Abstract of "**Optimizing Machine Learning Models through Parameter-Efficiency**" by Ahmed Agiza, Ph.D., Brown University, February 2025.

Machine learning (ML) technologies have experienced unprecedented advancements, catalyzing theoretical breakthroughs and enabling wide-ranging applications across various domains. However, as these systems become more complex and are expected to perform a wider range of tasks, the challenges associated with efficient training, fine-tuning, and deployment become increasingly prominent. One key strategy to address these challenges is parameter efficiency, which involves minimizing the number of parameters that need adjustment during training or inference, thus reducing computational resources while maintaining or improving performance. This dissertation examines various approaches to enhance parameter efficiency in ML models through exploring novel architectures and unconventional computing paradigms. We present three main contributions aimed at enhancing parameter efficiency in ML models while maintaining or improving performance. The first contribution introduces MTLoRA, a fine-tuning framework that leverages low-rank adaptation matrices for parameter-efficient training of MTL models. By effectively disentangling the parameter space for diverse tasks, MTLoRA minimizes the number of parameters that need to be adjusted, training only a small fraction of the model while maintaining high adaptability and performance. Building on the principles of parameter-efficient architectures, the second contribution, CRLoMTL, addresses the challenges of multi-objective optimization in MTL related to sharing parameters between different tasks. CRLoMTL employs parameter-efficient methods to tackle the issue of conflicting gradients across the shared backbone. By utilizing low-rank matrix adaptations to handle gradient conflicts, CRLoMTL integrates a multi-pass training strategy to effectively capture task interactions and systematically minimize inter-task conflicts in the shared parameters, leading to improved convergence and overall model performance. We further extend these ideas by improving the parameter-efficiency of MTL models through an adaptive framework for input-dependent dynamic sparsification in multi-task learning. Our framework employs a lightweight policy network to recognize unnecessary computations based on input complexity, further optimizing model adaptability and efficiency. Moving beyond traditional silicon-based systems, the third contribution explores unconventional platforms to build parameter-efficient systems. Unconventional computing, such as chemical computing, offers unique computational paradigms with potential inherent parameter efficiency through analog encoding, parallelism, and bio-compatibility. We present a molecular computation system that uses acid-base reactions to perform calculations, efficiently encoding information in analog chemical form. By leveraging the natural complementarity of acids and bases, our approach demonstrates the feasibility of constructing neural network classifiers in a chemical medium. Our approach efficiently encodes information in analog chemical form, relying on the natural complementarity of acids and bases, demonstrating the feasibility of constructing neural network classifiers in a chemical medium. The model is distinguished by two key advantages: inherent parallelism and signal encoding efficiency. The model leverages simultaneous chemical reactions to model the parallel processing of the neural network computation. Furthermore, our model achieves parameter efficiency through analog encoding of signals as acid-base concentration levels, providing a more granular and expansive range of values for each computational unit compared to traditional binary encoding. Furthermore,

we extend our chemical computation platform to leverage enzymatic reactions to model neural networks and digital gates through pH modulation, showcasing the potential for diversifying the computational strategies for building parameter-efficient architectures.

Optimizing Machine Learning Models through Parameter-Efficiency

by
Ahmed Agiza

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island
February 2025

© Copyright 2025 by Ahmed Agiza

This dissertation by Ahmed Agiza is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date _____ _____
Sherief Reda, Director

Recommended to the Graduate Council

Date _____ _____
Yu Cheng, Reader

Date _____ _____
Nikos Vasilakis, Reader

Date _____ _____
Eunsuk Kim, Reader

Approved by the Graduate Council

Date _____ _____
Thomas A. Lewis
Dean of the Graduate School

Vitae

Ahmed was born and raised in Egypt. He received his B.Sc degree in Computer Engineering with double minors in Mathematics and Business Administration from The American University in Cairo in 2017. In 2023, he completed his M.Sc in Computer Science at Brown University. His primary research areas include efficient machine learning, multi-task learning, and physical synthesis optimizations.

ahmed_agiza@brown.edu
https://agiza.me
Brown University, RI, USA

Selected Publications:

1. Agiza, Ahmed, Marina Neseem, Sherief Reda. "MTLoRA: A Low-Rank Adaptation Approach for Efficient Multi-Task Learning." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2024.

2. Neseem, Marina, Ahmed Agiza, and Sherief Reda. "AdaMTL: Adaptive Input-dependent Inference for Efficient Multi-Task Learning." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2023.

3. Agiza, Ahmed A., Kady Oakley, Jacob K. Rosenstein, Brenda M. Rubenstein, Eunsuk Kim, Marc Riedel, and Sherief Reda. "Digital circuits and neural networks based on acid-base chemistry implemented by robotic fluid handling." Nature Communications 14.1 (2023): 496.

4. Agiza, Ahmed, Stephen Marriott, Jacob K. Rosenstein, Eunsuk Kim, and Sherief Reda. "pH-Controlled Enzymatic Computing for Digital Circuits and Neural Networks." Physical Chemistry Chemical Physics 26.31 (2024): 20898-20907.

5. Agiza, Ahmed, Mohamed Mostagir, and Sherief Reda. "PoliTune: Analyzing the Impact of Data Selection and Fine-Tuning on Economic and Political Biases in Large Language Models." Proceedings of the 2024 AAAI/ACM Conference on AI, Ethics, and Society. 2024.

6. Agiza, Ahmed, Rajarshi Roy, Teodor Dumitru Ene, Saad Godil, Sherief Reda, and Bryan Catanzaro. "GraPhSyM: Graph Physical Synthesis Model." 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD). IEEE, 2023.

7. Agiza, Ahmed, and Sherief Reda. "OpenPhySyn: An Open-Source Physical Synthesis Optimization Toolkit." 2020 Workshop on Open-Source EDA Technology (WOSET). 2020.

8. Agiza, Ahmed, and Sherief Reda. "CRLoMTL: Conflict-Resolution through Low-rank Matrices for Multi-Task Learning." AAAI Conference on Artificial Intelligence 2025, Under Review.

Acknowledgements

This journey was made possible through the support and contributions of many individuals. I would like to express my sincere appreciation to those who have contributed to the development of this research.

I would like to express my deepest gratitude to my advisor, Prof. Sherief Reda, for his invaluable guidance, support, and mentorship throughout my doctoral journey. I would also like to thank my thesis committee, Prof. Yu Cheng, Prof. Nikos Vasilakis, and Prof. Eunsuk Kim, for their insightful feedback and contributions to my thesis.

I extend my heartfelt appreciation to my collaborators, without whom this research would not have been possible. Marina Neseem, whose close collaboration has been instrumental in shaping this work. I am also grateful to all my collaborators: Kady Oakley, Prof. Jacob K Rosenstein, Prof. Brenda M Rubenstein, Prof. Eunsuk Kim, Prof. Marc Riedel, Stephen Marriott, Prof. Mohamed Mostagir, Rajarshi Roy, Teodor Dumitru Ene, Saad Godil, Bryan Catanzaro, and once again, my advisor Prof. Sherief Reda.

To my wife, Rawan - words cannot express my gratitude for her unwavering support, patience, and love throughout this challenging journey. She has been a constant source of strength and motivation.

A special thanks to my best friend, Mohammed Rafaat, for being my biggest supporter overseas and valuable source of encouragement. I am deeply indebted to Abdulrahman Hosny, who has been an exceptional peer advisor, mentor, and supporter throughout my journey.

My sincere appreciation goes to Prof. Shalan, whose guidance and support over the past decade have been invaluable in shaping my academic and professional growth.

I would like to acknowledge my fellow colleagues at Prof. Reda's SCALE lab at Brown University: Sofiane Chetoui, Abdulrahman Hosny, Marina Neseem, Jingxiao Ma, Manar Abdelatty, Mahdi Boulila, and Morteza Nabavinejad. Their camaraderie and intellectual discussions have enriched my research experience. Finally, I want to express my love and gratitude to my family, especially my siblings: Youssef, Nermeen, Hagar, Abdulrahman, and Nour. Their love, support, and encouragement have been invaluable throughout this journey.

To all those mentioned and the many others who have contributed to my journey, I offer my sincerest thanks.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and Context

In recent years, machine learning has experienced substantial growth and innovation, influencing numerous domains and contributing to advancements in technology and research. ML models have demonstrated capabilities in problem-solving, pattern recognition in data, and enhancing decision-making processes across sectors such as healthcare, finance, autonomous systems, and natural language processing. This progression has been facilitated by factors including increased data availability, advancements in computational power, and the development of more sophisticated algorithms and architectures. As the capabilities of ML systems expand, the expectations placed upon them also increase. Modern ML models are increasingly required to perform a diverse range of tasks, often simultaneously, and to adapt to new domains and challenges. This trend towards more versatile AI systems has led to the development of larger and more complex models. Recent iterations of language and vision models, for example, contain a large number of parameters, allowing them to address a wide range of tasks. However, the growth in model size and complexity has introduced significant challenges. State-of-the-art models often require substantial computational resources for training and inference. This increase in scale has implications for the cost and time associated with model development, as well as the environmental impact of AI research and deployment [140]. For example, training large models can result in significant carbon emissions, raising concerns about the sustainability of AI development and highlighting the need for more efficient approaches [117]. Furthermore, the deployment of large models on resource-constrained devices, such as smartphones or IoT devices, presents a considerable challenge [92]. The disparity between the computational requirements of state-of-the-art models and the capabilities of edge devices has limited the widespread adoption of advanced ML technologies in many real-world scenarios. This limitation is particularly relevant in applications where real-time processing, privacy, or offline functionality is crucial. In response to these challenges, the ML research community has focused on developing more efficient architectures and training methodologies. The concept of parameter efficiency has emerged as an important area of study, aiming to reduce the number of parameters that require adjustment during training

or inference while maintaining or improving model performance [46, 69]. This approach addresses computational and resource constraints and has the potential to improve model generalization and reduce overfitting [17]. Techniques such as pruning, quantization, and knowledge distillation have shown the potential to reduce the model size and computational requirements while maintaining performance. Pruning involves removing unnecessary connections or neurons from a neural network. Quantization reduces the precision of the model's weights, allowing for more efficient computation and storage. Knowledge distillation transfers information from a larger model to a smaller, more efficient model, enabling the creation of compact models that retain much of the performance of their larger counterparts. Multi-task learning (MTL) has gained attention as a paradigm suited to addressing these efficiency challenges. By training a single model to perform multiple related tasks simultaneously, MTL leverages shared representations and knowledge transfer between tasks, potentially improving both efficiency and performance. This approach aligns with the goal of creating more versatile AI systems that can adapt to various tasks and domains without requiring separate models for each application. However, MTL also introduces its own set of challenges, particularly in balancing the learning objectives of different tasks and managing potential conflicts between them. The optimization of multi-task models is complex, as different tasks may have conflicting gradients or require different levels of attention to various parts of the shared model architecture. Addressing these challenges is necessary for realizing the potential of MTL in creating efficient and versatile ML systems. In parallel with these developments in traditional ML architectures, there is interest in exploring unconventional computing platforms that may offer advantages in terms of efficiency and scalability. Chemical and biological computing systems, for instance, have the potential to perform certain computations with lower energy requirements than traditional electronic systems. These alternative platforms could provide new avenues for building efficient AI systems. Biological computing, inspired by the information processing capabilities of living organisms, offers possibilities for creating energy-efficient and highly parallel computing systems [1]. Neural networks implemented using DNA molecules or protein interactions could potentially perform parallel computations with reduced energy consumption [122]. Similarly, quantum computing, while still in the early stages of development, suggests potential speedups for certain types of calculations that are relevant to machine learning tasks. Ongoing investigations explore approaches to enhance the efficiency and adaptability of machine learning models. This includes research into parameter-efficient architectures that can reduce the computational requirements of ML models while maintaining performance. Additionally, studies examine optimization techniques for multi-task learning, aiming to improve the balance between task-specific performance and shared knowledge representation. Finally, exploration of unconventional computing platforms, particularly chemical computing, is underway as a means to achieve efficiency in AI systems. The insights and methodologies emerging from these studies may inform the design of future ML systems, potentially enabling more efficient deployment in resource-constrained environments and facilitating more widespread adoption of AI across various domains.

1.1.1 Challenges in Machine Learning Optimization

As machine learning models evolve to tackle increasingly complex tasks, optimization challenges related to parameter efficiency have become more pronounced. These challenges encompass various aspects of model

design, training, and deployment, and their resolution is crucial for developing efficient and scalable machine learning systems. A fundamental optimization challenge is balancing model capacity with computational resource requirements. While larger models with more parameters often excel in complex tasks [80], they also demand substantial computational resources for training and inference. This tension is particularly evident in scenarios with limited computational resources, such as edge devices or real-time applications [56]. Hence, researchers are actively exploring methods to optimize model architectures, aiming to achieve high performance with fewer parameters through approaches like neural architecture search [42] and efficient network design [144]. The optimization of training procedures for parameter-efficient models introduces its own set of complexities because of the limited representation capacity. Related areas such as transfer learning and fine-tuning aim to leverage knowledge from pre-trained models to new tasks with limited data [165]. This led to significant advancements in parameter-efficient fine-tuning techniques. Methods such as adapter modules [68, 70, 49], sparse fine-tuning [12, 173], and Low-Rank Adaptation (LoRA) [69, 39] aim to adapt pre-trained models to new tasks by updating only a small subset of parameters or introducing a small number of new parameters. While these approaches show promise, optimizing them for both performance and efficiency remains an active area of research. Model compression techniques, including pruning [57] and quantization [74], offer potential solutions for reducing parameter count or precision. However, these methods often necessitate careful fine-tuning to maintain performance. The development of optimization strategies that can effectively identify and remove redundant parameters while preserving model accuracy continues to challenge researchers in the field. Multi-task learning scenarios amplify the complexity of optimizing for parameter efficiency. The essence of the issue lies in identifying shared representations that benefit multiple tasks while minimizing task-specific parameters [131]. Optimization algorithms must navigate the balance of potentially conflicting gradients from different tasks, a problem that grows increasingly intricate as the number of tasks expands [29]. Dynamic or conditional computation models, where only a subset of model parameters are active for any given input, introduce a distinct set of optimization hurdles [20]. While these models offer the potential for significant parameter efficiency, optimizing the routing or gating mechanisms that determine which parameters are active, as well as ensuring comprehensive training of all parameters, remains a complex task. Addressing these multifaceted optimization challenges is essential for improving parameter-efficiency in machine learning. Progress in this area demands interdisciplinary approaches, drawing from optimization theory, statistics, and domain-specific knowledge. As the need for more efficient and scalable machine learning models continues to grow, particularly for deployment in resource-constrained environments, these optimization challenges will likely remain at the forefront of machine learning research.

1.2 Thesis Contributions

This dissertation presents several contributions focusing on enhancing the efficiency and adaptability of models through architectural innovations and exploration of unconventional computing paradigms. The key contributions are as follows:

1. **Parameter-efficient multi-task learning architecture through low-rank adaptation and dynamic sparsification:**

Chapter 3 introduces MTLoRA (Multi-Task Low-Rank Adaptation), a fine-tuning framework designed to address parameter efficiency in multi-task learning (MTL) models. MTLoRA extends the concept of low-rank adaptations to the multi-task domain. The framework uses task-specific low-rank adaptation matrices to capture task-specific information while aiming to minimize interference between tasks. MTLoRA is designed to adapt pre-trained models to multiple tasks simultaneously, with a focus on minimizing the increase in overall parameter count. The chapter also explores the integration of input-dependent trainable policy networks for dynamic model sparsification. This mechanism aims to allocate computational resources based on the specific input and task. An empirical evaluation of MTLoRA is provided, examining its performance across various multi-task learning scenarios. The results are analyzed in terms of the framework's ability to balance performance with parameter efficiency.

2. **Leveraging low-rank matrices for parameter-efficient conflict resolution in multi-task learning:**

   Chapter 4 presents CRLoMTL (Conflict Resolution through Low-rank Multi-Task Learning), a framework that addresses the management of conflicts across shared parameters in multi-task learning. CRLoMTL employs low-rank matrix adaptations to handle gradient conflicts within the shared parameter space. The framework incorporates a multi-pass training strategy, which iteratively refines the model's handling of task interactions. Experimental results are presented to evaluate the effectiveness of CRLoMTL across different multi-task learning scenarios. The chapter discusses the implications of using low-rank matrix techniques in multi-task learning, considering both the advantages and limitations of this approach.

