-tuning shallow, pretrained models. It is not recommended for deep networks trained from scratch.
- Hybrid Strategies, especially *Pretrained-branch*, effectively mitigate overfitting at deeper exits while maintaining high accuracy. They provide a good balance between efficiency and generalization but may require architecture-specific tuning.
- Architectural Considerations: *E-MobileNet* is the most resource-efficient, making it ideal for edge deployment. In contrast, *E-ResNet* and *E-VGG* offer stronger representational power but demand stable training strategies like Distillation or Hybrid approaches to reach their full potential.

These guidelines provide a structured framework for selecting training methodologies in early-exit DNNs, enabling practitioners to balance accuracy, efficiency, and stability under real-world deployment constraints.

9. Limitations and future directions

While this benchmark spans four datasets and three CNN backbones, all experiments focus exclusively on image classification. Extending the evaluation to more complex tasks—such as full-scale ImageNet-1k, MS-COCO object detection, or Kinetics-400 video recognition—would strengthen the generalizability of our findings. Applying the same training strategies to non-convolutional architectures (e.g., Vision Transformers, MLP-Mixers) and multimodal backbones represents a critical next step.

A key limitation remains the instability of exits under certain strategies. Separate and Branch-wise training consistently overfit at deeper exits, underscoring the need for lightweight coordination mechanisms. Future work could explore auxiliary losses, shared regularizers, or exit-aware constraints to improve consistency across exits. The hybrid training space is also far from exhausted. More sophisticated variants—such as curriculum-guided, reinforcement-learning-based, or uncertainty-aware hybrids—may offer better trade-offs between generalization, training stability, and computational efficiency. Lastly, all performance profiling was conducted on data-centre hardware (e.g., Tesla V100). Evaluating these strategies on edge devices—such as Jetson Nano, Raspberry Pi, or Coral TPU—remains a vital direction for deployment-ready early-exit systems. Coupling the training schedules with real-time exit policies under latency or energy constraints would enable practical, adaptive inference in real-world applications.

10. Conclusion

We conducted a comprehensive benchmark of eight early-exit training strategies across two core datasets (CIFAR-10 and CIFAR-100), further validating our findings on two additional settings (ImageNet-100 and ChestX-ray14). Strategies that optimize the full backbone—either via joint training or knowledge distillation—consistently achieve the strongest exit-wise accuracy. Hybrid variants closely match this performance while offering a more favourable accuracy–efficiency trade-off. Two-stage training emerges as the fastest strategy, albeit with a slight compromise in final-exit accuracy. In contrast, methods that optimize exits independently—such as Separate and Branch-wise training—exhibit instability and reduced reliability at deeper exits. These insights provide actionable guidelines for selecting training strategies in early-exit DNNs, enabling informed deployment under varying resource and performance constraints. They also motivate future work on adaptive, scalable, and hardware-aware training paradigms for efficient deep inference.

CRediT authorship contribution statement

Haseena Rahmath P: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **Kuldeep Chaurasia:** Writing – review & editing, Supervision, Resources, Project administration; **Abhay Bansal:** Writing – review & editing, Supervision, Resources, Project administration.

Data availability statement

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors received no specific funding for this research.

Appendix A. Flowcharts of training strategies

This appendix provides flowcharts that visually explain the five core early-exit training strategies benchmarked in our study. Each figure highlights how exit branches are optimized, scheduled, and integrated during training. Below, we summarize the logic and implementation principles behind each approach:

Joint Training (Fig. 21): All exit branches and the backbone are optimized simultaneously. At each iteration, losses from all exits are combined—optionally with weighted coefficients—and backpropagation updates the full model jointly. This approach encourages uniform learning but may suffer from competition between shallow and deep exits.

Separate Training (Fig. 22): Each exit is trained independently as a separate sub-network, using a truncated backbone up to its attachment point. After training all exits separately, they are assembled into a unified model. This method is simple and parallelizable but can cause instability across exits due to the lack of shared supervision.

Branch-wise Training (Fig. 23): Exits are optimized sequentially, one at a time. Each training stage initializes from the previous one, but backbone layers below already-trained exits are frozen. This encourages localized learning and reduces interference, but it limits representation sharing and often leads to overfitting in deeper exits.

Two-stage Training (Fig. 24): The model is trained in two distinct phases: first, the full backbone and final exit are trained conventionally. Then, the backbone is frozen, and early exits are trained individually using the frozen features. This reduces gradient interference but risks weak supervision for shallow exits.

Distillation-based Training (Fig. 25): The final exit acts as a teacher, guiding early exits using soft labels. After training the full model, the final exit is frozen. Then, each early exit is trained with a combination of hard-label loss and Kullback–Leibler divergence from the teacher's softened outputs. This balances supervision depth and consistency across exits.

