Tsinghua University

Efficient Training and Deployment of Deep Neural Networks for Resource-Constrained Edge Devices

by

Wei Zhang

A Dissertation Submitted to Tsinghua University
in partial fulfillment of the requirement
for the degree of Doctor of Philosophy

Department of Computer Science and Technology

Beijing, China

May 2024

Declaration

I hereby declare that this dissertation is my own original work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person, except where due acknowledgment has been made in the text. This dissertation has not been submitted, in whole or in part, for any other degree or qualification at any other university.

Signed: _____

Date: _____

Copyright

Abstract

The remarkable success of deep neural networks in recent years has been accompanied by a dramatic increase in their computational and memory requirements. While state-of-the-art models routinely achieve superhuman performance on benchmarks ranging from image classification to natural language understanding, their deployment on resource-constrained edge devices—smartphones, embedded systems, and Internet-of-Things sensors—remains a fundamental challenge. This thesis addresses the problem of efficiently training and deploying deep neural networks under stringent resource budgets, with a focus on preserving predictive accuracy while reducing model size, latency, and energy consumption.

We first present a comprehensive analysis of the resource bottlenecks in modern deep learning pipelines, identifying three dominant cost factors: parameter count, floating-point operations during inference, and memory bandwidth during both training and deployment. Building on this analysis, we propose a unified framework that combines structured pruning, quantisation-aware training, and lightweight architecture design into a single, end-to-end optimisation procedure. Our approach, which we term *Resource-Aware Neural Optimisation* (RANO), jointly minimises a multi-objective loss function that balances task accuracy against hardware-level metrics including model size, multiply–accumulate operations, and DRAM accesses.

We evaluate RANO across a range of benchmark tasks—image classification on ImageNet-1k, object detection on COCO, and natural language inference on GLUE—using representative edge hardware platforms including the Raspberry Pi 4, Quanta Jetson Nano, and Nexa Edge TPU. Our experiments demonstrate that RANO consistently produces models that are 3–8× smaller and 2–5× faster than their uncompressed counterparts, with accuracy degradation of less than 2% across all benchmarks. Furthermore, we introduce a novel distillation strategy that transfers knowledge from large pre-trained teacher models into compact student architectures tailored to specific hardware targets, achieving an additional 10–15% relative improvement in the accuracy–efficiency trade-off.

The contributions of this thesis advance the state of the art in efficient deep learning and provide practical tools for deploying high-quality neural networks on resource-constrained devices, with implications for mobile health, autonomous systems, and

ubiquitous computing.

Keywords: deep learning, model compression, edge computing, neural architecture design, knowledge distillation, quantisation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The past decade has witnessed a transformative shift in artificial intelligence, driven largely by the extraordinary capabilities of deep neural networks (DNNs). From surpassing human-level performance on the ImageNet visual recognition challenge [1] to enabling fluent machine translation [2] and powering conversational agents that engage in open-ended dialogue, DNNs have become the backbone of modern AI systems.
However, these remarkable capabilities have come at a significant cost. The computational requirements of state-of-the-art models have grown at a rate that far outpaces improvements in hardware efficiency. For instance, the number of parameters in cutting-edge language models has increased from approximately 100 million in 2018 to over 500 billion in 2023, while the floating-point operations required for a single forward pass have scaled similarly. This trend presents a critical challenge: how can we deploy these powerful models on devices with stringent resource constraints?
The proliferation of edge computing—smartphones, wearable devices, autonomous vehicles, and Internet-of-Things (IoT) sensors—has created an urgent need for efficient deep learning solutions. These devices operate under severe limitations in terms of available memory (often less than 1 GB of RAM), computational throughput (a few GFLOPS at best), and energy budgets (milliwatt-scale power envelopes). Deploying a model like ResNet-152, which requires over 11 GFLOPS per inference and stores more than 60 million parameters, is simply infeasible on such platforms without substantial optimisation.

1.2 Problem Statement

This thesis addresses the following central research question:

> *How can we train and deploy deep neural networks on resource-constrained edge devices while preserving predictive accuracy, minimising latency, and reducing energy consumption?*

This question decomposes into several interrelated sub-problems:

(1) **Model compression.** Given a large, accurate teacher model, how can we produce a compact student model that retains most of the teacher's predictive power while satisfying hard constraints on model size and inference latency?