3. **Input-dependent dynamic sparsification for parameter-efficient multi-task learning:**

   Chapter 5 introduces AdaMTL, an adaptive framework that enhances parameter efficiency in multi-task learning models through input-dependent dynamic sparsification. AdaMTL builds upon the concepts of efficient multi-task learning by introducing a mechanism to dynamically adjust model complexity based on input characteristics. The framework incorporates a lightweight, task-aware policy network that learns to recognize unnecessary computations in the multi-task model. This policy network generates execution strategies at runtime, deciding which parts of the model to activate based on the input complexity and target computational budget. AdaMTL introduces a multi-grained decision-making process, consisting of both block-level and token-level policy networks. This approach allows for fine-grained control over model activation, optimizing both accuracy and efficiency across multiple tasks. The chapter also presents a new training recipe for co-training the policy network alongside the multi-task learning model, including a task-aware alternating training strategy to effectively balance the needs of multiple tasks. We present extensive experimental results demonstrating the effectiveness of AdaMTL in reducing computational complexity while maintaining or improving accuracy compared to static multi-task models and single-task baselines.

4. **Designing unconventional analog computing platform for neural networks based on acid-base chemistry:**

   Chapter 6 explores the potential of acid-base chemistry as a substrate for implementing parameter-efficient neural networks and digital circuits. This approach investigates the use of chemical reactions

as computational primitives for constructing neural network classifiers with high parameter density. The research examines how the complementarity of acids and bases can be used to encode information in a continuous, analog manner. This analog representation allows for a more efficient encoding of parameters compared to traditional discrete digital systems. By leveraging the continuous nature of chemical concentrations, this approach potentially enables the representation of a wider range of values within a single parameter, effectively increasing the information density per parameter. The chapter provides an analysis of how basic neural network operations can be implemented using acid-base reactions, with a focus on achieving higher parameter efficiency. It demonstrates how the analog nature of chemical reactions can be exploited to perform computations with fewer discrete components compared to traditional digital implementations. This approach potentially allows for more compact and parameter-efficient representations of neural network architectures. The chapter also discusses the challenges and limitations of this approach, including issues of precision, scalability, and practical implementation in the context of parameter efficiency. By leveraging the inherent parallelism and continuous value representation of chemical systems, this unconventional computing approach offers a new perspective on designing novel parameter-efficient neural network architectures.

5. **Designing enzymatic analog computing framework for neural networks and digital circuits through pH modulation:**

   Chapter 7 extends the exploration of chemical computing for parameter-efficient models by investigating a framework for implementing neural networks and digital circuits using enzymatic reactions controlled through pH modulation. The research examines how enzymatic reactions sensitive to pH changes can be used to perform both analog (neural network) and digital (logic gate) computations through the use of pH as a control mechanism for activating or inhibiting specific enzymatic reactions. The chapter provides a demonstration and analysis for neural network operations and digital logic operations implemented using enzymatic reactions while discussing the challenges of scalability and precision in enzymatic computing. This work showcases the potential implications of enzymatic computing for designing a novel parameter-efficient paradigm.

These contributions examine various approaches to parameter-efficient machine learning, multi-task optimization, and unconventional computing platforms. They explore potential methods for designing efficient and adaptable artificial intelligence systems, considering both traditional and biologically-inspired computing paradigms.

Chapter 2

Background

2.1 Parameter Efficiency in Machine Learning

Parameter efficiency has emerged as a critical area of focus in machine learning research, driven by the need to develop models that are computationally efficient, scalable, and deployable across a wide range of devices and scenarios. As machine learning models have grown in size and complexity, particularly in deep learning, the number of parameters in state-of-the-art models has increased dramatically. While this increase has led to significant improvements in performance across various tasks, it has also introduced challenges related to computational resources, memory requirements, and energy consumption. The concept of parameter efficiency in machine learning refers to the ability of a model to achieve high performance with a minimal number of parameters. This efficiency is crucial for several reasons. First, models with fewer parameters generally require less computational power for both training and inference, making them more suitable for deployment on resource-constrained devices such as mobile phones or edge computing systems. Second, parameter-efficient models often exhibit better generalization capabilities, as they are less prone to overfitting. Third, reducing the number of parameters can lead to faster training times and lower memory requirements, enabling more rapid iteration in model development and easier deployment in production environments. Several approaches have been developed to improve parameter efficiency in machine learning models. One fundamental technique is pruning, which involves removing unnecessary connections or neurons from a neural network [57]. Pruning can be performed during training (dynamic pruning [94]]) or after training (static pruning[56]). Magnitude-based pruning [57], for example, removes connections with the smallest absolute weight values, assuming these contribute least to the model's output. More sophisticated pruning techniques consider the impact of removing connections on the overall network performance, leading to more optimal pruning decisions. Another approach to parameter efficiency is quantization, which reduces the precision of the model's weights and activations [74]. Instead of using 32-bit floating-point numbers, quantized models might use 8-bit integers or even binary values. This reduction in precision decreases the model size and leads to faster computation, especially on hardware optimized for low-precision arithmetic. While quantization can introduce some loss in accuracy, modern implementation techniques and fine-tuning

can often mitigate these effects. Low-rank factorization has emerged as a powerful technique for reducing the number of parameters in neural networks [69, 75]. This approach decomposes weight matrices into products of lower-rank matrices, effectively reducing the total number of parameters while preserving much of the model's expressive power. Techniques like Low-Rank Adaptation (LoRA) have shown particular promise in the context of fine-tuning large language models, allowing for efficient adaptation to new tasks with minimal additional parameters. Knowledge distillation is another method for achieving parameter efficiency, where a smaller "student" model is trained to mimic the behavior of a larger "teacher" model [63]. This approach allows the smaller model to benefit from the knowledge encoded in the larger model, often achieving comparable performance with significantly fewer parameters. Distillation has been particularly successful in areas like natural language processing, where very large pre-trained models can be distilled into more compact and efficient versions. Recent research has also explored the concept of conditional computation, where only a subset of a model's parameters are active for any given input [20]. This approach allows for the creation of large, expressive models that use a small fraction of their parameters for each inference, effectively combining the benefits of large model capacity with efficient computation. Techniques like mixture-of-experts and sparse transformers fall into this category. Parameter-efficient architectures have also been a focus of research, with models like EfficientNet [144] for computer vision and ALBERT for natural language processing [90] demonstrating how careful design choices can lead to models that achieve state-of-the-art performance with fewer parameters than their predecessors. The pursuit of parameter efficiency in machine learning is not just about reducing model size or computational requirements. It also intersects with other important areas of research, such as interpretability and robustness. Smaller, more efficient models are often easier to interpret and analyze, which is crucial for applications in sensitive domains like healthcare or finance. Moreover, parameter-efficient models may exhibit improved robustness to adversarial attacks and better generalization to out-of-distribution data. Parameter efficiency remains a key consideration in the development of new models and techniques. The challenge lies in balancing the trade-offs between model capacity, computational efficiency, and task performance. Future advancements in this area are likely to focus on developing more sophisticated pruning and quantization techniques, exploring novel parameter-efficient architectures, and leveraging insights from neuroscience and information theory to create models that more closely mimic the efficiency of biological neural networks.

2.1.1 Parameter Efficiency Through Multi-Task Learning and Its Challenges

As introduced earlier in this dissertation, Multi-Task Learning (MTL) has emerged as a promising approach in machine learning, offering a way to improve model efficiency and performance by leveraging shared information across related tasks [148, 172, 33]. Building upon this concept, we now explore in greater depth how MTL contributes to parameter efficiency and the unique challenges it presents. In contrast to traditional single-task learning, MTL involves training a single model to perform multiple related tasks simultaneously, potentially leading to more parameter-efficient architectures. The core principle of MTL aligns well with the goal of parameter efficiency. By sharing representations between related tasks, MTL allows models to use parameters more effectively, potentially reducing the overall number of parameters required compared

to separate models for each task. This shared learning can be particularly beneficial when computational resources are limited or when deploying models on resource-constrained devices. MTL offers several potential advantages in the context of parameter efficiency. Shared representations allow the model to make better use of available training data across all tasks, potentially reducing the amount of data needed for each individual task. This improved data efficiency can lead to more compact models. Additionally, MTL can act as a regularizer, reducing the risk of overfitting on individual tasks by forcing the model to find representations that are useful for multiple objectives. This can lead to more robust and generalizable models without increasing parameter count. In scenarios where computational resources or model size are constrained, MTL allows for more efficient use of model capacity by sharing parameters across tasks. Despite these advantages, MTL also presents several significant challenges, particularly in the context of maintaining parameter efficiency [148, 172, 33]. One primary challenge is task interference. While the goal is to achieve positive transfer between tasks, negative transfer can occur when learning on one task interferes with or degrades performance on another. This challenge is particularly relevant to parameter efficiency, as it may necessitate additional parameters or more complex architectures to mitigate interference. Balancing task importance presents another crucial challenge. In many MTL scenarios, not all tasks are equally important or equally difficult. Determining how to balance the contribution of each task to the overall learning objective is complex, especially when trying to maintain a compact, efficient model [83]. This balance is critical for ensuring that the shared parameters are utilized effectively across all tasks. Architectural design remains an ongoing area of research in MTL [148, 172]. Designing MTL architectures that effectively share information between tasks while allowing for task-specific specialization is challenging. The trade-off between shared and task-specific parameters is crucial for maintaining efficiency while ensuring good performance across all tasks. This challenge becomes more pronounced as the number of tasks increases, requiring careful consideration of how to structure the model to maximize parameter efficiency. Training MTL models can be more challenging than single-task models from an optimization perspective. Different tasks may have conflicting gradients, leading to difficulties in finding a set of parameters that performs well across all tasks. This can result in slower convergence or suboptimal solutions [29], potentially requiring more complex optimization strategies that may impact parameter efficiency. As the number of tasks increases, the complexity of MTL systems can grow significantly. Managing this complexity while maintaining parameter efficiency becomes increasingly challenging with large numbers of tasks. This scalability issue is particularly relevant when considering the deployment of MTL models in resource-constrained environments. In scenarios where tasks are learned sequentially or where new tasks are added to an existing MTL model, there is a risk of catastrophic forgetting. Addressing this challenge while maintaining a compact model is crucial for developing truly adaptive, parameter-efficient MTL systems. Recent advances in MTL have sought to address these challenges through various approaches, many of which focus on maintaining or improving parameter efficiency. Adaptive loss balancing techniques have been proposed to dynamically adjust the importance of different tasks during training [83], potentially allowing for more efficient use of shared parameters. Novel architectural designs, such as task-specific attention mechanisms [97] or adaptive parameter-sharing schemes [106], have shown promise in managing task interference and improving overall performance while maintaining parameter efficiency. Gradient manipulation techniques, like gradient normalization [29], have been explored to address optimization

difficulties in MTL without significantly increasing model complexity. The intersection of MTL with other areas of machine learning research, such as meta-learning and continual learning, has opened up new avenues for addressing challenges like task relatedness and catastrophic forgetting [45, 67, 115]. These approaches often focus on learning efficient, adaptable representations that can be quickly fine-tuned for new tasks with minimal additional parameters. MTL remains an important area of research with the potential to significantly impact the development of more efficient, versatile, and robust AI systems. The ongoing challenges in MTL drive innovation in model architecture and training techniques for building parameter-efficient ML models.

2.2 Unconventional Computing for Parameter-Efficient Encoding

Researchers are increasingly exploring unconventional computing platforms as potential solutions for computational efficiency and scalability challenges. These alternative approaches to computation offer the promise of overcoming some of the limitations inherent in traditional silicon-based electronic systems, particularly in terms of parameter encoding efficiency through analog encoding, information density, and novel forms of parallel information processing. Unconventional computing platforms [2, 1, 19, 3, 52] encompass a wide range of technologies and approaches that diverge from the standard von Neumann architecture prevalent in modern computers. These platforms often draw inspiration from biological systems, physical phenomena, or theoretical models of computation that offer unique advantages for efficient parameter representation and processing. For example, neuromorphic computing is category of unconventional computing, which aims to mimic the structure and function of biological neural networks in hardware. Neuromorphic systems typically use analog or mixed-signal circuits to emulate the behavior of neurons and synapses, potentially offering significant improvements in energy efficiency for neural network computations. Examples of neuromorphic hardware include IBM's TrueNorth chip [104] and Intel's Loihi processor [35]. These systems have shown promise in tasks such as pattern recognition and sensory processing, with potential applications in edge computing and IoT devices where energy efficiency is paramount. Quantum computing represents another frontier in unconventional computing platforms [134, 89, 127]. Quantum computers leverage the principles of quantum mechanics, such as superposition and entanglement, to perform certain types of calculations exponentially faster than classical computers. While large-scale, fault-tolerant quantum computers are still in development, current noisy intermediate-scale quantum (NISQ) devices [118] have shown potential in areas such as optimization, simulation of quantum systems, and certain machine learning tasks. Quantum machine learning, an emerging field at the intersection of quantum computing and machine learning, explores how quantum algorithms and hardware can enhance or accelerate machine learning processes. Optical computing is another area of active research in unconventional platforms [105]. By using light instead of electricity to perform computations, optical systems offer the potential for high-speed, low-power operation and massive parallelism. Recent advancements in photonic integrated circuits and optical neural networks have demonstrated the feasibility of implementing machine learning algorithms directly in the optical domain, potentially offering significant speedups for certain types of computations [137]. DNA computing and molecular computing represent a radically different approach to information processing, using biological molecules as computational elements [162, 23, 119]. These approaches leverage the massive parallelism inherent in molecular

interactions and the information density of DNA to perform computations. While still largely in the theoretical and experimental stages, DNA computing has shown potential for solving certain classes of problems, such as the traveling salesman problem, and could offer unique advantages in areas like medical diagnostics and drug discovery. Chemical computing systems, which use chemical reactions to perform computations, offer another avenue for unconventional information processing [4, 65]. These systems can leverage the inherent parallelism of chemical reactions and the ability to encode information in molecular structures. Recent work has explored the use of chemical systems for implementing neural network-like computations, offering potential advantages in terms of energy efficiency and novel forms of information encoding. Chemical computing systems, which use chemical reactions to perform computations, offer a particularly promising avenue for parameter-efficient information processing [4, 65]. These systems leverage the inherent parallelism of chemical reactions and the ability to encode information in molecular structures, potentially allowing for highly efficient parameter representations. The continuous nature of chemical concentrations enables analog encoding of information, which can lead to significantly higher parameter density compared to traditional digital systems. In chemical computing, a single parameter, represented by a chemical concentration, can encode a wide range of values, potentially reducing the total number of parameters needed to represent complex functions or neural networks. This analog encoding in chemical systems offers several advantages for parameter efficiency. The continuous value representation allows chemical concentrations to represent a continuum of values, potentially encoding more information per parameter than binary digital systems. Furthermore, multiple chemical species can interact in solution, allowing for the encoding of high-dimensional information in a compact space. The parallel nature of chemical reactions, occurring simultaneously throughout the solution, enables parallel computation and potentially more efficient use of parameters. Additionally, certain mathematical operations, like addition or multiplication, can be implemented naturally through chemical mixing or reaction kinetics, potentially reducing the number of parameters needed to perform these operations compared to digital implementations. The exploration of unconventional computing platforms for machine learning is driven by the need for more parameter-efficient and scalable systems. As traditional electronic systems approach physical limits in terms of miniaturization and parameter density, alternative computing paradigms offer potential pathways for continued improvement in the efficiency of machine learning models. While the development and adoption of unconventional computing platforms for machine learning face challenges such as scalability, integration with existing systems, and reliability, the potential benefits in terms of parameter efficiency and computational density make them an important area of research. As these technologies mature, they may offer new avenues for improving the efficiency and capabilities of machine learning systems, potentially enabling new applications through more parameter-efficient computational platforms.

Chapter 3

Low-Rank Adaptation for Parameter-Efficient Multi-Task Learning

3.1 Introduction

Building upon the foundations of MTL and the motivation for parameter efficiency discussed in the previous chapters, we now present our first major contribution to address the challenges of efficiently fine-tuning MTL models. General-purpose vision and language models, particularly those trained on large-scale datasets, show remarkable adaptability to a wide range of downstream tasks [100, 146]. However, individually fine-tuning all parameters of these models for every downstream task poses significant efficiency challenges. This approach becomes increasingly inefficient as the number of tasks grows, especially in environments constrained by computational resources. Therefore, there is a need to develop resource-efficient fine-tuning techniques [69, 101, 99, 70]. These methods aim to optimize training efficiency by limiting the number of trainable parameters, all while attempting to preserve or enhance task-specific fine-tuning. Most existing parameter-efficient adaptation methods are primarily tailored for single-task adaptation, and they may lose their effectiveness when applied to MTL scenarios. This is attributed to the inherent complexity of MTL, where the goal is to optimize the performance of a single model across a spectrum of tasks, introducing an additional layer of complexity. Moreover, focusing solely on individual task adaptation overlooks the potential benefits of cross-task knowledge sharing. Such knowledge sharing in an MTL context can significantly enhance the performance of each task [33, 172].