These diagrams serve as implementation-level references for practitioners and researchers working on efficient, multi-exit DNN training pipelines.

References

Barinov, R., Gai, V., Kuznetsov, G., & Golubenko, V. (2023). Automatic evaluation of neural network training results. *Computers*, *12*(2), 26. https://doi.org/10.3390/computers12020026

Belilovsky, E., Eickenberg, M., & Oyallon, E. (2019). Greedy layerwise learning can scale to imagenet. In *International conference on machine learning* (pp. 583–593). PMLR.

Berestizshevsky, K., & Even, G. (2019). Dynamically sacrificing accuracy for reduced computation: Cascaded inference based on softmax confidence. In *International conference on artificial neural networks* (pp. 306–320). Springer.

Bolukbasi, T., Wang, J., Dekel, O., & Saligrama, V. (2017). Adaptive neural networks for fast test-time prediction. arXiv preprint arXiv:1702.07811 *1*(3).

Brock, A., Lim, T., Ritchie, J. M., & Weston, N. (2017). Freezeout: Accelerate training by progressively freezing layers. arXiv preprint arXiv:1706.04983

Chun-Hsiao Yeh, Y. C. (2022). IN100pytorch: Pytorch implementation: Training resnets on imagenet-100. https://github.com/danielchyeh/ImageNet-100-Pytorch.

Ding, Y., Kang, W., Yang, A., Zhang, Z., Zhao, J., Feng, J., Hong, D., & Zheng, Q. (2025a). Adaptive homophily clustering: A structure homophily graph learning with adaptive filter for hyperspectral image. arXiv preprint arXiv:2501.01595

Ding, Y., Zhang, Z., Yang, A., Cai, Y., Xiao, X., Hong, D., & Yuan, J. (2025b). Slcgc: A lightweight self-supervised low-pass contrastive graph clustering network for hyperspectral images. arXiv preprint arXiv:2502.03497

Figurnov, M., Collins, M. D., Zhu, Y., Zhang, L., Huang, J., Vetrov, D., & Salakhutdinov, R. (2017). Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1039–1048).

Gao, X., Liu, Y., Huang, T., & Hou, Z. (2023). Pf-berxit: Early exiting for bert with parameter-efficient fine-tuning and flexible early exiting strategy. *Neurocomputing*, *558*, 126690.

Garg, S., & Moschitti, A. (2021). Will this question be answered? question filtering via answer model distillation for efficient question answering. In M.-F. Moens, X. Huang, L. Specia, & S. W.-t. Yih (Eds.), *Proceedings of the 2021 conference on empirical methods in natural language processing* (pp. 7329–7346). Online and Punta Cana, Dominican Republic: ACL.

Han, Y., Huang, G., Song, S., Yang, L., Wang, H., & Wang, Y. (2021). Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*(11), 7436–7456.

He, J., Zhang, Q., Ding, W., Miao, D., Zhao, J., Hu, L., & Cao, L. (2024). De^3-bert: Distance-enhanced early exiting for BERT based on prototypical networks. CoRR, abs/2402.05948

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778).

Hettinger, C., Christensen, T., Ehlert, B., Humpherys, J., Jarvis, T., & Wade, S. (2017). Forward thinking: Building and training neural networks one layer at a time. arXiv preprint arXiv:1706.02480

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861

Huang, G., Chen, D., Li, T., Wu, F., van der Maaten, L., & Weinberger, K. (2018). Multi-scale dense networks for resource efficient image classification. In *International conference on learning representations*.

Ilhan, F., Chow, K.-H., Hu, S., Huang, T., Tekin, S., Wei, W., Wu, Y., Lee, M., Kompella, R., Latapie, H. et al. (2024). Adaptive deep neural network inference optimization with eenet. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision* (pp. 1373–1382).

Jo, J., Kim, G., Kim, S., & Park, J. (2023). Locoexnet: Low-cost early exit network for energy efficient CNN accelerator design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *42*(12), 4909–4921.

Karpikova, P., Radionova, E., Yaschenko, A., Spiridonov, A., Kostyushko, L., Fabbricatore, R., & Ivakhnenko, A. (2023). Fiancee: Faster inference of adversarial networks via conditional early exits. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)* (pp. 12032–12043). New Orleans, LA, USA: IEEE.

Kaya, Y., Hong, S., & Dumitras, T. (2019). Shallow-deep networks: Understanding and mitigating network overthinking. In *International conference on machine learning* (pp. 3301–3310). PMLR.

Krizhevsky, A., Hinton, G. et al. (2009). Learning multiple layers of features from tiny images.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, *25*,1097–1105.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90.