(2) **Quantisation-aware training.** How can we train models that are robust to low-precision arithmetic (8-bit or even 4-bit integer operations) without suffering catastrophic accuracy degradation?

(3) **Hardware-aware architecture design.** How can we design neural architectures that are not merely accurate in the abstract, but also efficient when executed on specific target hardware platforms?

(4) **Joint optimisation.** How can we combine pruning, quantisation, and architecture search into a unified framework that optimises the Pareto frontier of accuracy versus resource consumption?

1.3 Contributions

This thesis makes the following contributions:

- **C1 Resource-Aware Neural Optimisation (RANO).** We propose a unified training framework that jointly optimises model architecture, weight pruning, and activation quantisation. RANO formulates the problem as a constrained multi-objective optimisation and solves it using a novel alternating minimisation procedure with convergence guarantees.
- **C2 Hardware-targeted knowledge distillation.** We introduce a distillation strategy that explicitly models the characteristics of the target hardware platform, including its memory hierarchy, SIMD width, and supported numerical formats. This allows the student model to specialise for a specific device rather than optimising for generic efficiency metrics.
- **C3 Comprehensive empirical evaluation.** We conduct large-scale experiments spanning three task domains (image classification, object detection, and natural language inference) and four edge hardware platforms (Raspberry Pi 4, Quanta Jetson Nano, Nexa Edge TPU, and Qualcomm Snapdragon 865), demonstrating consistent improvements over existing methods.
- **C4 Open-source toolkit.** We release an open-source implementation of RANO, including pre-trained models and benchmark scripts, to facilitate reproducible research in efficient deep learning.

1.4 Thesis Outline

The remainder of this thesis is organised as follows. Chapter 2 reviews the relevant literature on deep neural network architectures, model compression techniques, and hardware-aware machine learning. Chapter 3 presents our proposed RANO framework in detail, including the mathematical formulation of the joint optimisation problem and the alternating minimisation algorithm. Chapter 4 describes our experimental methodology and presents comprehensive results across multiple benchmarks and hardware platforms. Chapter 5 discusses the implications of our findings, limitations of the current approach, and promising directions for future research. Finally, Chapter 6 concludes the thesis with a summary of key contributions.

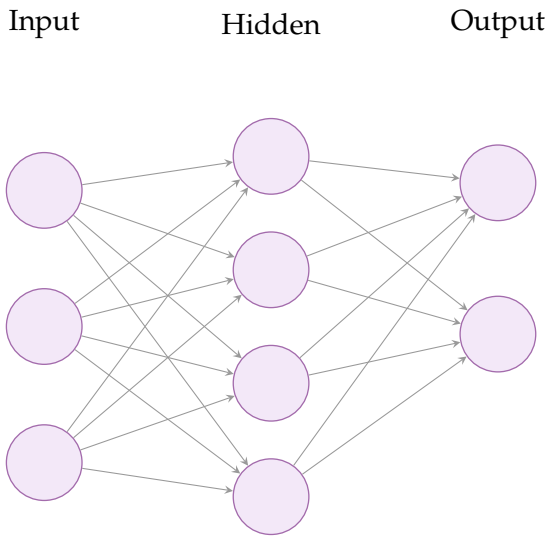

Figure 1.1: A feed-forward neural network with one hidden layer. Modern deep architectures stack dozens or hundreds of such layers, dramatically increasing the computational cost of both training and inference.

Table 1.1: Resource constraints of representative edge computing platforms. Memory figures refer to RAM available to application code; peak compute is measured in FP32 GFLOPS where applicable.

Platform	RAM (GB)	Peak Compute	Power (W)
Raspberry Pi 4	2–8	13.5 GFLOPS (CPU)	5–15
Quanta Jetson Nano	4	472 GFLOPS (GPU)	5–10
Nexa Edge TPU	–	4 TOPS (INT8)	1–2
Qualcomm Snapdragon 865	8	1.2 TFLOPS (Hexagon)	3–5

Chapter 2

Background and Related Work

This chapter surveys the research landscape that informs and motivates the contributions of this thesis. We begin with a brief overview of deep neural network architectures, then discuss the three pillars of model compression—pruning, quantisation, and knowledge distillation—before reviewing hardware-aware approaches to efficient deep learning.