To realize multi-task adaptation using existing methods [99, 101], individual modules specific to the different tasks have to be added and adapted to one downstream task at a time as shown in Figure 3.2a. This approach enables customization and improvement for each task's unique needs, especially useful when tasks have unique characteristics or require specialized knowledge [132, 106]. However, this strategy incurs

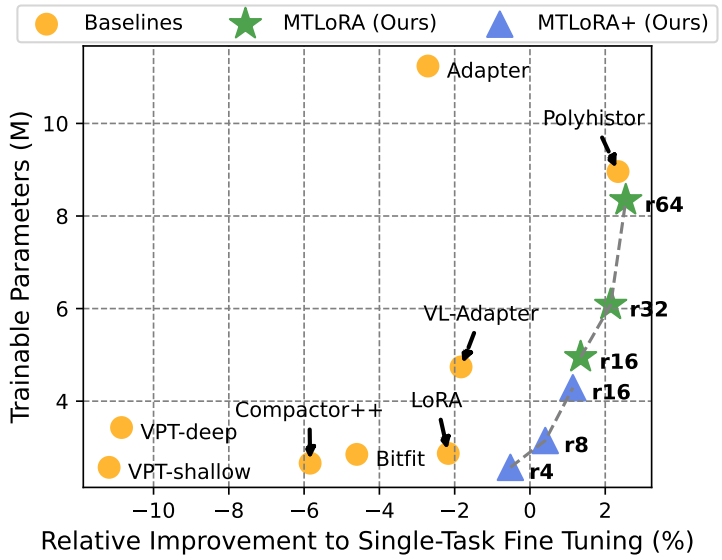


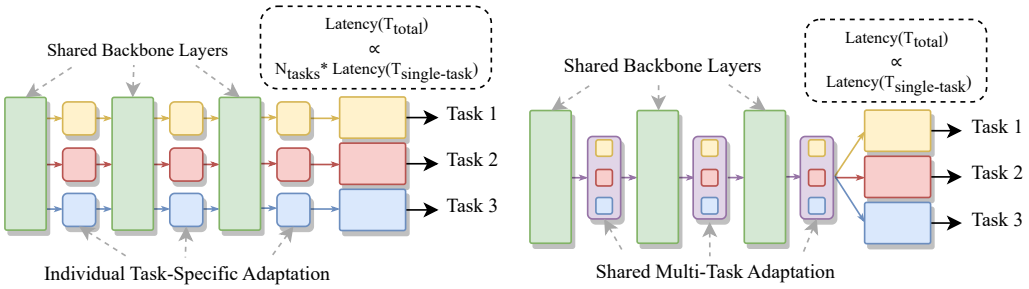
Figure 3.1: *MTLoRA* versus state-of-the-art parameter-efficient training approaches using Swin-Tiny vision transformer as a backbone. **r** represents the different ranks for the low-rank decomposition modules inside MTLoRA.

a significant drawback in terms of efficiency during both training and inference. Due to the task-specific nature of the fine-tuning, the model must be trained and inferred separately for each task. This results in a proportional increase in computational cost and time with the number of tasks. For instance, adapting a model to five distinct tasks would require five separate training passes. Similarly, during inference, the backbone needs to be executed five times, once for each task, leading to a linear escalation in inference and training duration as the task count increases.

Our research diverges from conventional parameter-efficient adaptation methods by concentrating on parameter-efficient training specifically for multi-task learning (MTL) architectures. In MTL models, a single shared backbone is trained to simultaneously extract feature representations for various downstream tasks [34, 97, 161]. MTL offers significant efficiency advantages since the shared backbone is executed only once, as shown in Figure 3.2b, leading to more resource-efficient training and inference processes where latency does not increase linearly with the number of tasks. Despite these advantages, the aspect of parameter-efficient training in MTL architectures remains largely unexplored.

The primary challenge in fine-tuning MTL models efficiently lies in addressing task conflicts during fine-tuning. These conflicts arise when different tasks have competing demands or induce divergent updates in the model. Hence, the focal point for many MTL architectures [77, 96] is to balance these conflicting updates. Consequently, a pivotal question emerges: *How can we efficiently adapt a single shared backbone to serve multiple tasks without sacrificing the individual performance of each task?*

In pursuit of this objective, we introduce *MTLoRA* - a novel framework designed for parameter-efficient fine-tuning of MTL models. *MTLoRA* addresses the challenges of fine-tuning a shared backbone to effectively serve multiple downstream tasks, particularly under the constraints of conflicting task requirements.



(a) Individual Task-Specific Adaptation (b) Shared Multi-Task Adaptation (Ours)

Figure 3.2: *(a) Individual Task Adaptation* results in parallel execution paths for each task, resulting in inference and training time that scales linearly with the number of tasks. On the other hand, (b) *Shared Multi-Task Adaptation* maintains inference and training time close to the single task model since only the decoders are executed separately.

This is accomplished through a strategic combination of *Task-Agnostic* and *Task-Specific* low-rank decomposition modules. By fine-tuning these modules, *MTLoRA* successfully untangles the parameter space involved in MTL fine-tuning, enabling the model to balance between learning shared features and those specific to individual tasks. Remarkably, *MTLoRA* demonstrates superior accuracy in downstream tasks compared to fully fine-tuning the entire MTL model while requiring the training of significantly fewer parameters. This enhanced performance is attributed to *MTLoRA*'s ability to facilitate positive knowledge sharing during fine-tuning, thereby improving the effectiveness of learning each downstream task. **Our contributions** can be summarized as follows:

- To the best of our knowledge, *MTLoRA* is the first to address the problem of parameter-efficient training of multi-task learning models. *MTLoRA* effectively balances between learning both shared and task-specific features during parameter-efficient fine-tuning.
- We design novel *Task-Agnostic* and *Task-Specific* low-rank adaptation modules leveraging them to adapt a shared vision-transformer backbone to multiple downstream dense prediction tasks.
- We observe that adding low-rank adaptation to the patch-merging layers in vision transformers, a practice not previously explored, significantly improves the accuracy-efficiency trade-off during fine-tuning MTL models. We highlight that observation by introducing *MTLoRA+*.
- We apply *MTLoRA* and *MTLoRA+* to a hierarchical-transformer-based MTL architecture. *MTLoRA* demonstrates superior accuracy in downstream tasks compared to fully fine-tuning the entire MTL model while training significantly less number of parameters. In addition, *MTLoRA* dominates state-of-the-art parameter-efficient training approaches, as shown in Figure 3.1.

The rest of the chapter is organized as follows. We review the related work in Section 3.2. Then, we introduce our MTLoRA framework in Section 5.3. Next, we show the setup and evaluation of MTLoRA in Section 4.4. Finally, we conclude in Section 5.5.

3.2 Related Work

Multi-task Learning: Multi-task learning is commonly used to learn various related tasks simultaneously [106, 102, 112]. The typical design of a multi-task architecture includes an encoder to distill feature representations from input frames and a set of task-specific decoders for generating predictions unique to each downstream task [172]. An important aspect to consider within these multi-task architectures is the mechanism of information sharing. The two main strategies are soft sharing and hard sharing. Soft parameter sharing involves each task having its own set of backbone parameters, with the primary objective of facilitating cross-task information exchange. On the other hand, hard parameter sharing employs a shared set of parameters within the backbone, with each task employing independent decoders for output generation [83, 135, 18]. Further classification of these architectures takes into account the stage at which task interactions occur - leading to the categorization into encoder-focused and decoder-focused frameworks [161]. Encoder-focused architectures centralize information exchange within the encoder stage [149, 98], whereas, in decoder-focused architectures, tasks exchange information during the decoding stage. Notably, some models adopt a more integrative approach, allowing for cross-task information sharing to occur at encoder and decoder stages [106].

Parameter-Efficient Training for Single-Task Models: Parameter-efficient training (PEFT) has become increasingly important, especially when dealing with large-scale pre-trained models [69, 170, 49, 70] since traditional fine-tuning methods, which involve adjusting a significant portion of a model's parameters for specific tasks, can be resource-intensive. Two common techniques in this domain are adapters [170, 49] and Low-Rank Adaptation (LoRA) [69, 39]. *Adapters* are lightweight modules inserted between the layers of a pre-trained model, which allows for targeted modifications to the model's behavior without altering the original pre-trained weights. This approach is beneficial as it reduces the number of parameters that need to be fine-tuned, thus lowering the computational burden. Adapters have shown effectiveness in various tasks, providing a flexible and efficient way to adapt large models to specific tasks or datasets. However, one limitation of adapters is the additional parameters they introduce, which can lead to increased computational requirements during inference. On the other hand, *LoRA* offers a different approach to PEFT. LoRA involves modifying the weight matrices of a pre-trained model using low-rank decomposition. This method allows for fine-tuning the model's behavior while maintaining the original structure and size of the weight matrices. The key advantage of LoRA is that it does not introduce additional parameters during the model's runtime. Instead, it updates the pre-existing weights to enhance the model's performance on new tasks with minimal increase in computational requirements. LoRA has been successfully applied in various fields, including NLP [69, 39, 26, 28] and computer vision [62], demonstrating its versatility and effectiveness. However, these methods, while efficient, only focus on single-task models.

Parameter-Efficient Training for Multi-Task Models: In a multi-task setting, PEFT is more challenging as the model must cater to the needs of multiple tasks simultaneously, often leading to increased complexity and potential for task interference. Consequently, some recent studies have proposed new solutions to extend the benefits of PEFT for multi-task adaptation. One such approach is the Hypernetworks [101], which uses shared networks to generate adapter parameters for all layers conditioned on the task, thus allowing for the sharing of information across different tasks while enabling task-specific adaptation through

task-specific adapters. Building on top of it, Polyhistor [99] explores PEFT in the domain of dense vision tasks, specifically on hierarchical vision transformers. Polyhistor proposes two ideas: decomposing hyper-networks into low-rank matrices and using custom kernels to scale fine-tuning parameters to the different transformer blocks. However, these two approaches rely on separate execution of the model for each task to apply its adapter, which does not benefit from the MTL's potential for efficient training or inference.

3.3 Methodology

Problem Setting: Given a general-purpose transformer-based backbone pre-trained on large-scale image datasets (e.g., ImageNet [36]), our goal is to efficiently adapt it to several downstream tasks in a Multi-Task Learning (MTL) architecture setting. We are considering the common MTL architecture with one shared encoder and multiple task-specific decoders, as shown in Figure 3.2b. Following the existing works in parameter-efficient training, the criteria of parameter-efficient MTL training include the accuracy of downstream tasks and the number of training parameters.

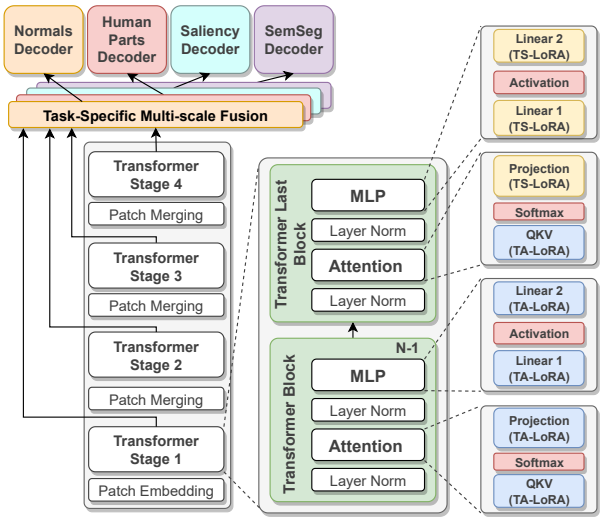
Method Overview: Our approach for efficiently adapting MTL models to various downstream tasks consists of two novel aspects: (1) efficiently share homogeneous information across tasks via a pool of task-agnostic and task-specific low-rank matrices and (2) efficiently allow multi-scale task-specific feature sharing between the shared-encoder and task-specific decoders of the MTL architecture.

This section is organized as follows. We start with an overview of the used MTL architecture in Subsection 3.3.1. Then, we propose our parameter-efficient task-specific adaptation method in Subsection 3.3.2. In Subsection 3.3.3, we propose our multi-scale task-specific efficient feature-sharing method. Finally, we explore the effect of fine-tuning the non-attention modules in Subsection 3.3.4.

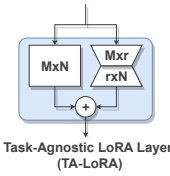
3.3.1 MTL Architecture Overview

We first establish a Multi-Task Learning (MTL) framework designed for experimentation with parameter-efficient adaptation in MTL contexts. Aligning with established MTL methodologies [163, 149], our model comprises three main components: a shared hierarchical encoder, task-specific inter-scale fusion modules, and a pool of task-specific decoders. We adopt an off-the-shelf hierarchical vision transformer as the shared encoder [100], which extracts visual features from input frames for all downstream tasks. The hierarchical structure of the encoder allows for capturing visual features at various scales, providing a comprehensive representation of the input data. The extracted multi-scale visual features are then fused and processed by various task-specific decoders to execute the downstream tasks. Our MTL framework is designed to accommodate different vision transformer and decoder architectures, which makes our parameter-efficient adaptation approach suitable for a wide range of MTL architectures.

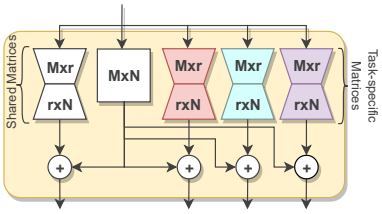
To effectively adapt the MTL architecture to various downstream tasks, we draw inspiration from the low-rank adaptation (LoRA) technique commonly employed in Language Models [69], traditionally used for single-task adaptation. Our primary inquiry is: *'How does fine-tuning low-rank matrices perform when optimizing for multiple visual downstream tasks?'*. Previous studies in low-rank adaptation mainly aim to identify a unique set of low-rank matrices for adapting the encoder to an individual downstream task [78,



(a) Low-Rank Adaptation matrices are applied to all the blocks of the shared backbone.



(b) TA-LoRA: Task-Agnostic Low-Rank Adaptation Module in MTLoRA



(c) TS-LoRA: Task-Specific Low-Rank Adaptation Module in MTLoRA

Figure 3.3: *MTLoRA* framework overview. Task-Agnostic LoRA modules (*TA-LoRA*) are placed at each transformer block, excluding the last ones in each stage where our Task-Specific LoRA (*TS-LoRA*) modules are placed to capture task-specific fine-tuning at different scales.

143, 101], as illustrated in Figure 3.2a. This approach requires running the entire model separately for each of the tasks, which is inefficient for real-time applications. In contrast, our research seeks to develop a single set of low-rank adaptation matrices that are applicable across multiple downstream tasks, as depicted in Figure 3.2b. This methodology enables a single execution of the backbone for all tasks. Multi-task learning often presents challenges, such as the "conflicting gradients problem" [77]. This issue becomes more pronounced in low-rank adaptation due to the limited number of trainable parameters.

3.3.2 Low-Rank Adaptation for MTL Architectures

Low-rank decomposition modules are increasingly used to adapt pre-trained models for various tasks [69]. These modules, incorporated into layers that involve matrix multiplication, are notably used in the attention layers of transformer-based models. The function of these modules can be mathematically described as follows:

$$Output_{Layer_i} = W_i x + b_i + \alpha B_i A_i x \tag{3.1}$$

Here, W_i and b_i are the original weights and biases of the layer. A and B are the rank decomposition matrices, and x is the input to the layer i. α is the adaptation scale which controls the deviation of the tuned model from the original model. During training, only parameters of A, B, and potentially b_i are trained, significantly reducing the memory footprint and leading to faster training. During inference, the A and B matrices can be merged into W_i, ensuring that the introduction of low-rank decomposition does not add any extra latency to the inference process. For Hierarchical Vision transformers, several locations within the architecture are identified as suitable for the application of low-rank matrices, enhancing task adaptability as shown in Figure

3.3a.