Laskaridis, S., Kouris, A., & Lane, N. D. (2021). Adaptive inference through early-exit networks: Design, challenges and directions. In *Proceedings of the 5th international workshop on embedded and mobile deep learning* (pp. 1–6).

Laskaridis, S., Venieris, S. I., Almeida, M., Leontiadis, I., & Lane, N. D. (2020). Spinn: Synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th annual international conference on mobile computing and networking* MobiCom '20 (pp. 1–15). London, United Kingdom: Association for Computing Machinery.

Lee, C.-Y., Xie, S., Gallagher, P., Zhang, Z., & Tu, Z. (2015). Deeply-supervised nets. In *Artificial intelligence and statistics* (pp. 562–570). PMLR.

Leontiadis, I., Laskaridis, S., Venieris, S. I., & Lane, N. D. (2021). It's always personal: Using early exits for efficient on-device CNN personalisation. In *Proceedings of the 22nd international workshop on mobile computing systems and applications* (pp. 15–21).

Li, E., Zeng, L., Zhou, Z., & Chen, X. (2019a). Edge AI: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications*, *19*(1), 447–457.

Li, H., Zhang, H., Qi, X., Yang, R., & Huang, G. (2019b). Improved techniques for training adaptive deep networks. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 1891–1900).

Li, Z., Yang, Y., Liu, X., Zhou, F., Wen, S., & Xu, W. (2017). Dynamic computational time for visual attention. In *Proceedings of the IEEE international conference on computer vision workshops* (pp. 1199–1209).

Lin, J., Rao, Y., Lu, J., & Zhou, J. (2017). Runtime neural pruning. *Advances in Neural Information Processing Systems*, *30*,2181–2191.

Liu, S., Zeng, Z., Ren, T., Li, F., Zhang, H., Yang, J., Jiang, Q., Li, C., Yang, J., Su, H. et al. (2024). Grounding dino: Marrying dino with grounded pre-training for open-set object detection. In *European conference on computer vision* (pp. 38–55). Springer.

Liu, W., Zhou, P., Wang, Z., Zhao, Z., Deng, H., & Ju, Q. (2020). FastBERT: A self-distilling BERT with adaptive inference time. In *Proceedings of the 58th annual meeting of the association for computational linguistics* (pp. 6035–6044).

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. CoRR, abs/1907.11692

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., & Xie, S. (2022). A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 11976–11986).

Marquez, E. S., Hare, J. S., & Niranjan, M. (2018). Deep cascade learning. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(11), 5475–5485.

Matsubara, Y., Levorato, M., & Restuccia, F. (2022). Split computing and early exiting for deep learning applications: Survey and research challenges. *ACM Computing Surveys*, *55*(5), 1–30.

Moon, T., Choi, M., Yun, E., Yoon, J., Lee, G., & Lee, J. (2023). Early exiting for accelerated inference in diffusion models. In *Icml 2023 workshop on structured probabilistic inference & generative modeling*. Honolulu, HI, USA: PMLR.

Mullapudi, R. T., Mark, W. R., Shazeer, N., & Fatahalian, K. (2018). Hydranets: Specialized dynamic architectures for efficient inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 8080–8089).

Pacheco, R. G., Bochie, K., Gilbert, M. S., Couto, R. S., & Campista, M. E. M. (2021a). Towards edge computing using early-exit convolutional neural networks. *Information*, *12*(10), 431.

Pacheco, R. G., & Couto, R. S. (2020). Inference time optimization using branchynet partitioning. In *2020 IEEE symposium on computers and communications (ISCC)* (pp. 1–6). IEEE.

Pacheco, R. G., Oliveira, F. D., & Couto, R. S. (2021b). Early-exit deep neural networks for distorted images: Providing an efficient edge offloading. In *2021 IEEE global communications conference (GLOBECOM)* (pp. 1–6). IEEE.

Panda, P., Sengupta, A., & Roy, K. (2016). Conditional deep learning for energy-efficient and enhanced pattern recognition. In *2016 design, automation & test in Europe conference & exhibition (DATE)* (pp. 475–480). IEEE.

Panda, P., Sengupta, A., & Roy, K. (2017). Energy-efficient and improved image recognition with conditional deep learning. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, *13*(3), 1–21.

Phuong, M., & Lampert, C. (2019). Distillation-based training for multi-exit architectures. In *2019 IEEE/CVF international conference on computer vision (ICCV)* (pp. 1355–1364). https://doi.org/10.1109/ICCV.2019.00144

Rahmath P, H., & Chaurasia, K. (2024). Adaptive early-exit inference in graph neural networks based hyperspectral image classification. In A. Abraham, A. Bajaj, T. Hanne, P. Siarry, & K. Ma (Eds.), *Intelligent systems design and applications* (pp. 444–453). Cham: Springer Nature Switzerland.