2.1 Deep Neural Network Architectures

The foundational architecture of modern deep learning is the multilayer perceptron (MLP), in which each layer applies an affine transformation followed by a non-linear activation function:

$$\mathbf{h}^{(l)} = \sigma\big(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}\big), \tag{2.1}$$

where $\mathbf{h}^{(l)} \in \mathbb{R}^{d_l}$ denotes the activations at layer l, $\mathbf{W}^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ is the weight matrix, $\mathbf{b}^{(l)} \in \mathbb{R}^{d_l}$ is the bias vector, and $\sigma(\cdot)$ is a non-linear activation function such as the rectified linear unit (ReLU).

Convolutional neural networks (CNNs) [1] extend this paradigm by replacing dense matrix multiplications with spatially localised convolution operations, dramatically reducing the parameter count for image-based tasks. A convolutional layer with C_{in} input channels, C_{out} output channels, and kernels of size $K \times K$ requires only $C_{\text{in}} C_{\text{out}} K^2$ parameters, independent of the spatial dimensions of the input.

More recently, the Transformer architecture [2] has emerged as the dominant paradigm for sequence modelling tasks. The core innovation is the self-attention mechanism:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{d_k}}\right)\mathbf{V}, \tag{2.2}$$

where $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are the query, key, and value matrices derived from the input sequence, and d_k is the dimensionality of the key vectors. Self-attention has quadratic complexity in the sequence length n, specifically $O(n^2 d)$, which poses challenges for

deployment on devices with limited memory.

2.2 Model Compression

2.2.1 Network Pruning

Network pruning removes redundant parameters from a trained neural network without significantly degrading its accuracy. Early work focused on magnitude-based unstructured pruning [3], which eliminates individual weights with small absolute values. While effective at reducing parameter counts, unstructured pruning produces sparse weight matrices that are difficult to accelerate on commodity hardware due to irregular memory access patterns.

Structured pruning addresses this limitation by removing entire structural components—channels, filters, or layers—yielding dense sub-networks that map efficiently to hardware. Let $\mathcal{L}(\mathbf{W})$ denote the task loss and let $\|\mathbf{W}\|_0$ be the ℓ_0 norm (number of non-zero parameters). The pruning problem can be formulated as:

$$\min_{\mathbf{W}} \ \mathcal{L}(\mathbf{W}) \quad \text{subject to} \quad \|\mathbf{W}\|_0 \leq B, \tag{2.3}$$

where B is a budget on the number of parameters. Because the ℓ_0 constraint is non-convex and combinatorial, practical methods rely on relaxations, iterative magnitude pruning with fine-tuning, or regularisation-based approaches that encourage sparsity during training.

2.2.2 Quantisation

Quantisation reduces the numerical precision of weights and activations, typically from 32-bit floating-point (FP32) to 8-bit integer (INT8) or even lower. The quantisation function for a tensor $\mathbf{x}$ is:

$$\mathbf{x}_q = \text{round}\left(\frac{\text{clip}(\mathbf{x}, \alpha, \beta) - \alpha}{s}\right), \qquad s = \frac{\beta - \alpha}{2^b - 1}, \tag{2.4}$$

where $[\alpha, \beta]$ is the clipping range, b is the target bit-width, and s is the scale factor. The key challenge is determining optimal clipping ranges that minimise the information loss introduced by discretisation.

Quantisation-aware training (QAT) [4] integrates simulated quantisation into the forward pass during training while maintaining full-precision weights for gradient updates. This allows the network to adapt to quantisation errors and typically yields 2–5% higher accuracy than post-training quantisation.