- **QKV Computation in Attention Layers**: The Query, Key, Value (QKV) computation, the main components of the attention mechanism, represents a prime candidate for low-rank adaptation. Fine-tuning this computation allows for modifications to the attention mechanism, making it more suited for specific downstream tasks, which can improve the model's ability to process visual inputs in a task-specific manner.
- **Projection Layer**: The projection layer in transformers is responsible for projecting the attention layer's output back to the original feature space. Fine-tuning the projection layer allows the attention output to be projected into the task's feature space, yielding better performance on downstream tasks.
- **Feed Forward Layers in the MLP block**: These layers, consisting of two dense layers (FC1 and FC2) with nonlinear activation in-between, transform the attention output into the final feature representation. Fine-tuning these layers dictates the model's capacity to effectively generate task-specific final feature representations to be processed by subsequent stages or task-specific decoders.

Adopting low-rank decomposition modules in these layers provides controllable knobs to trade-off fine-tuning efficiency (i.e., Number of trained parameters) and adaptation quality (i.e., Performance on the downstream tasks). We consider two variants of low-rank decomposition modules as shown in Figure 3.3: (1) *Task-Agnostic Low-Rank Adaptation* module to capture shared features among the various tasks, and (2) *Task-Specific Low-Rank Adaptation* module that can learn task-specific features.

Task-Agnostic Low-Rank Adaptation (*TA-LoRA*): In our framework, we utilize task-agnostic low-rank adaptation modules, which employ low-rank decomposition to adjust the corresponding weights, as detailed in Equation 3.1. The *TA-LoRA* modules are designed to identify and leverage shared features across multiple downstream tasks, thereby facilitating knowledge sharing. We have integrated *TA-LoRA* modules into the transformer blocks of the Hierarchical Vision Transformer backbone, with the exception of the final block in each stage, as illustrated in Figure 3.3a. Specifically, these *TA-LoRA* modules are applied to adapt key computational layers within the transformer blocks, namely the QKV Layer, the Projection Layer, and the MLP block. The inclusion of *TA-LoRA* modules aims to promote a balanced and synergistic learning process. This approach ensures fine-tuning that is unbiased towards any specific task, preventing overfitting. Additionally, to address the challenge of conflicting gradients in MTLoRA, the final block of each stage incorporates our novel *Task-Specific Low-Rank Adaptation* modules, which aim to capture task-specific features as explained in the following paragraph.

Task-Specific Low-Rank Adaptation (*TS-LoRA*): One of the main challenges in multi-task low-rank adaptation is to disentangle the feature learning space in order to solve the conflicts between the various downstream tasks. To achieve that, we propose our novel *TS-LoRA* modules. *TS-LoRA* incorporates separate task-specific low-rank matrices in addition to the shared low-rank matrices as shown in Figure 3.3c. These modules are designed to operate in two distinct modes. First, when a *TS-LoRA* module follows a layer with a *TA-LoRA* module (for instance, in the projection layer), it processes the shared input to derive task-specific representations. Conversely, in scenarios where a layer with a *TS-LoRA* module succeeds another with a

similar module (as observed in MLP feed-forward layers), it processes task-specific inputs to produce corresponding task-specific outputs as follows:

$$Output_{layer_i/task_j} = W_i x + b_i + \alpha_i B_{task_j} A_{task_j} x \tag{3.2}$$

Here, $Output_{layer_i/task_j}$ represents the $task_j$'s specialized output at layer i. W_i and b_i are the original weights and biases of the layer. x is input to $layer_i$. B_{task_j} and A_{task_j} are the *TS-LoRA* matrices for task j. These modules fine-tune the model according to the specific needs of each task. The outputs of the *TS-LoRA* modules are directed toward the corresponding task-specific fusion modules and decoders, as shown in Figure 3.3. This allows the encoder to generate task-specific feature representations at various scales. Since these *TS-LoRA* matrices are only connected to their corresponding task-specific decoders in the computation graph, the backward propagation only updates those matrices according to their corresponding task's loss.

In MTLoRA, the usage of both *TA-LoRA* and *TS-LoRA* modules is key to achieving an optimal balance between generalization and specialization within the MTL model. The *TA-LoRA* modules are designed to capture generalized information throughout the model, ensuring that a fundamental level of generality is maintained across various tasks. In contrast, the *TS-LoRA* modules are used to encapsulate unique updates that are tailored to each specific task. This dual-module approach ensures that while the model efficiently processes shared features relevant across multiple tasks, it also possesses the capacity to cater to the specific demands of individual tasks.

3.3.3 Multi-Scale Task-Specific Feature Sharing in Encoder-Decoder MTL Architecture

Multi-scale feature propagation within the encoder-decoder architecture has been shown to enhance performance in vision tasks [174, 149] where the input data is captured at various scales, providing various levels of abstractions. Typically, a hierarchical vision transformer processes input through multiple stages, with each stage generating features at a different scale. Merging features from these different scales results in a more comprehensive feature representation. In conventional setups, features at various scales are often fused together to create a unified, shared multi-scale feature set applicable to all tasks. However, our *TS-LoRA* module allows for a unique specialization at each scale for every task since it generates task-specific features at the end of every transformer stage as shown in Figure 3.3a. This enables the creation of task-specific multi-scale features which pushes the model to fine-tune the features at each scale according to the requirements of each task. Our learnable task-specific multi-scale fusion layers use a residual blocks-based architecture to combine the features at different scales (i.e., receptive fields) in an informative way for every downstream task.

3.3.4 Fine-tuning Non-Attention Modules

Several studies in the domain of parameter-efficient training have highlighted the benefits of unfreezing some low training-cost modules [49], such as layer normalization, which can positively impact the model's performance without significantly increasing the number of trainable parameters. Hence, we explore the effect

of unfreezing different modules within MTLoRA. In addition to training the shared *TA-LoRA* and the task-specific *TS-LoRA* modules, we unfreeze the patch embedding layer, the patch merging layer, the layer normalization, and the position bias in the attention layer. We provide insights about the effect of freezing each of those layers on the accuracy-efficiency trade-off in Subsection 3.4.4. Additionally, we explore adding low-rank decomposition modules to the *patch merging* module instead of completely unfreezing it. This allows for further reduction in training parameters; we denote this lighter version as *MTLoRA+* referring to those extra low-rank decomposition modules added outside the transformer blocks.

3.4 Experimental Results

3.4.1 Implementation Details

Dataset: We evaluate our method on the PASCAL MTL dataset [43]. Following other papers in MTL literature [163, 149, 161], we use the PASCAL-Context split that has annotations for various dense prediction tasks such as semantic segmentation, human part detection, surface normals estimation, and saliency distillation. It has 4,998 images in the training split and 5,105 in the validation split.

Evaluation metrics: Following common multi-task learning evaluation practices [149], the semantic segmentation, saliency estimation, and human part segmentation tasks are evaluated using mean intersection over union (mIoU). We use the root mean square error (rmse) in the predicted angles to evaluate the surface normals task. We also measure the overall performance Δm as the average per-task reduction in performance compared to the single-task baseline st:

$$\Delta m = \frac{1}{T}\sum_{i=1}^{T}(-1)^{l_i}(M_i - M_{st,i})/M_{st,i} \tag{3.3}$$

where l_i = 1 if a lower value means better for performance measure M_i of task i, and 0 otherwise. The single-task performance is measured for a fully converged model that uses the same backbone network only for that task.

Implementation: *MTLoRA* is implemented using PyTorch, and the code is publicly available on GitHub. Our main artifact is an easily pluggable *MTLoRALinear* layer that encapsulates our *TS-LoRA* and *TA-LoRA* modules, enabling the model to adapt to different tasks by using task-specific low-rank matrices. We use rank 4 for the task-specific matrices while we explore different ranks for the shared matrices. We adopt the publicly available *Swin Transformer* backbone [100], which was pre-trained on the ImageNet dataset [36] as our shared encoder. Then, we attach simple task-specific decoders for different dense tasks. Specifically, we use a simple decoder similar to the one in HR-Net [152], which includes linear and bilinear upsampling layers to efficiently perform dense vision tasks, and we adapt the number of output dimensions to different tasks. The number of decoder parameters is only 6% of the overall MTL model's parameters when using *Swin-Tiny* as a backbone. We run each experiment on a single NVIDIA V100 GPU.

Training: To train our multi-task learning model, we use a loss function equal to the weighted sum of the

Table 3.1: Results - MTLoRA versus SOTA parameter efficient training methods. The table summarizes the number of trainable parameters in each method. It also includes the accuracy of the downstream tasks as well as the average MTL model's accuracy (Δm). The last column indicates whether or not the model allows all the tasks to be executed simultaneously. The symbols $\uparrow$ and $\downarrow$ indicate higher and lower is better, respectively. **bold** numbers highlights how *MTLoRA* dominates the full finetuning while training $3.6\times$ less parameters.

Method	**SemSeg** ($mIoU \uparrow$)	**Human Parts** ($mIoU \uparrow$)	**Saliency** ($mIoU \uparrow$)	**Normals** ($rmse \downarrow$)	$\Delta m(\%)$	**Trainable Parameters** (M)	**Single Inference For All Tasks**
Single Task	67.21	61.93	62.35	17.97	0	112.62	×
MTL - Tuning Decoders Only	65.09	53.48	57.46	20.69	-9.95	1.94	✓
MTL - Full Fine Tuning	67.56	60.24	65.21	16.64	+2.23	30.06	✓
Adapter [60]	69.21	57.38	61.28	18.83	-2.71	11.24	×
Bitfit [168]	68.57	55.99	60.64	19.42	-4.60	2.85	×
VPT-shallow [78]	62.96	52.27	58.31	20.90	-11.18	2.57	×
VPT-deep [78]	64.35	52.54	58.15	21.07	-10.85	3.43	×
Compactor [81]	68.08	56.41	60.08	19.22	-4.55	2.78	×
Compactor++ [81]	67.26	55.69	59.47	19.54	-5.84	2.66	×
LoRA [69]	70.12	57.73	61.90	18.96	-2.17	2.87	×
VL-Adapter [143]	70.21	59.15	62.29	19.26	-1.83	4.74	×
HyperFormer [101]	71.43	60.73	65.54	17.77	+2.64	72.77	×
Polyhistor [99]	70.87	59.15	65.54	17.77	+2.34	8.96	×
MTLoRA ($r = 16$)	68.19	58.99	64.48	17.03	+1.35	4.95	✓
MTLoRA ($r = 32$)	67.74	59.46	64.90	16.59	+2.16	6.08	✓
MTLoRA ($r = 64$)	67.9	59.84	65.40	16.60	**+2.55**	**8.34**	✓
MTLoRA+ ($r = 4$)	68.12	57.77	63.14	17.60	-0.52	2.57	✓
MTLoRA+ ($r = 8$)	68.54	58.30	63.57	17.41	+0.29	3.15	✓
MTLoRA+ ($r = 16$)	68.28	58.70	64.323	17.034	+1.19	4.29	✓

losses of the various downstream tasks as follows:

$$Loss_{MTL} = \sum_{i}^{T} \omega_{task\_i} \times L_{task\_i} \tag{3.4}$$

where ω_{task_i} and L_{task_i} are the weight and the loss of the various tasks in the MTL model, respectively. Specifically, we use the standard per-pixel cross-entropy for semantic segmentation and human part segmentation, $L1$ loss for surface normals estimation, and balanced cross-entropy for saliency detection. We also adopt the task weights used by Vandenhende *et. al.* [149].

3.4.2 Baselines

To evaluate the performance of *MTLoRA*, we compare its accuracy and the number of training parameters to other parameter-efficient training methods. We first compare to *Single task* baselines where each task has a separate model and all parameters are fine-tuned to minimize the loss on the corresponding task. We also build a multi-task learning model with a shared encoder and task-specific decoders and evaluate the post-training accuracy when only decoders are fine-tuned (*MTL - Tuning Decoders Only*) and when the full model is fine-tuned (*MTL - Full Fine-Tuning*) to reduce the overall tasks losses as shown in equation 3.4.

As mentioned earlier, *MTLoRA* is the first to achieve parameter-efficient training for multi-task learning models. Therefore, to evaluate our method against state-of-the-art methods, we compare *MTLoRA* to other single-task parameter-efficient training methods where a task-wise module is added for each task using the

setup provided by Liu *et al.* [99]. Specifically, we compare to the following baselines: (1) Adapter [60] where task-specific bottleneck modules are placed into transformer layers (2) Bitfit [168] where only biases, patch merging layers, and patch projection layers are fine-tuned. (3) VPT [78] where tunable embeddings (i.e., 50 embeddings per layer) are inserted in the first input layer (VPT-shallow) and all layers (VPT-deep). (4) Compacter [81], which decomposes the fast matrix into two low-rank vectors, and Compacter++, which only places modules after MLP layers. (5) LoRA [69], where the low-rank decomposition is applied on attention layers with rank r = 4 and the adapter output scale (i.e., 4), which matches our *MTLoRA* hyperparameter. (6) VL-Adapter [143], which shares an adapter across different tasks. (7) Hyperformer [101], where a hyper-network is used to produce the weights for adapters for various tasks. (8) Polyhistor [99], where decomposed hyper-networks are used to share information across different tasks while still necessitating separate training and inference paths for each task.

3.4.3 Quantitative Analysis

For a fair comparison, *MTLoRA*, *MTLoRA+*, as well as all other baselines, are all based on the *Swin-Tiny* variant of the Swin Transformers family of models [100]. Table 3.1 shows the per-task accuracy, the overall MTL accuracy based on Equation 3.3, the number of trainable parameters of *MTLoRA* and *MTLoRA+* compared to our baselines. The last column indicates whether or not the corresponding parameter-efficient training method allows all the tasks to be executed simultaneously. As mentioned earlier in Figure 3.2, the ability to execute all tasks in a single inference path is essential for applications where efficiency and latency are critical. As shown in Figure 3.1, *MTLoRA* and *MTLoRA+* offer a Pareto for the trade-off between the number of trainable parameters and the accuracy of the downstream tasks.

3.4.4 Ablation

Effect of task-specific modules in MTLoRA: To show the effectiveness of the task-specific Low-rank-decomposition modules in *MTLoRA*, we compare the performance of *MTLoRA* and *MTLoRA+* to a similar setup with only task-agnostic low-rank decomposition modules. The results of this comparison, as shown in Figure 3.4, clearly demonstrate the impact of adding task-specific adaptation modules. Notably, the integration of these modules results in a substantial improvement in the accuracy-efficiency trade-off during parameter-efficient fine-tuning. This enhancement indicates the ability of the task-specific modules to effectively untangle the parameter space involved in MTL. Consequently, this leads to positive knowledge sharing during the fine-tuning process, significantly boosting the performance of each downstream task.

Effect of Various Backbone Adaptation Locations: As mentioned in Subsection 3.3.2, we insert low-rank decomposition modules in four different locations in the Hierarchical Transformer encoder: 1) the feed-forward layers in the MLP block (FC1 and FC2), 2) the QKV layer, and 3) the projection layer. We analyze the effect of removing the low-rank decomposition modules from each of those layers to get insights about the effectiveness of adapting each of those weights. Table 3.2 shows the overall MTL accuracy (Δm) versus the number of trainable parameters when the low-rank decomposition modules are removed from each of the four locations. Those results are from MTLoRA applied on a Swin-Tiny backbone where all low-rank

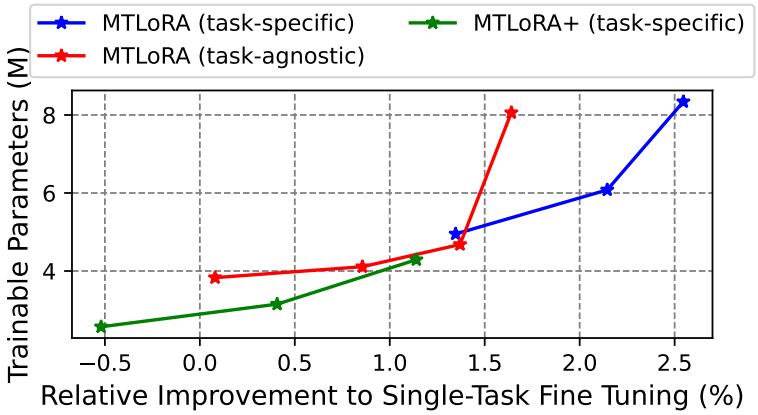


Figure 3.4: Accuracy versus trainable parameters of MTLoRA with task-agnostic vs task-specific adaptation modules.