Rahmath P, H., Chaurasia, K., & Anika (2025). E2g-net: Enhancing efficiency in graph neural networks with early-exit branches. *IEEE Transactions on Emerging Topics in Computational Intelligence*, (pp. 1–11). https://doi.org/10.1109/TETCI.2025.3573240

Rahmath P, H., Chaurasia, K., Gupta, A., & Srivastava, V. (2024a). HyperGCN–a multi-layer multi-exit graph neural network to enhance hyperspectral image classification. *International Journal of Remote Sensing*, *45*(14), 4848–4882.

Rahmath P, H., Srivastava, V., & Chaurasia, K. (2023). A strategy to accelerate the inference of a complex deep neural network. In *Proceedings of data analytics and management: ICDAM 2022* (pp. 57–68). Springer.

Rahmath P, H., Srivastava, V., Chaurasia, K., Pacheco, R. G., & Couto, R. S. (2024b). Early-exit deep neural network - a comprehensive survey. *ACM Computing Surveys*. Just Accepted https://doi.org/10.1145/3698767. https://doi.org/10.1145/3698767

Sabet, A., Hare, J. S., Al-Hashimi, B. M., & Merrett, G. V. (2021). Temporal early exits for efficient video object detection. CoRR, abs/2106.11208 https://arxiv.org/abs/2106.11208.

Scardapane, S., Scarpiniti, M., Baccarelli, E., & Uncini, A. (2020). Why should we add early exits to neural networks? *Cognitive Computation*, *12*(5), 954–966.

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556

Teerapittayanon, S., McDanel, B., & Kung, H.-T. (2016). Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)* (pp. 2464–2469). IEEE.

Tian, Y., Krishnan, D., & Isola, P. (2020). Contrastive multiview coding. In *Computer vision–ECCV 2020: 16th European conference, Glasgow, UK, August 23–28, 2020, proceedings, part XI 16* (pp. 776–794). Springer.

Veit, A., & Belongie, S. (2018). Convolutional networks with adaptive inference graphs. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 3–18).

Wang, G., Xie, X., Lai, J., & Zhuo, J. (2017a). Deep growing learning. In *Proceedings of the IEEE international conference on computer vision* (pp. 2812–2820).

Wang, M., Mo, J., Lin, J., Wang, Z., & Du, L. (2019). Dynexit: A dynamic early-exit strategy for deep residual networks. In *2019 IEEE international workshop on signal processing systems (siPS)* (pp. 178–183). https://doi.org/10.1109/SiPS47522.2019.9020551

Wang, Q., Fang, W., & Xiong, N. N. (2024). Tlee: Temporal-wise and layer-wise early exiting network for efficient video recognition on edge devices. *IEEE Internet of Things Journal*, *11*(2), 2842–2854.

Wang, X., Peng, Y., Lu, L., Lu, Z., Bagheri, M., & Summers, R. M. (2017b). Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2097–2106).

Wang, X., Yu, F., Dou, Z.-Y., Darrell, T., & Gonzalez, J. E. (2018). Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 409–424).

Wang, Y., Shen, J., Hu, T.-K., Xu, P., Nguyen, T., Baraniuk, R., Wang, Z., & Lin, Y. (2020). Dual dynamic inference: Enabling more efficient, adaptive, and controllable deep inference. *IEEE Journal of Selected Topics in Signal Processing*, *14*(4), 623–633.

Wu, Z., Nagarajan, T., Kumar, A., Rennie, S., Davis, L. S., Grauman, K., & Feris, R. (2018). Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 8817–8826).

Xin, J., Tang, R., Yu, Y., & Lin, J. (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, G., Hao, J., Shen, L., Hu, H., Luo, Y., Lin, H., & Shen, J. (2023). Lgvit: Dynamic early exiting for accelerating vision transformer. In *Proceedings of the 31st ACM international conference on multimedia (MM '23)* (pp. 9103–9114). New York, NY, USA: Association for Computing Machinery.

Yang, L., Zheng, Z., Wang, J., Song, S., Huang, G., & Li, F. (2024). Adadet: An adaptive object detection system based on early-exit neural networks. *IEEE Transactions on Cognitive and Developmental Systems*, *16*(1), 332–345.

Ying, C., & Fragkiadaki, K. (2018). Depth-adaptive computational policies for efficient visual tracking. In *Energy minimization methods in computer vision and pattern recognition: 11th international conference, EMMCVPR 2017, Venice, Italy, October 30–November 1, 2017, revised selected papers 11* (pp. 109–122). Springer.

Zheng, B., Gou, B., Kil, J., Sun, H., & Su, Y. (2024). Gpt-4v (ision) is a generalist web agent, if grounded. In *International conference on machine learning* (pp. 61349–61385). PMLR.