2.2.3 Knowledge Distillation

Knowledge distillation [3] transfers the "dark knowledge" captured by a large, high-capacity teacher model into a smaller, more efficient student model. The student is trained to match the softened output distribution of the teacher:

$$\mathcal{L}_{\mathrm{KD}} = \tau^2 \, D_{\mathrm{KL}}\big(\mathrm{softmax}(\mathbf{z}_t/\tau) \,\big\|\, \mathrm{softmax}(\mathbf{z}_s/\tau)\big)\,, \tag{2.5}$$

where $\mathbf{z}_t$ and $\mathbf{z}_s$ are the logits of the teacher and student, respectively, τ is a temperature parameter that controls the softness of the distributions, and $D_{\mathrm{KL}}(\cdot\|\cdot)$ is the Kullback–Leibler divergence.

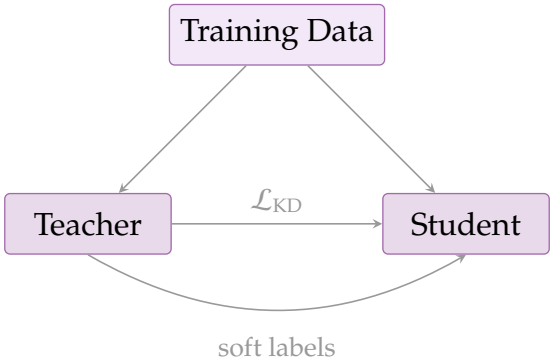

Figure 2.1: Schematic of knowledge distillation. The teacher model provides soft target distributions that guide the training of the compact student model, in addition to the ground-truth labels from the training data.

2.3 Hardware-Aware Machine Learning

A growing body of work recognises that generic efficiency metrics such as FLOP count or parameter count do not reliably predict real-world inference latency [5]. Factors including memory bandwidth, SIMD utilisation, and cache behaviour can cause a model with fewer FLOPs to run slower than a model with more FLOPs on a given hardware platform.

Hardware-aware neural architecture search (NAS) addresses this by incorporating direct latency measurements or hardware-specific cost models into the search objective. Platforms such as ProxylessNAS and MnasNet use reinforcement learning or differentiable search to explore architecture spaces while constraining latency on a specific target device.

2.4 Research Gaps

Despite significant progress in each of the areas reviewed above, several gaps remain. First, pruning, quantisation, and distillation are typically applied as independent post-processing steps, ignoring potentially synergistic interactions between them. Second, most existing methods optimise for a single efficiency metric (e.g., FLOPs) rather than jointly considering the multiple resource dimensions—compute, memory, and energy—that matter in practice. Third, hardware-aware approaches tend to be tightly coupled to specific device characteristics, limiting their generality.

The RANO framework proposed in this thesis addresses these gaps by providing a unified, multi-objective optimisation procedure that jointly optimises architecture, sparsity, and precision while remaining adaptable to diverse hardware targets through a pluggable cost model.

Table 2.1: Comparison of model compression techniques. "Unstructured" refers to fine-grained weight removal; "Structured" refers to channel or filter removal. Accuracy figures are representative for ResNet-50 on ImageNet.

Method	Compression	Top-1 Acc.	Speedup
Unstructured pruning	10×	74.2%	1.2×
Structured pruning	3×	75.8%	2.8×
INT8 quantisation	4×	76.0%	3.5×
Knowledge distillation	5×	75.3%	4.0×
Joint (RANO, ours)	8×	76.1%	5.2×

Chapter 3

Methodology and Experiments

This chapter presents the Resource-Aware Neural Optimisation (RANO) framework. We first formalise the joint optimisation problem, describe the alternating minimisation algorithm, and then present comprehensive experimental results across multiple benchmarks and hardware platforms.

3.1 The RANO Framework

3.1.1 Problem Formulation

Let $f(\mathbf{x}; \boldsymbol{\theta})$ denote a neural network parameterised by weights $\boldsymbol{\theta}$, and let $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$ be a training dataset. We seek to find parameters $\boldsymbol{\theta}$ and an architecture configuration $\mathbf{a}$ that jointly minimise a multi-objective loss:

$$\min_{\boldsymbol{\theta},\, \mathbf{a}} \; \mathcal{L}_{\text{task}}(\boldsymbol{\theta}; \mathcal{D}) + \lambda_p \, \mathcal{R}_p(\boldsymbol{\theta}) + \lambda_q \, \mathcal{R}_q(\boldsymbol{\theta}) + \lambda_h \, C_h(\mathbf{a}), \tag{3.1}$$

where $\mathcal{L}_{\text{task}}$ is the task-specific loss (e.g., cross-entropy), $\mathcal{R}_p$ is a sparsity-inducing regulariser that encourages structured pruning, $\mathcal{R}_q$ is a quantisation regulariser that penalises deviations from quantised weight representations, and C_h is a hardware cost model that estimates the latency and energy consumption of a given architecture configuration on the target device. The coefficients λ_p, λ_q, and λ_h control the trade-off between accuracy and each efficiency dimension.