Table 3.2: Effect of removing the various low-rank decomposition matrices in *MTLoRA* from the different locations in the backbone vision transformer. *None* refers to the default of MTLoRA, where all the low-rank modules are adopted.

	None	*No FC1*	*No FC2*	*No Proj*	*No QKV*
Δm	+2.55	+1.93%	+1.65%	+1.65%	+1.26%
Trainable (M)	8.34	6.81	6.81	7.73	7.21

decomposition modules have a rank of 32. We can see that adapting the QKV weights is the most important since removing its low-rank modules causes the most accuracy degradation. On the other hand, removing the adaptation from the first linear layer of the MLP block seems the least effective in comparison. However, the table shows that each low-rank decomposition module offers a significant improvement in the overall performance of the multiple downstream tasks. Removing some of them can be used to achieve a different accuracy efficiency trade-off during training.

Effect of Freezing Non-Attention Modules: As mentioned earlier in SubSection 3.3.4, besides fine-tuning the low-rank decomposition matrices, we unfreeze the patch embedding layer, the patch merging layer, the layer normalization, and the position bias in the attention layer. We analyze the effect of freezing these extra modules to provide insights into the accuracy-efficiency trade-off associated with each component. Table 3.3 shows the overall MTL accuracy (Δm) versus the number of trainable parameters when each of those modules is frozen. We can notice that unfreezing all those modules yields the highest accuracy.

Results with Larger Backbones and Pre-training Datasets: To analyze the efficacy of *MTLoRA* scaled backbone and pre-training datasets. We apply *MTLoRA* on *Swin-Base* that was pre-trained on *ImageNet22k* dataset. Figure 3.5 shows the improvement in the accuracy of *MTLoRA* compared to single-task models. We can see that MTLoRA scales well with a larger backbone, providing significant improvement compared to the single-task models while training significantly fewer parameters. More analyses are included in the Appendix.

Table 3.3: Effect of freezing the different modules outside the transformer block.

	Patch Embed	*Layer Norm*	*Position Bias*	*Patch Merging*
Accuracy (W/ ST)	+2.02	+2.06	+1.99	+1.74
Training Parameters (M)	8.34	8.32	8.32	6.80

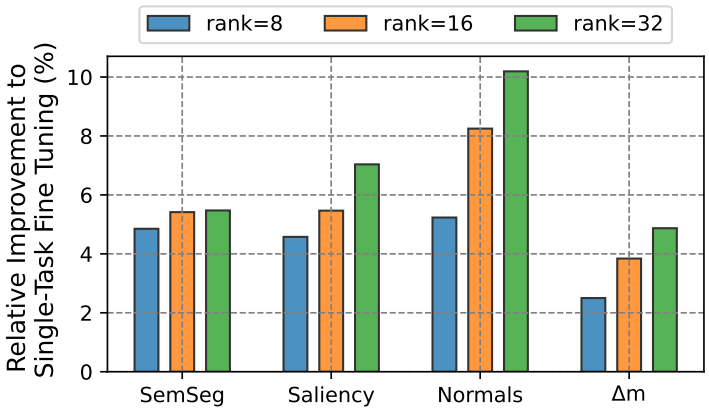


Figure 3.5: Performance of *MTLoRA* on various downstream tasks when applied to a *Swin-Base* model pre-trained on *ImageNet-22K*

3.5 Conclusion

In conclusion, this chapter introduced *MTLoRA*, an innovative framework designed to enable parameter-efficient training for Multi-Task Learning (MTL) models. Central to *MTLoRA* are the Task-Agnostic and Task-Specific Low-Rank Adaptation modules, which are instrumental in effectively disentangling the parameter space during MTL fine-tuning. Those modules allow the fine-tuning to balance both task specialization and interaction within MTL environments. We have demonstrated the application of *MTLoRA* in hierarchical-transformer-based MTL architectures, tailoring them to a variety of downstream dense prediction tasks. Our experiments show that *MTLoRA* not only surpasses the accuracy of fully fine-tuned MTL models but also achieves this with a substantially lower number of trained parameters ($3.6\times$ reduction in trainable parameters). Additionally, *MTLoRA* provides Pareto-optimality in the trade-off between the number of trainable parameters and accuracy compared to existing state-of-the-art parameter-efficient training approaches.

Chapter 4

Parameter-Efficient Gradient Conflict Resolution in Multi-Task Learning

4.1 Introduction

Based on our previous explorations of parameter-efficient multi-task learning (MTL), this chapter introduces a new approach to address one of the key challenges in MTL which is task interference and gradient conflicts. As we have established in earlier chapters, MTL offers significant advantages in terms of model efficiency and performance. However, the optimization process in MTL introduces unique challenges that require innovative solutions. To recap, multi-task learning aims to train a single model capable of handling multiple tasks simultaneously. By leveraging shared representations and parameters across different tasks, MTL enhances learning efficiency and significantly reduces the complexity and size of the model compared to training individual models for each task. Key benefits of MTL include improved model generalization, reduced risk of overfitting, and enhanced performance through effective cross-task knowledge transfer. Moreover, MTL aims to minimize the model's computational demands while maximizing its generalization capabilities across diverse tasks by efficiently sharing learned representations [148, 172]. The primary appeal of MTL, as we've discussed, lies in its potential to leverage shared information to improve the model's performance and learning efficiency by reducing hardware resources such as memory and FLOPS per task. By training on multiple tasks, the model is exposed to varied data points and objectives, which can lead to a more robust and well-rounded model. For instance, a model trained on both image recognition and object detection tasks could utilize shared features such as edges and textures, thus optimizing resource usage while improving task-specific performance [148, 172]. Despite these advantages, MTL introduces a unique set of challenges, particularly in the optimization process. A significant issue is the phenomenon of task interference or negative transfer, where learning one task can impede the performance of another [157, 166]. This interference often manifests as conflicts between gradients derived from different tasks, leading to a compromise in learning where one task's requirements overshadow another's [157, 166]. Task interference typically occurs when the

optimal parameter updates for different tasks are incompatible or contradictory. In the context of gradient-based optimization, this is observed when task gradients point in opposing directions, significantly hindering the model's convergence and overall performance. The severity of this issue is further amplified by the diverse and sometimes contradictory nature of the tasks that MTL models must learn simultaneously. Current methodologies to address these gradient conflicts in MTL are primarily centered around gradient manipulation techniques. These include the re-weighting of task losses [83], gradient projection methods to find a compromise direction [157, 166, 176], or even more complex adaptive methods that adjust learning rates or optimize task-specific parameters independently [135, 96, 95]. However, these approaches are often computationally expensive, lack scalability, and may not guarantee convergence to an optimal solution. Low-rank matrices have emerged as a powerful tool in machine learning, offering a way to represent high-dimensional data efficiently with fewer parameters [69]. Building on our previous work with low-rank adaptations, we observe a compelling opportunity to utilize these low-rank matrices to capture task-specific updates. While existing strategies focus on manipulating gradients to mitigate conflicts, low-rank adaptations provide an efficient mechanism for directly adapting the parameter space of the model, which simplifies the optimization process by reducing dependency on complex gradient manipulations. Hence, we propose CRLoMTL, a novel framework designed to address task conflicts in MTL using low-rank matrices. Our contributions are as follows:

- We introduce a new training strategy that alternates between task-specific passes and global passes. In the task-specific passes, each task constructs and updates its own set of low-rank matrices, capturing unique task-specific features and gradient information. In the global passes, we learn the optimal linear combination of these task-specific parameters using trainable coefficients followed by a softmax function to compute the weighted average of the task matrices, effectively integrating them into a single representative matrix that encapsulates task interactions while minimizing conflicts. Additionally, unlike many existing methods, the final trained model does not include any additional overhead in memory or computations.
- Unlike previous approaches that focus on gradient manipulation, CRLoMTL utilizes low-rank matrices to mitigate negative transfer directly in the parameter space. The low-rank matrices capture the task-specific features while generating a unified backbone for efficient inference.
- We employ feature regularization by using an L_2 loss between the task-specific and unified representations to mitigate overfitting to any single task. This regularization aims to promote generalization across all tasks.
- We present comprehensive results using the Swin Transformer family[100] as an MTL backbone, demonstrating significant performance improvements on the PASCAL-Context dataset [43]. Our empirical evaluation shows an enhancement of 2.82% over single-task models while training less than 4% of the parameters. Additionally, we evaluate CRLoMTL against multiple SOTA techniques where it dominates in terms of accuracy and number of trainable parameters.

CRLoMTL attempts to balance task-specific needs with global model optimization. Our approach provides substantial improvements over existing MTL methods, particularly in terms of computational efficiency and

model performance across diverse tasks. The rest of the chapter is organized as follows. We discuss the preliminary background and related work in Section 4.2. Then, we introduce CRLoMTL in Section 4.3. Section4.4 shows the setup and evaluation of CRLoMTL. Finally, we conclude in Section 5.5.

4.2 Background & Related Work

4.2.1 Multi-task Learning & Challenges

MTL is a machine learning paradigm that aims to leverage shared representations to learn multiple related tasks simultaneously. By training a single model to perform various tasks, MTL offers several advantages over traditional single-task learning through shared representations and faster learning through auxiliary information [148, 172]. MTL is particularly valuable in scenarios where data for individual tasks is limited, as it allows the model to benefit from the collective knowledge across all tasks [148, 172].

In a typical MTL setup, a model $f(x;\theta)$ with parameters θ is trained on a set of T tasks, each with its own loss function L_t. The overall objective is to minimize a weighted sum of task-specific losses:

$$L(\theta) = \sum_{t=1}^{T} w_t L_t(f(x;\theta)), \tag{4.1}$$

where w_t represents the weight assigned to each task, the choice of these weights is crucial and can significantly impact the model's performance across tasks. MTL has found applications in various domains, including computer vision [87], natural language processing [31], and healthcare [108]. In computer vision, for instance, a single model might be trained to perform object detection, semantic segmentation, and depth estimation simultaneously. In NLP, a model could be trained to perform named entity recognition, part-of-speech tagging, and sentiment analysis on the same input text. Despite its potential benefits, MTL faces several challenges that can hinder its effectiveness. A significant challenge in MTL is negative transfer, where forcing tasks to share representations can hinder the performance of one or more tasks [156]. This is particularly problematic when tasks are not sufficiently related or when the shared representation is not expressive enough to capture task-specific nuances. Task imbalance is also a significant concern, especially when tasks have different levels of difficulty or dataset sizes. In such cases, the model may prioritize easier tasks or those with more data, leading to suboptimal performance on more challenging or data-scarce tasks [29]. Conflicting gradients is a closely related issue to negative transfer. This occurs when the gradients of different tasks point in opposing directions, leading to a situation where optimizing for one task may degrade the performance of another [166]. Mathematically, gradient conflict can be identified when the inner product of gradients from different tasks is negative:

$$\langle \nabla_\theta L_i(\theta), \nabla_\theta L_j(\theta) \rangle < 0, \tag{4.2}$$

where $\nabla_\theta L_i(\theta)$ and $\nabla_\theta L_j(\theta)$ are gradients from tasks i and j respectively. The presence of conflicting gradients can lead to several adverse effects in MTL. The model may take longer to converge as it struggles to find a direction that improves all tasks simultaneously. Some tasks may suffer from reduced performance as the

model attempts to balance conflicting objectives. The learning process may become unstable, with performance fluctuating wildly across epochs. Additionally, the model may prematurely converge to a suboptimal solution that represents a compromise between tasks rather than excelling at all of them. The architecture design for MTL models presents its own set of challenges. Determining the optimal structure, including which layers to share across tasks and where to introduce task-specific components, can significantly impact performance [131]. Furthermore, MTL introduces additional complexity to the optimization process. Balancing the learning of multiple tasks simultaneously can lead to difficulties in convergence and stability [135].

4.2.2 Gradient-Based Methodologies

To address these challenges, researchers have proposed various approaches, with a particular focus on gradient-based methods. These methods aim to find the update directions that balance the needs of all tasks, often by manipulating gradients directly. PCGrad [166] addresses conflicting gradients by modifying each task's gradient with respect to all other tasks' gradients. Each task sequentially considers the gradients of all other tasks. If the current task's gradient conflicts with another task's gradient (i.e., their dot product is negative), it projects the current task's gradient onto the plane orthogonal to the conflicting task's gradient, removing the conflicting component. CAGrad [96] approaches MTL as a multi-objective optimization problem that aims to find an update direction that improves all tasks while staying close to their individual gradient directions. It formulates this as a min-max optimization problem, seeking a direction that maximizes the improvement of the worst-performing task. GradNorm [29] focuses on adaptive task weighting by adjusting the weights of different tasks during training based on the rate of loss reduction for each task. Multiple Gradient Descent Algorithm (MGDA) [38] is another approach that formulates MTL as multi-objective optimization. MGDA seeks a descent direction that improves all tasks simultaneously by solving a quadratic programming problem aiming to provide Pareto optimal solutions, where no task can be improved without degrading another. While these gradient-based methods address MTL challenges, they often introduce additional computational complexity. The trade-off between computational efficiency and performance improvement remains a consideration in the development of MTL methods.

4.2.3 Architectural Approaches

In addition to gradient-based methods, researchers have explored architectural solutions to address MTL challenges. These approaches focus on designing network structures that can effectively share information across tasks while maintaining task-specific representations. PAD-Net [161] introduces a MTL architecture for simultaneous depth estimation and scene parsing. It first predicts intermediate auxiliary tasks and then uses these predictions as multi-modal input for the final tasks. Building upon similar principles, Vandenhende et al. [149] introduced MTI-Net, extending the feature-sharing concept by incorporating multi-scale feature fusion and task-specific feature refinement. It uses a multi-scale feature extractor and introduces interaction modules that allow for bidirectional information flow between tasks at different scales. Other approaches, such as AdaShare[142], focus on building adaptive parameter-sharing policies. More recently, vision-transformers

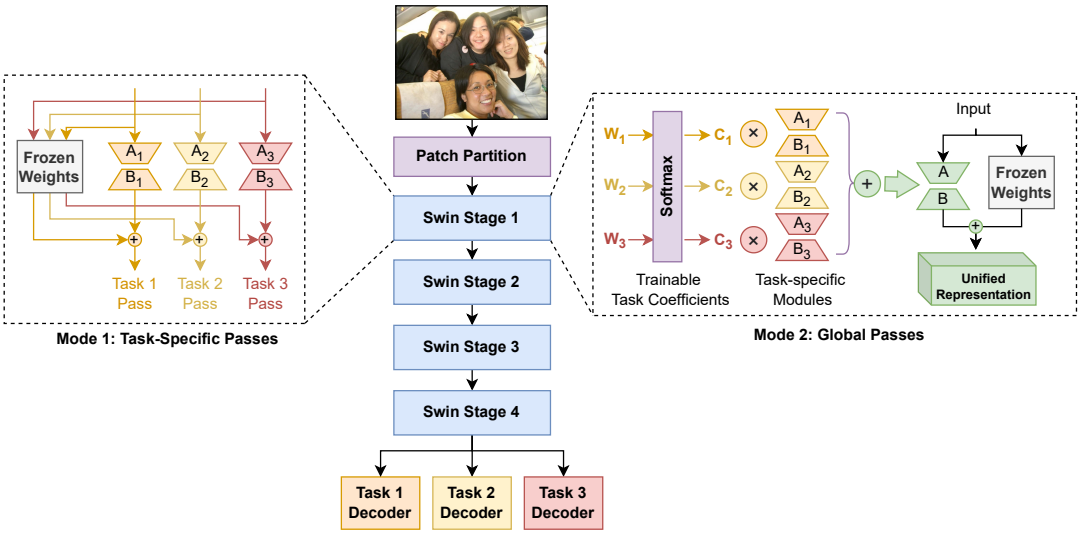


Figure 4.1: Overview of the CRLoMTL architecture and training modes. The model alternates between task-specific passes, where each task updates its own set of low-rank matrices and a global pass to generate a unified representation, where trainable coefficients are used to compute a weighted average of the task matrices.

became more popular as backbones for MTL models [163, 164]. For instance, InvPT [163] adapts the inverted bottleneck structure, commonly used in computer vision models, to the MTL setting. It employs a shared backbone followed by task-specific inverted bottleneck modules. MTLoRA is another transformer-based architecture that leverages low-rank matrices to model task interactions [8]. These architectural approaches offer different strategies for balancing shared representations with task-specific adaptations. While they can potentially capture complex inter-task relationships, they often require careful design choices and may increase the overall model complexity compared to single-task alternatives. Additionally, many existing approaches [149, 161, 8] require adding more parameters to the trained model or inducing branching in the inference flow, which adds performance overhead. Furthermore, the theoretical foundations of MTL are still being developed. Questions about the conditions under which MTL is beneficial, how to measure and maximize positive transfer between tasks, and how to quantify the relatedness of tasks remain active areas of research [169].

4.2.4 Parameter-Efficient Fine-Tuning (PEFT) and Low-Rank Adaptation (LoRA)

Recent advancements in PEFT techniques have shown promise in adapting large pre-trained models to specific tasks with minimal computational overhead. Among these techniques, LoRA [69, 39] has gained significant attention due to its simplicity and effectiveness. LoRA introduces task-specific low-rank matrices to adapt pre-trained model weights. For a pre-trained weight matrix $W \in \mathbb{R}^{d \times k}$, LoRA defines the adapted weight as:

$$W' = W + BA, \tag{4.3}$$

where $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times k}$ are low-rank matrices (typically $r \ll \min(d,k)$). During fine-tuning, only A and B are updated, significantly reducing the number of trainable parameters. PEFT methods like LoRA offer several advantages in terms of efficiency and flexibility. They require much less memory than full fine-tuning, as most of the model parameters remain frozen. Training is faster due to the reduced number of trainable parameters. Additionally, multiple task-specific adaptations can be stored and switched easily without modifying the base model. These characteristics make PEFT techniques particularly attractive for MTL scenarios, where they can potentially capture task-specific information efficiently while maintaining a shared base model.

4.2.5 Problem Formulation

In this section, we formally define the MTL problem and introduce the notation used throughout the chapter.

Let $\mathcal{D} = \{(x_i, y_i^1, ..., y_i^T)\}_{i=1}^N$ be a dataset of N samples, where $x_i \in \mathcal{X}$ represents the input features and $y_i^t \in \mathcal{Y}^t$ is the corresponding label for task $t \in \{1, ..., T\}$. We consider a shared model $f_\theta : \mathcal{X} \to \mathcal{Y}^1 \times ... \times \mathcal{Y}^T$ with parameters θ, which aims to learn T tasks simultaneously.

For each task t, we define a task-specific loss function $\mathcal{L}_t : \mathcal{Y}^t \times \mathcal{Y}^t \to \mathbb{R}$ that measures the discrepancy between the predicted output and the ground truth. The MTL objective can then be formulated as:

$$\min_\theta \sum_{t=1}^{T} w_t \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_t(f_\theta^t(x_i), y_i^t), \tag{4.4}$$

where $w_t > 0$ is the weight assigned to task t, and $f_\theta^t(x)$ denotes the output of the model for task t given input x. The gradient of the overall loss with respect to the model parameters θ is given by:

$$\nabla_\theta \mathcal{L}(\theta) = \sum_{t=1}^{T} w_t \nabla_\theta \mathcal{L}_t(\theta), \tag{4.5}$$

where $\nabla_\theta \mathcal{L}_t(\theta)$ is the gradient of the loss for task t. As discussed earlier, gradient conflicts occur when the gradients from different tasks point in opposing directions, as described in Equation 4.2. The primary challenge in MTL is finding a set of parameters θ that performs well across all tasks, even in the presence of conflicting gradients. This involves balancing the potentially competing objectives of different tasks and finding a compromise that leads to good overall performance. Hence, our proposed framework attempts to leverage task-specific low-rank matrices to adapt the model parameters for each task and find an optimal shared representation for all the tasks.

4.3 Methodology

CRLoMTL leverages low-rank matrix adaptations to address the challenges of MTL. Our approach operates directly on the model's parameter space, enabling task-specific adaptations while maintaining a shared representation. Figure 7.1b provides an overview of the CRLoMTL architecture. The framework alternates between two modes of operation during training: task-specific passes and global passes. In task-specific passes, each task independently updates its own set of low-rank matrices, capturing unique task-specific features. In

global passes, the model learns to combine these task-specific adaptations into a unified representation using trainable coefficients. The key components of CRLoMTL and described below.

4.3.1 Low-Rank Matrices for Task-Specific Learning

For each task t, we introduce task-specific low-rank matrices. These matrices allow us to model task-specific adaptations efficiently:

$$\theta_t = \theta + A_t B_t, \tag{4.6}$$

where θ represents the shared model parameters and θ_t represents the task-specific adapted parameters. These task-specific low-rank matrices are deployed across the different parts of the model to be updated in their corresponding training passes.

4.3.2 Multi-Pass Training & Trainable Coefficients

CRLoMTL employs a novel training strategy that alternates between task-specific passes and global passes:

Task-Specific Passes

During task-specific passes, shown in Figure 7.1b, each task t independently updates its own set of low-rank matrices A_t and B_t. In these passes, the inference for each task goes through both the frozen backbone and its task-specific parameters. Specifically, for a given task t, the model uses $\theta_t = \theta + A_t B_t$ in the forward pass, where θ represents the frozen backbone parameters. In the backward pass, only A_t and B_t for the current task t are updated. Importantly, both the backbone parameters θ and the low-rank matrices of all other tasks (A_j, B_j) for $j \neq t$ remain completely unaffected during this process. They are neither used in the computation nor updated. This design ensures that the computation graph for each task-specific pass naturally affects only the low-rank parameters of the given task while still leveraging the shared knowledge in the backbone. It enables the model to capture unique task-specific features and gradient information for each task without any interference or gradient conflicts from other tasks or the shared backbone.

Global Passes

In the global passes, shown in Figure 7.1b, we learn an optimal linear combination of the task-specific parameters. This is achieved using trainable coefficients c_t followed by a softmax function to compute the weighted average of the task matrices:

$$\theta_{\text{unified}} = \theta + \sum_{t=1}^{T} \frac{e^{c_t}}{\sum_{i=1}^{T} e^{c_i}} A_t B_t, \tag{4.7}$$

The intuition behind using the softmax is to effectively combine different matrices. By applying softmax to the trainable coefficients, we ensure that their sum is always 1 to preserve the scale of the unified matrix, which guarantees that the final output matrix remains within the same activation range as the individual

task-specific matrices, eliminating the need for additional normalization steps and helping to maintain the stability of the network during training and inference. Moreover, the softmax function provides a natural way to interpret the relative importance of each task's contribution to the unified representation. The resulting coefficients can be viewed as probabilities, offering insights into how the model prioritizes different tasks in the combined representation. The coefficients c_t are learned seamlessly within the same training flow since the linear combination of task-specific matrices becomes part of the computational graph. This approach allows the model to dynamically adjust each task's contribution to the unified representation enabling the model to find an optimal balance between tasks. The coefficients c_t are updated through backpropagation along with the other model parameters, allowing the model to learn the most effective combination of task-specific adaptations. By learning this optimal combination, the model can effectively integrate task-specific adaptations into a single representative matrix that encapsulates task interactions. In the final trained model, the unified matrix is merged seamlessly with the original model weight to maintain the same parameter count and inference efficiency as the original model, unlike many existing methods.

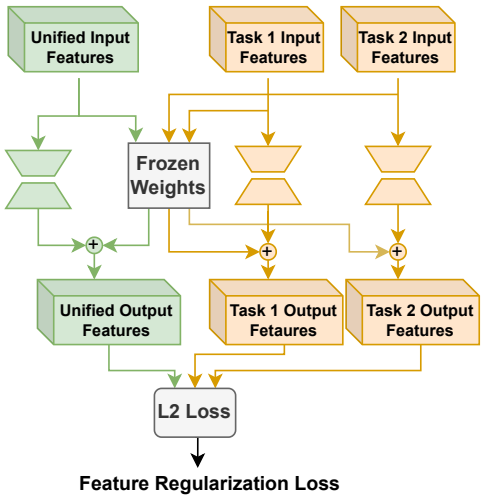


Figure 4.2: Feature Regularization process in CRLoMTL. The L_2 loss between task-specific output features and unified output features encourages similarity between task-specific and shared representations, promoting generalization across tasks and preventing overfitting to any single task.

4.3.3 Feature Regularization Loss

To prevent overfitting to any single task and encourage a balanced learning process, we incorporate feature regularization between the task-specific and unified representations, as shown in Figure 4.2. This is implemented as an L_2 loss added to the overall loss function:

$$L_{\text{reg}} = \sum_{t=1}^{T} |f_{\theta_t}(x) - f_{\theta_u}(x)|_2^2, \tag{4.8}$$

where $f_{\theta_t}(x)$ represents the feature outputs from the task-specific passes, $f_{\theta_u}(x)$ represents the feature outputs from the unified model. This regularization term encourages similarity between task-specific and unified representations, promoting knowledge sharing across tasks while allowing for task-specific specialization. Overall Loss Function and Optimization The overall loss function for CRLoMTL is thus:

$$L_{\text{total}} = \sum_{t=1}^{T} w_t L_t(\theta_t) + \lambda L_{\text{reg}}, \tag{4.9}$$

where $L_t(\theta_t)$ is the task-specific loss for task t, w_t is the corresponding task weight, and λ is a hyperparameter controlling the strength of the regularization. By combining these components, CRLoMTL can effectively balance task-specific adaptations with shared representations while addressing the challenges of conflicting gradients and negative transfer. The flow of CRLoMTL training is summarized in Algorithm 1.

Algorithm 1 CRLoMTL Training Algorithm

Require: dataloader, model, tasks, optimizer, λ
for each epoch in range(epochs) **do**
  for each (input, targets) in dataloader **do**

 # Task-specific passes:
    for each task in tasks **do**
      optimizer.zero_grad()
      task_output[task] = model(input, task)
      loss = task_loss$_{task}$(task_output[task], targets[task])
      loss.backward()
      optimizer.step()
    end for
    optimizer.zero_grad()
 # Combined pass:
 # The next step generates the learnable shared representation internally.
    unified_output = model(input)
    unified_loss = mtl_loss(unified_output, targets)
    reg_loss = reg_loss(unified_output, task_output)
    total_loss = unified_loss + $\lambda\cdot$ reg_loss
    total_loss.backward()
    optimizer.step()
  end for
end for

4.4 Experimental Results

4.4.1 Setup

Table 4.1: Evaluating CRLoMTL versus SOTA methods on PASCAL-Context dataset using Swin-Tiny backbone. Δm represents the relative improvement of the MTL model compared to the single-task models.

Method	SemSeg (mIoU ↑)	Human Parts (mIoU ↑)	Saliency (mIoU ↑)	Normals (rmse ↓)	Trainable Parameters (M)	Δm(%)
Single Task	67.21	61.93	62.35	17.97	112.62	0
MTL	67.56	60.24	65.21	16.64	29.47	+2.23
MTI-Net	66.93	59.66	64.76	16.23	31.47	+2.37
MTLoRA	67.90	59.84	65.40	16.60	8.34	+2.55
PCGrad	67.19	60.50	64.90	16.52	29.47	+2.46
CAGrad	67.53	60.59	64.87	16.58	29.47	+2.51
NashMTL	67.25	60.60	64.82	16.68	29.47	+2.26
MGDA	66.21	60.12	64.92	16.31	29.47	+2.22
CRLoMTL (base)	67.11	60.50	65.00	16.33	4.39	+2.73
CRLoMTL (+ft. reg.)	67.05	60.58	65.06	16.29	**4.39**	**+2.82**

Table 4.2: Evaluating CRLoMTL Using Swin-Small backbone.

Method	SemSeg (mIoU ↑)	Human Parts (mIoU ↑)	Saliency (mIoU ↑)	Normals (rmse ↓)	Trainable Parameters (M)	Δm(%)
MTL	70.20	63.30	66.09	16.17	50.78	+5.66
CRLoMTL (base)	70.83	63.60	66.32	16.20	6.75	+6.06
CRLoMTL (+ft. reg.)	71.16	63.63	66.30	16.26	**6.75**	**+6.12**

We implement CRLoMTL using PyTorch. The code and model weights will be made publicly available on GitHub. We adopt the publicly available Swin Transformer backbone [100], pre-trained on the ImageNet-1K dataset [36], as our shared encoder. We attach simple task-specific decoders for different dense tasks, similar to the one in HR-Net [153], which includes linear and bilinear upsampling layers. We use rank 8 for the LoRA matrices with the typical LoRA scale of 2.0. We evaluate our method on the PASCAL dataset [43]. Following the common practice in MTL literature [149, 97], we use the PASCAL-Context split, which provides annotations for various dense prediction tasks, including semantic segmentation, human part segmentation, surface normal estimation, and saliency detection. The dataset consists of 4,998 images in the training split and 5,105 images in the validation split. The number of decoder parameters constitutes only 7% of the overall MTL model's parameters when using Swin-Tiny as a backbone. To train our MTL model, we use a loss function equal to the weighted sum of the losses of the various downstream tasks:

$$\text{Loss}_{\text{MTL}} = \sum_{i=1}^{T} \omega_{\text{task } i} \times L_{\text{task } i}, \tag{4.10}$$

where $\omega_{\text{task } i}$ and $L_{\text{task } i}$ are the weight and the loss of the various tasks in the MTL model, respectively. We use the standard per-pixel cross-entropy for semantic segmentation and human part segmentation, L1 loss for

surface normal estimation, and balanced cross-entropy for saliency detection. We adopt the task weights used by Vandenhende et al. [149]. All experiments are conducted on a single NVIDIA V100 GPU using a batch size of 32.

4.4.2 Baseline Methods

We compare CRLoMTL against several SOTA MTL methods to evaluate its performance:

MTI-Net

We implement MTI-Net [149], utilizing its multi-scale feature fusion and task-specific feature refinement capabilities.

CAGrad

We apply CAGrad [96] to the MTL model, leveraging its gradient update strategy to balance task-specific objectives.

PCGrad

We apply PCGrad [166] employing its gradient projection technique to mitigate negative transfer between tasks.

NashMTL

We implement NashMTL [111], utilizing its game-theoretic approach to find equilibrium among competing tasks in our MTL scenario.

MGDA

We apply MGDA [38] to our multi-task setup, using its mechanism for gradient updates.

MTLoRA

We evaluate against MTLoRA [8] that utilizes low-rank matrices to model task interactions.
Each of these methods is implemented within the same experimental framework as CRLoMTL to ensure a fair comparison. We use the same backbone architecture and dataset across all methods, with the primary difference being the specific MTL strategy employed by each approach.

4.4.3 Evaluation Results on PASCAL-Context

We use mean intersection over union (mIoU) to evaluate semantic segmentation, saliency estimation, and human part segmentation. For the surface normal task, we use the root mean square error (RMSE) in the

predicted angles. To measure overall performance, we use Δm, which is the average per-task reduction in performance compared to the single-task baseline:

$$\Delta m = \frac{1}{T}\sum_{i=1}^{T}(-1)^{l_i}\frac{(M_i - M_{st,i})}{M_{st,i}}, \tag{4.11}$$

where $l_i = 1$ if a lower value means better for performance measure M_i of task i, and 0 otherwise. Single-task performance is measured for a fully converged model using the same backbone network only for that task. Tables 4.1 and 4.2 present the results for models using Swin-Tiny and Swin-Small backbones, respectively. The base model indicates using CRLoMTL without the feature regularization loss. We compare CRLoMTL against existing SOTA MTL methods and single-task baselines. As shown in Table 4.1, CRLoMTL achieves a 2.82% improvement over the single-task baseline while using only 4% of the parameters when using Swin-Tiny backbone, demonstrating how CRLoMTL achieves more optimal trade-offs in balancing task interactions. In Table 4.2, CRLoMTL shows a 6.12% improvement over the single-task baseline while maintaining less than 6% of the parameters when using Swin-Small backbone, maintaining its effectiveness with increased model capacity. Additionally, unlike existing methods such as MTI-Net [149] and MTLoRA [8], CRLoMTL does not incur any additional parameters or branching overhead in the final model. Moreover, the parameter efficiency from the low-rank matrices provides a competitive advantage on resource-constrained devices. The results emphasize CRLoMTL's potential in addressing some of the challenges in MTL, particularly in balancing task-specific adaptations with shared representations.

4.5 Conclusion

We introduced CRLoMTL, a novel framework designed to address the challenges in training MTL models. Our approach leverages low-rank matrix adaptations to effectively manage gradient conflicts and task-specific requirements, operating directly on the model's parameter space. We demonstrated the effectiveness of CRLoMTL on the widely used PASCAL-Context[43] dataset for dense prediction. Our experiments, comparing against existing MTL methods, showed that CRLoMTL dominates in both performance and number of trainable parameters. Our work offers a new perspective on balancing task interactions within MTL setting. CRLoMTL's ability to achieve high performance with reduced parameter counts makes it suitable for applications with limited computational resources or requiring rapid task adaptation. Future work could explore CRLoMTL's application to other domains beyond computer vision and further analyze the principles underlying task conflicts and knowledge sharing in MTL models.