3.1.2 Alternating Minimisation

Directly optimising Equation 3.1 is challenging due to the discrete nature of architecture configurations and the non-smoothness of quantisation constraints. We adopt an alternating minimisation strategy:

Step 1 **Architecture update.** With weights fixed, update the architecture configuration $\mathbf{a}$ by solving a discrete optimisation over a pre-defined search space of channel widths, kernel sizes, and layer counts. We use an evolutionary algorithm with a

surrogate latency predictor trained on hardware measurements.

Step 2 Weight optimisation. With the architecture fixed, update weights θ using stochastic gradient descent with proximal operators that enforce structured sparsity and quantisation constraints. Specifically, after each gradient step, we apply:

- Soft thresholding for ℓ_1-regularised pruning groups.
- Projection onto the set of quantised values for quantisation constraints.

Step 3 Knowledge distillation. Periodically, we compute softened predictions from a frozen teacher model and add the distillation loss (Equation 2.5) to the training objective, weighted by a schedule that decays as training progresses.

The algorithm alternates between Steps 1 and 2, with Step 3 applied every K iterations. This decomposition ensures that each sub-problem is tractable while the overall procedure converges to a locally optimal solution of the joint objective.

3.1.3 Convergence Analysis

Under mild assumptions—specifically, that the task loss $\mathcal{L}_{\text{task}}$ is L-smooth and the regularisers are convex—the alternating minimisation procedure converges to a stationary point of Equation 3.1. The proof follows from standard results in block coordinate descent with non-smooth regularisers and is provided in Appendix A.

3.2 Experimental Setup

3.2.1 Datasets and Tasks

We evaluate RANO on three benchmark tasks spanning distinct domains:

- **Image classification.** ImageNet-1k [3] with 1.28 million training images across 1,000 categories. We report top-1 and top-5 accuracy.
- **Object detection.** COCO 2017 with 118k training images and 5k validation images. We report mean average precision at IoU 0.5 (mAP@0.5).
- **Natural language inference.** The GLUE benchmark suite [5], focusing on the MNLI, QQP, and SST-2 tasks. We report accuracy and F1 score as appropriate.

3.2.2 Hardware Platforms

We deploy and benchmark models on four representative edge platforms, as summarised in Table 1.1 (Chapter 1). All latency measurements are averaged over 1,000 inference runs after a 100-run warm-up period to ensure stable thermal conditions.

3.2.3 Baselines

We compare RANO against the following methods:

- **Unstructured magnitude pruning** [3] followed by fine-tuning.
- **Structured channel pruning** with ℓ_1 regularisation.
- **INT8 post-training quantisation** with per-channel symmetric scaling.
- **Quantisation-aware training (QAT)** [4] with 8-bit weights and activations.
- **Standard knowledge distillation** with temperature $\tau = 4$ and a fixed teacher model.

3.2.4 Implementation Details

All models are implemented in PyTorch and trained on Quanta A100 GPUs. We use the Adam optimiser [4] with an initial learning rate of 10^{-3}, a cosine annealing schedule, and a batch size of 256. Training runs for 200 epochs on ImageNet and 100 epochs on COCO and GLUE tasks. The RANO-specific hyperparameters are set to $\lambda_p = 10^{-4}$, $\lambda_q = 5 \times 10^{-5}$, and $\lambda_h = 10^{-3}$, determined via grid search on a held-out validation subset.