Chapter 5

Input-Dependent Dynamic Sparsification for Parameter-Efficient Multi-Task Learning

5.1 Introduction

Modern computer-vision-based applications require solving multiple tasks simultaneously in order to form a complete perception of the surrounding visual environment. For example, a simple augmented reality application might need to determine the surface normals, estimate the depths, detect an object of interest, and track it. Similarly, an autonomous vehicle should be able to detect both static and dynamic objects in the scene, determine their proximity and track them [73, 109]. Moreover, all that complex processing needs to be performed on every input frame in real-time, which is extremely challenging, especially since those applications usually run on resource-constrained devices with strict compute, memory, and energy budgets. That is why enabling efficient computation models where these tasks can be performed simultaneously is crucial to make those applications practical. As we discussed the foundations of multi-task learning (MTL) in earlier chapters, this chapter addresses the challenges of optimizing MTL models for these real-world computer vision applications. As previously established, MTL offers significant advantages in terms of efficiency and performance. However, the complexity of MTL models, particularly in the context of resource-constrained devices, presents unique challenges that require innovative solutions. That is why enabling efficient computation models where these tasks can be performed simultaneously is crucial to make those applications practical. In recent years, Multi-task learning (MTL) has been used to learn related vision tasks simultaneously [139, 97, 169]. On the one hand, MTL models leverage shared representations and inter-task interactions to improve performance on each task, potentially outperforming single-task models. On the other hand, compared to single-task models, MTL models can reduce both the memory footprint, the energy consumption, and the latency during inference since they avoid recomputing the features in the shared layers. Different

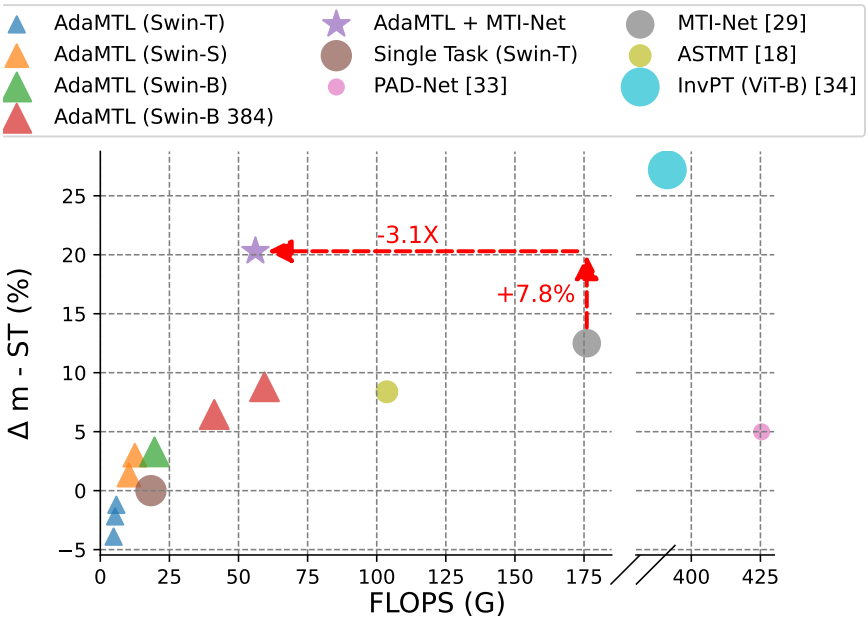


Figure 5.1: Accuracy-Efficiency trade-off by AdaMTL compared to SOTA MTL techniques. The x-axis shows the FLOPS, while the y-axis represents the average accuracy of the tasks compared to the single-task model.

approaches have been proposed to determine which layers should be shared across tasks and which layers should be task-specific in MTL models [106, 161, 142].

One common MTL approach is to have a shared encoder that extracts the critical features from the input scene, followed by some task-specific decoders that predict the output of the corresponding task [148, 149, 161]. Generally, the shared encoder in MTL models needs to have a large representational capacity in order to generalize well to various tasks and input data from different complexities. Vision Transformers have proven to be a powerful tool for extracting strong feature representations from the inputs, improving the performance of the downstream tasks [100, 53, 145]. Hence, few recent works have explored using them as a shared encoder for feature extraction in MTL models [22, 163]. Although these approaches achieve impressive performance outcomes, their computational complexity and memory footprint are usually huge, making them impractical for real-time processing. That's why, in this work, we focus our efforts on improving the computational complexity of transformer-based Multi-task Learning models.

Real-world visual scenes have large variations in complexity. For example, a scene with a single object and an open background would be easier to process than one with an occluded object and a cluttered background. We argue that treating all input frames equally regarding processing complexity can be wasteful. Therefore, we propose using an adaptive inference policy to reduce the computational complexity of MTL models based on the input complexity. Dynamic Input-dependent inference policies have only been explored for single-task image classification problems [167, 103]. However, MTL models are more complex due to inter-task dependencies and complex architectures.

To this end, we propose an adaptive MTL framework that recognizes the unnecessary computations in the

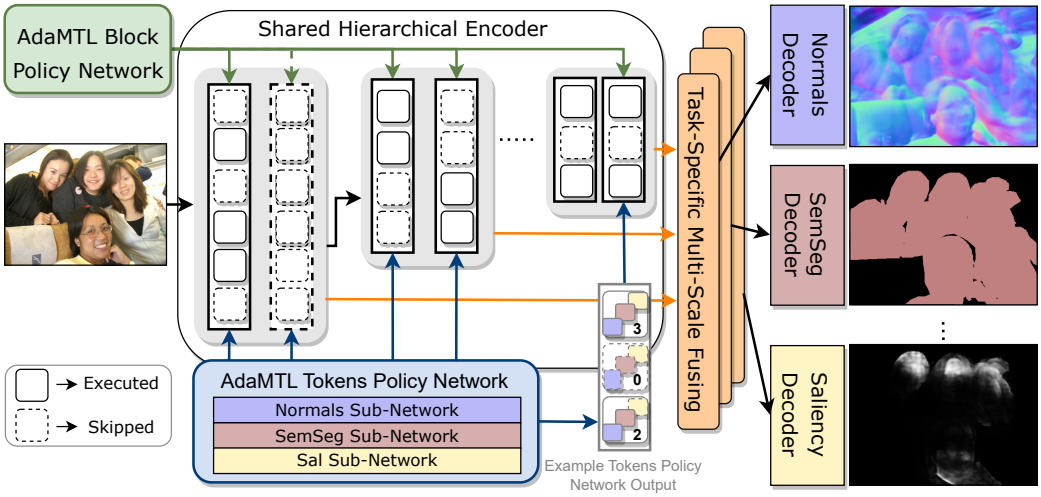


Figure 5.2: Overview of our proposed AdaMTL framework integrating our AdaMTL Block Policy Network and our AdaMTL Tokens Policy Network. The AdaMTL Block policy network decides on which blocks to activate during runtime. If the decision is to activate a certain block, our AdaMTL Tokens policy network runs to decide which tokens to process through that block. Our policy network achieves a task-aware behavior that improves the quality of the generated policies for multi-task learning models.

model depending on the input complexity. We achieve this by learning an adaptive task-aware policy network that ultimately decides on which parts of the MTL model to activate during runtime. Our contributions can be summarized as follows:

- We propose an adaptive Multi-task Learning framework that optimizes Transformer-based MTL models depending on the input complexity.
- We introduce a task-aware policy network, and we show that it is more effective in recognizing the unnecessary computations in MTL models compared to task-agnostic policy networks.
- To prove that our policy network can be easily plugged to improve MTL models' efficiency, we combine AdaMTL with a SOTA MTL model [149] and show that AdaMTL boosts the accuracy by 7.8% while improving the efficiency by 3.1×.
- We deploy AdaMTL on the Vuzix M4000 AR glasses [151], reducing the inference latency and the energy consumption by up to 21.8% and 37.5%, respectively, compared to the static MTL model.

The rest of the chapter is organized as follows. We review the related work in Section 5.2. Then, we introduce our adaptive input-dependent MTL framework in Section 5.3. Next, we show the qualitative and quantitative analysis of our methodology as well as the ablation study in Section 5.4. Finally, we conclude in Section 5.5.

5.2 Background and Related Work

Deep Multi-task Architectures For dense prediction tasks, deep multi-task architectures usually consist of an encoder that extracts a feature representation from the input frame, followed by task-specific decoders that generate predictions for each of the downstream tasks [161, 106, 149]. Researchers categorize multi-task architectures based on the location of task interactions into encoder- and decoder-focused architectures [148]. Encoder-focused architectures share the information between tasks in the encoder stage [106, 98], while the decoder-focused architectures exchange information between tasks in the decoding stage [149, 161]. Moreover, some approaches share the information across tasks in both encoder and decoder stages [142, 22]. In this chapter, our baseline MTL model follows an encoder-focused architecture where a shared encoder is used to extract visual features from the input frame, followed by task-specific decoders. Moreover, our MTL framework can easily adopt other decoder-based task-interaction techniques, as we will show in Subsection 5.4.3.

Vision Transformer Models Transformers [150] have achieved impressive performance improvements in various domains such as language understanding [40], speech recognition [55], and computer vision [41, 145]. The self-attention modules in transformers have proved to be capable of extracting strong feature representations from the inputs, improving the performance of the downstream tasks. To preserve local visual context, Vision Transformers (ViTs) split the input image into patches which are embedded as tokens [41]. Those tokens pass through successive ViT blocks. Each ViT block has a multi-head attention module followed by a multi-layer perceptron module to extract the global relationship among input tokens. Improvements have been made to ViTs enabling data-efficient training [145] as well as efficient inference [53]. Inspired by ResNets [61], Swin Transformers [100] use a hierarchical ViT block architecture as well as shifted window attention to serve as a general-purpose backbone for computer vision tasks. That is why, ViTs started replacing CNNs as a backbone for different computer vision tasks such as image classification [145, 53], object detection [25], and segmentation [141]. Moreover, several works proposed using them for multi-task learning [22, 163].

Sparsely-Activated Vision Models As computer vision models become increasingly complex, researchers have started sparsifying the models to prevent redundant computations. For example, DSelect-k [59] and M^3ViT [93] proposed Mixture-of-Experts (MoE) architectures that use trainable sparse gates to activate a subset of experts depending on the given input. Other methods employ early-exiting strategies to adaptively allocate computations depending on input complexity [175, 71]. Other dynamic models skip redundant layers [155], while others use a bottleneck layer to direct the computations in an input-dependent manner [113]. More recent approaches use lightweight policy networks to generate execution strategies based on the input complexity [167, 103]. Despite the effectiveness of these approaches for image classification problems, adaptive policies have not yet been explored for MTL scenarios. Sparsifying MTL models is more challenging; the multi-objective nature of those models complicates the overall objective of optimizing the performance of the different tasks in addition to the policy networks. Therefore, there is a need to explore and develop techniques that can adaptively sparsify MTL models based on input complexity while being aware of the multi-objective nature of those models.

5.3 Methodology

In this section, we propose AdaMTL – an adaptive end-to-end Multi-task Learning framework. We start by presenting our transformer-based MTL architecture in Subsection 5.3.1. In Subsection 5.3.2, we introduce our task-aware policy network that dynamically recognizes the unnecessary computations in the MTL model depending on the input complexity. Finally, we explain our proposed multi-staged task-aware training recipe in Subsection 5.3.3.

5.3.1 Overview

Figure 5.2 illustrates an overview of our proposed adaptive end-to-end Multi-task Learning framework - AdaMTL. AdaMTL consists of two main components: a static MTL model and a lightweight policy network. Our MTL model has three parts: a Shared Hierarchical Encoder, a learnable task-specific multi-scale fusing layer, and a pool of task-specific decoders. We adopt an off-the-shelf hierarchical Vision Transformer *Swin* [100] as our shared encoder to extract visual features from the input frames. Vision Transformers (ViTs) usually split the input image into patches which are embedded as tokens. Those tokens pass through successive ViT blocks. Each ViT block has a multi-head attention module followed by a multi-layer perceptron module to extract the global relationship among input tokens. Our learnable multi-scale fusing layers use a residual blocks-based architecture [61]. They are added to combine the features at different scales (i.e., receptive fields) in an informative way for every downstream task. Finally, we use simple task-specific decoders consisting of two convolutional layers. Each decoder takes the output from the corresponding multi-scale fusing layer to generate the corresponding task predictions.

On top of our static MTL model, AdaMTL adds a lightweight policy network that runs alongside the original model to decide which parts of the model to activate, adapting to the complexity of the input frame. Our policy network works as a multi-grained decision maker; it consists of an *AdaMTL Block policy network* and an *AdaMTL Tokens policy network*. The *AdaMTL Block policy network* decides on which blocks to activate during runtime. If the decision is to activate a certain block, our *AdaMTL Tokens policy network* runs to decide which tokens to process through that block. Our intuition behind AdaMTL is that not all patches in the input frame are equally informative; for example, a patch of an input frame from the background is less informative than a patch with a person for a task such as human-parts detection. Moreover, the receptive field at which different patches should be processed differs depending on the scale of the objects in the input frame. For example, an image with multiple smaller objects might benefit from being processed by the first few layers of the models where the receptive field is small. However, the later layers with larger receptive fields are essential for an accurate output on a zoomed-in frame with one object. In other words, our *AdaMTL Policy Network* decides which patches are needed to accurately perform the downstream tasks and the receptive field at which those patches should be processed.

5.3.2 AdaMTL Policy Network

Our policy network works as a multi-grained decision maker; the *Block Policy Network* decides on which blocks to activate, while the *Tokens Policy Network* decides on which tokens (i.e., patches) to process through

the activated blocks. Our *Block Policy Network* generates a learnable binary mask for each block in the shared encoder. We use this binary mask to recognize the necessary blocks needed by the MTL model in order to perform well on the downstream tasks. Similarly, for each block, we attach a *Tokens Policy Network* that generates a learnable policy to determine the tokens that need to be processed through the activated block. Intuitively, the policy network should learn to recognize the most informative patches in every input frame as well as the receptive field at which it needs to be processed. Each policy network has two simple fully-connected layers, followed by a Gumbel Softmax activation to generate binary masks [76]. We devise two different settings for the policy network: a *task-agnostic* policy network and a *task-aware* policy network.

Task-agnostic Policy: In the *task-agnostic* setting, the policy network is unaware of the number of downstream tasks. We achieve this by co-training the whole policy network alongside the MTL model. We mainly experiment with this setting to show the necessity of task awareness while creating an effective policy network for multi-task scenarios.

Task-aware Policy: In the *task-aware* setting, we want our policy network to capture task-specific computational needs. We achieve this by dividing the policy network into sub-networks, where each sub-network is responsible for recognizing the necessary blocks/tokens for the corresponding task. As shown in Figure 5.2, the *AdaMTL Tokens Policy Network* consists of a sub-network for each task (i.e., Normals Controller, Semantic Segmentation Controller, Saliency Controller, etc.). Each sub-network makes a decision that is plausible to its respective task. Finally, to get a unified policy, we make a decision to activate a token if at least one of the tasks needs to process it. The intuitive way to combine the masks would be to apply the logical ORing operation on all the generated task-specific masks. However, to make it more learnable, we combine the masks using addition and clamping. As shown in the example output from the Tokens Policy Network in Figure 5.2, the policy network only activates a token if at least one task-specific policy sub-network decides to activate it.

5.3.3 AdaMTL Training Recipe

For our adaptive MTL framework to work effectively, we need to learn the MTL model weights as well as the execution policy (i.e., policy network's binary masks) that achieves the target efficiency without compromising the accuracy of the various tasks in our MTL model. Our end-to-end training recipe consists of 3 stages:

Stage 1: Static MTL Model Training

First, we train a static MTL model. We adopt a shared encoder along with task-specific decoders to perform multi-task learning as explained in Subsection 5.3.1. For our shared encoder, we use the publicly-available pre-trained Swin Transformer backbones [100]. Then, we attach the multi-scale fusing layer as well as task-specific decoders, and we fine-tune the end-to-end MTL model to get our static baseline. In this stage, our loss function is the weighted sum of the losses of the various downstream tasks as follows.

$$L_{stage1} = \sum_{i}^{m} \omega_{task\_i} \times L_{task\_i} \tag{5.1}$$

where ω_{task_i} and L_{task_i} are the weight and the loss of the various tasks in the MTL model, respectively, and

m is the number of downstream tasks in the MTL model. We adopt the task weights used by Vandenhende *et. al.* [149].