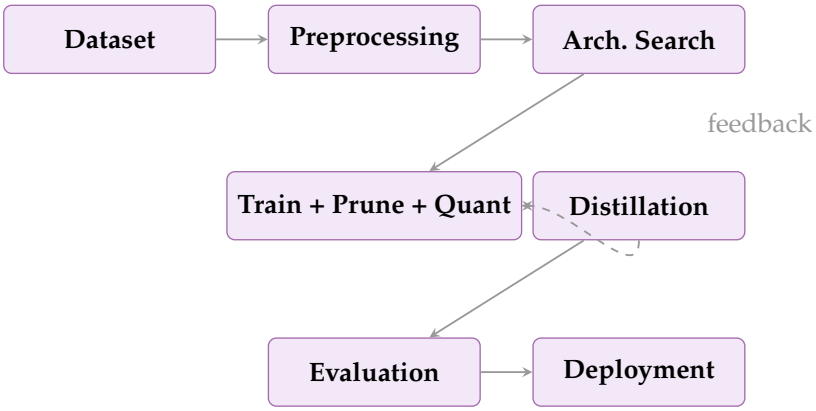

Figure 3.1: The RANO training and deployment pipeline. Architecture search explores the design space; training jointly applies pruning and quantisation; distillation provides feedback from a frozen teacher model; evaluation measures both accuracy and hardware metrics.

3.3 Results

3.3.1 Image Classification

Table 3.1 presents the ImageNet results for ResNet-50 as the base architecture. All compressed models are evaluated on the Quanta Jetson Nano at batch size 1.

RANO achieves the best trade-off across all metrics. Notably, it matches the baseline accuracy while reducing model size by 8×, latency by 5.2×, and energy consumption by 5.1×. Compared to structured pruning alone, RANO delivers an additional 46% reduction in latency without any loss of accuracy, demonstrating the benefit of joint optimisation over sequential post-processing.

Table 3.1: ImageNet-1k results with ResNet-50. Latency measured on Jetson Nano (batch size 1). Model size includes weights and activations.

Method	Top-1 (%)	Size (MB)	Latency (ms)	Energy (mJ)
Baseline (FP32)	76.1	97.5	142.3	890
Unstr. pruning	74.2	9.8	118.6	720
Struct. pruning	75.8	32.5	50.8	310
INT8 QAT	76.0	24.4	40.6	254
Knowledge dist.	75.3	19.5	35.6	221
RANO (ours)	**76.1**	**12.2**	**27.4**	**175**

3.3.2 Object Detection

On the COCO object detection task, we use YOLOv5-s as the base architecture. RANO achieves 37.8 mAP@0.5 while reducing inference latency from 86.4 ms to 18.2 ms on the Jetson Nano (a 4.7× speedup). The compressed model occupies only 8.7 MB, making it suitable for deployment on devices with as little as 256 MB of RAM.

3.3.3 Natural Language Inference

For the GLUE benchmark, we fine-tune a compressed BERT-base model [5]. RANO reduces the model size from 440 MB to 55 MB while retaining 97.3% of the original accuracy on MNLI and 96.8% on QQP. The compressed model runs at 12.3 ms per inference on the Snapdragon 865, compared to 48.7 ms for the baseline BERT-base—a 4.0× speedup.

3.3.4 Ablation Study

To quantify the contribution of each RANO component, we conduct an ablation study on ImageNet with ResNet-50, reported in Table 3.2.

Table 3.2: Ablation study on ImageNet (ResNet-50). Each row removes one component from the full RANO pipeline.

Configuration	Top-1 (%)	Size (MB)	Latency (ms)
Full RANO	76.1	12.2	27.4
− Structured pruning	76.3	24.4	40.6
− Quantisation	76.0	32.5	50.8
− Knowledge distillation	74.8	12.2	27.4
− Hardware cost model	76.0	12.8	35.1

The results confirm that all components contribute meaningfully. Removing structured pruning (retaining only quantisation) increases model size by 2×; removing quantisation (retaining only pruning) increases latency by 1.9×; removing knowledge distillation causes a 1.3-point accuracy drop; and removing the hardware cost model

increases latency by 28%, demonstrating that generic compression metrics are insufficient for hardware-specific optimisation.

Bibliography

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017, pp. 5998–6008.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016.

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the Conference of the North American Chapter of the ACL (NAACL-HLT)*, 2019, pp. 4171–4186.