Stage-2 Policy Network Initialization

We aim to co-train both the policy network as well as the MTL model. Randomly initializing the policy network while co-training can lead to degrading the model accuracy since the policy network would make random decisions in the earlier epochs. That's why we choose initialization weights for the policy network that activates all the blocks and the tokens. To achieve this, we freeze the static MTL model and pre-train our AdaMTL policy network with the following loss function:

$$L_{stage2} = \sum_{k}^{blocks} (1 - M_{b/k}) + \sum_{k}^{blocks} (1 - M_{t/k}) \tag{5.2}$$

Where $M_{b/k}$ and $M_{t/k}$ are the output masks generated by the Block Policy Network and the Tokens Policy Network attached to the Encoder block k, respectively. This results in an adaptive model that behaves exactly like the static model, where all blocks and tokens are activated. This acts as a good initialization point to start co-training the policy network and the MTL model.

Stage 3: Policy Network/MTL Model Co-training

In this stage, we co-train the policy network along with the MTL model. Our goal is to learn the binary masks that our policy network should generate in order to meet the target computational budget (i.e., the target percentage of the MTL model components to be activated) while maintaining the accuracy of the downstream tasks. Thus, we use a multi-objective loss as shown in Equation 5.3. Our loss function incorporates the various task losses L_{tasks} as well as the efficiency loss L_{eff} multiplied by a factor α. α represents the efficiency weight which controls the trade-off between accuracy and efficiency. In our experiments, we set α to unity; however, different values can be used to control the trade-off depending on the application requirements.

$$L_{stage3} = L_{tasks} + \alpha L_{eff} \tag{5.3}$$

The efficiency loss incorporates the efficiency of the decisions made by the blocks as well as the tokens controller, as shown in Equation 5.4. In order to minimize the number of activated blocks, we set L_{blocks} as mean squared error (MSE) between the actual percentage of the activated blocks and the target percentage of activated blocks as shown in Equation 5.5. Similarly, we set L_{tokens} as the MSE between the actual percentage of the activated tokens and the target percentage of activated tokens. However, it is common for hierarchical ViTs to have more tokens in the earlier layers compared to the later layers (i.e., the earlier layers have smaller receptive fields, thus more patches, while later layers have larger receptive fields, thus fewer patches). This means that the number of tokens does not linearly reflect the computational complexity since layers with more tokens have smaller embedding dimensions per token, while layers with fewer tokens have larger embedding dimensions. That's why we multiply the percentage of tokens in each layer by a weight ω_d equivalent to the embedding dimension in this layer as shown in Equation 5.6.

$$L_{eff} = L_{blocks} + L_{tokens} \tag{5.4}$$

$$L_{blocks} = MSE(\frac{B_{activ}}{B_{total}}, \frac{B_{target}}{B_{total}}) \tag{5.5}$$

$$L_{tokens} = MSE(\frac{\omega_d \times T_{activ}}{T_{total}}, \frac{\omega_d \times T_{target}}{T_{total}}) \tag{5.6}$$

Algorithm 2 AdaMTL - Alternating Task Training (ATT)

Require: $model$: Model with initialized policy network
Require: $Tasks$: Task names in MTL model
Require: m: Number of tasks in MTL model
Require: $epochs_{att}$: Number of epochs for ATT
Require: L_i: Loss for the epoch i
1 **for** $i = 0$ to $epochs_{att}$ **do**
2 $current\_task \leftarrow Tasks[i \mod m]$
3 $L_i \leftarrow L_{current\_task} + \alpha L_{eff}$
4 **for all** $task \in Tasks$ **do**
5 **if** $task = current\_task$ **then**
6 $model$.unfreeze_decoder($task$)
7 $model$.enable_policy_network($task$)
8 **else**
9 $model$.freeze_decoder($task$)
10 $model$.disable_policy_network($task$)
11 **end if**
12 **end for**
13 train_one_epoch($model, L_i$)
end for

To make our policy network task-aware, we co-train each task-specific policy sub-network independently with the end-to-end model. Sequentially co-training the policy sub-networks along with the end-to-end model suffers from catastrophic forgetting (i.e., the model gets biased towards behaving well on the last task, and the performance deteriorates on the earlier tasks). That's why we propose an *Alternating Task Training* (ATT) where we shift the focus between the tasks every one epoch. Algorithm 2 shows the steps of our ATT technique. For each epoch, we choose a task to focus on, and we set the loss accordingly, as shown in lines 2 and 3, respectively. Then, we only activate the decoder and the policy sub-network corresponding to the chosen task, as shown in lines 4-13. We train the MTL model for one epoch using that setting. Then, we move on to co-training the decoders and the policy sub-networks of the other tasks. Finally, we perform end-to-end fine-tuning to improve the overall model accuracy. The training loss remains the same as in Equation 5.3, and we unfreeze all the model's components. This results in an adaptive MTL model that generates task-aware inference policies depending on the complexity of the input.

5.4 Experiments

5.4.1 Setup

Dataset: We evaluate our method on the PASCAL dataset [43]. Following other papers in MTL literature [163, 149, 161], we use the PASCAL-Context split that has annotations for various dense prediction tasks

Table 5.1: Quantitative analysis of AdaMTL on PASCAL dataset. The table shows the accuracy-efficiency trade-off by our adaptive MTL model compared to the single-task model as well as static MTL models. *H* and *L* represent high and low computational complexity targets for AdaMTL, respectively. Δ m (ST) and Δ FLOPS (ST) show the change in the average accuracy of the tasks and percentage of FLOPS compared to the single-task model, respectively. $\downarrow$ means the lower the better, while $\uparrow$ means the higher the better. **Bolded** values represent the Pareto-frontier of the accuracy-efficiency trade-off.

Policy	**Backbone**	**Image Size**	**Sal** maxF $\uparrow$	**Human Parts** mIoU $\uparrow$	**Sem Seg** mIoU $\uparrow$	**Normals** mERR $\downarrow$	**Δ m (ST)** (%) $\uparrow$	**GFLOPS** $\downarrow$	**Δ FLOPS (ST)** (%) $\downarrow$	**Params** (M) $\downarrow$
Single-Task	Swin-T	224	71.93	48.63	60.35	18.45	0.00	18.33	1 $\times$	111.42
Static MTL		224	74.15	47.62	59.08	19.20	**-1.20**	5.79	**0.32** $\times$	34.77
AdaMTL (H)	Swin-T	224	73.72	47.64	57.13	19.16	**-2.18**	5.34	**0.29** $\times$	34.87
AdaMTL (L)		224	73.01	46.85	55.9	19.54	**-3.86**	4.82	**0.26** $\times$	34.87
Static MTL		224	75.00	50.66	61.84	18.71	+2.35	12.5	0.68 $\times$	67.02
AdaMTL (H)	Swin-S	224	75.23	51.22	61.88	18.79	**+2.65**	12.51	**0.66** $\times$	67.12
AdaMTL (L)		224	74.75	50.07	59.91	18.61	**+1.31**	10.35	**0.57** $\times$	67.12
Static MTL		384	73.93	56.68	67.47	17.69	**+8.81**	59.39	**3.24**$\times$	108.66
AdaMTL (H)	Swin-B	384	76.55	56.28	65.04	17.74	**+8.43**	51.29	**2.80**$\times$	108.88
AdaMTL (L)		384	76.23	55.04	62.63	17.92	**+6.46**	41.229	**2.25**$\times$	108.88

such as semantic segmentation, human part detection, surface normals estimation, and saliency distillation. It has 4,998 images in the training split and 5,105 in the validation split.

Implementation and Training details: We implemented AdaMTL using PyTorch. As mentioned in Subsection 5.3.1, we adopt the publicly available pre-trained Swin Transformer backbone [100] as our shared encoder. To get our adaptive MTL model, we employ a three-stage training recipe as explained in Subsection 5.3.3. In Stage 1, we fine-tune the Swin Transformer backbone along with our task-specific decoders for 1000 epochs. Then in stage 2, we freeze the MTL model and initialize the policy network by training it to activate the whole MTL model. We run this stage for another 80 epochs. Finally, in stage 3, we use our proposed *Alternating Task Training technique* to co-train the policy network along with the MTL model for another 150 epochs, followed by fine-tuning the end-to-end AdaMTL model for another 150 epochs. Our method does not only increase the model efficiency during inference, but it also reduces the carbon emission since we avoid retraining complex ViTs from scratch by reusing off-the-shelf pre-trained backbones; AdaMTL needs only 1 V100 GPU for around 24-48 hours (i.e., depending on the used backbone) in order to run our end-to-end training recipe.

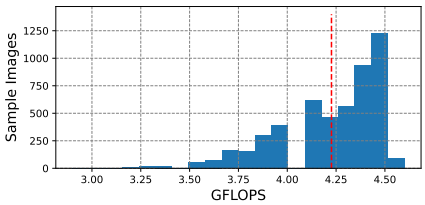
5.4.2 Quantitative Analysis

Accuracy-Efficiency Trade-off: We apply AdaMTL on three SOTA ViTs from the Swin Transformer family [100]. We include *Swin-Tiny*, *Swin-Small*, and *Swin-Base*, representing three different scales of ViTs in terms of computational complexity. We include two different computational complexity targets: *H* and *L*. *H* represents a higher computational budget where the target percentage of activated tokens and blocks are 60% and 90%, respectively. *L* represents a lower computational budget where both the target percentage of activated tokens and blocks are set to 50%. In both settings, the accuracy-efficiency trade-off weight (i.e., α in Equation 5.3) is set to unity. To evaluate the accuracy-efficiency trade-off by AdaMTL, we compare it to

Table 5.2: Comparison with SOTA MTL models.

Method	**Backbone**	**Δ m** (ST) ↑ (%)	**Δ FLOPS (ST)** ↓ (%)
PAD-Net [161]	HRNet-18	+4.98	23.21×
AdaMTL (Ours)	Swin-B	**+6.46**	**2.25** ×
ASTMT [102]	R26-DLv3	+8.38	5.66 ×
AdaMTL (Ours)	Swin-B	**+8.81**	**3.24** ×
MTI-Net [149]	ResNet-50	+13.49	9.6 ×
InvPT [163]	ViT-B	+27.20	21.35 ×

(a) 55% FLOPS (b) 65% FLOPS (c) 75% FLOPS (d) 85% FLOPS



(e) GFLOPs Histogram of AdaMTL on PASCAL dataset

Figure 5.3: Qualitative insights on the computational budget allocated by AdaMTL to process input frames of different complexity. We can notice that AdaMTL assigns fewer computations for simpler scenes, as in (a) and (b), while it justly assigns more computations to process more complex and cluttered scenes as in (c) and (d).

two baselines: (1) Single-Task models and (2) Static MTL models (i.e., our base MTL model before attaching our task-aware policy network). Table 5.1 shows the accuracy, the computational complexity (i.e., FLOPS) as well as the model size (i.e., Params) of AdaMTL compared to the single-task model and the static MTL models. Δ m (ST) and Δ FLOPS (ST) show the change in the average accuracy of the tasks and percentage of FLOPS compared to the single-task model, respectively. Results show how our method enhances the accuracy-efficiency trade-off for MTL models. For example, by applying AdaMTL to *Swin-S* backbone, we can get an MTL model with 43% less FLOPS and 1.31% more accuracy compared to the single-task model. Similarly, by applying AdaMTL to *Swin-T* backbone, we can get an MTL model with 71% less FLOPS and only 2.18% drop in accuracy compared to the single-task model. Therefore, given any target computational complexity, AdaMTL can meet it while potentially improving the accuracy.

Comparison to SOTA MTL models: We also compare the accuracy-efficiency trade-off of AdaMTL to four SOTA MTL models that vary in computational complexity and accuracy. Figure 5.1 shows the accuracy and the computational complexity of AdaMTL compared to those of *PAD-Net* [161], *ASTMT* [102], *MTI-Net* [149], and *InvPT* [163]. We can notice that AdaMTL dominates both *PAD-Net* and *ASTMT*. Moreover, AdaMTL achieves a more efficient trade-off compared to *MTI-Net* and *InvPT*. As shown in Table 5.2, AdaMTL has 3× less FLOPS than MTI-Net and 7× less FLOPS than *InvPT*. It is important to note that our effort in AdaMTL is directed specifically toward enabling efficient Multi-Task learning. That is why we argue

Table 5.3: Combining AdaMTL with SOTA MTL components.

Method	Backbone	Δ m %	Δ FLOPS ↓	Params (M)
Single-Task	Swin-T	+0.00	18.33 G	111.42
AdaMTL	Swin-B	+8.81	1.1×	108.66
MTI-Net [149]	ResNet-50	+13.49	9.6×	91.00
AdaMTL + MTI-Net	Swin-B	**+20.29**	**3.1**×	101.14

that while the performance of *MTI-Net* and *InvPT* is impressive, their demanding computational complexity might not be suitable for real-time processing on resource-constrained devices.

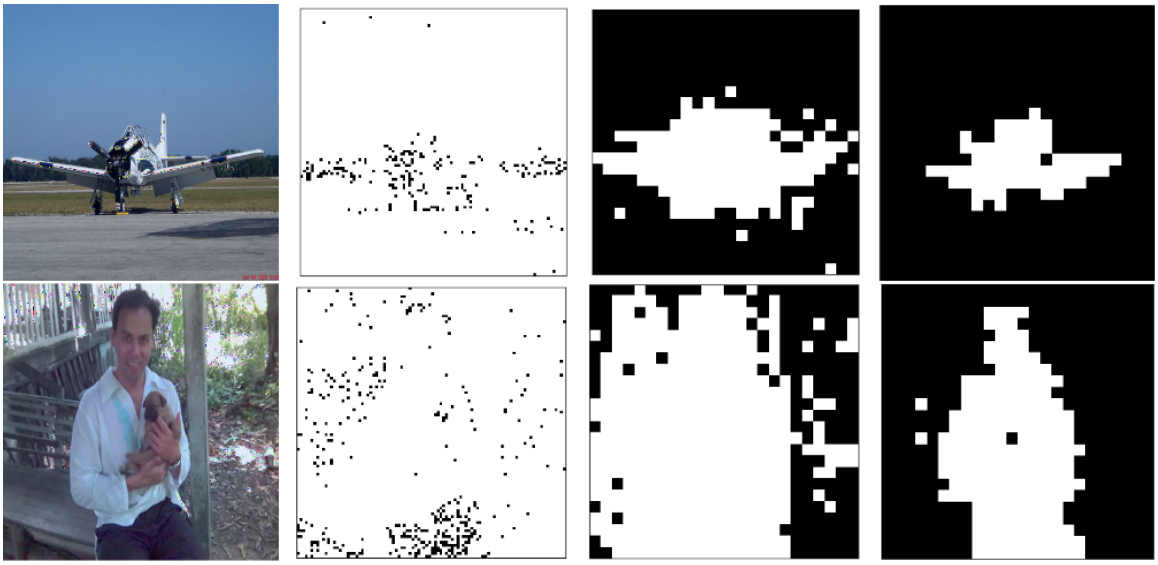
5.4.3 Combining with SOTA MTL components

Our adaptive MTL framework can easily adopt other SOTA MTL components to further enhance the accuracy-efficiency trade-off. In this section, we show a case study where we integrate AdaMTL with the SOTA MTL concepts in *MTI-Net* [149] in order to improve both its efficiency and accuracy. *MTI-Net* has two main modules: (1) A multi-scale multi-modal distillation unit to model task interactions at different scales and (2) A feature propagation module that propagates distilled task information from lower to higher scales. In this experiment, we attach those two modules between the shared hierarchical encoder and the task-specific decoders in our AdaMTL framework. Following the exact same training recipe introduced in Subsection 5.3.3, the results in Table 5.3 show that AdaMTL can be integrated with other MTL modules from *MTI-Net* to boost *MTI-Net*'s accuracy by 7.8% while improving its efficiency by 3.1×.

5.4.4 Qualitative Analysis

Figure 5.3 shows some insights about the allocated amount of computations by AdaMTL for input frames of different visual complexity. Figures 5.3a, 5.3b, 5.3c, and 5.3d shows examples where AdaMTL allocated 55%, 65%, 75%, and 85% of the static model FLOPS respectively. We can see that AdaMTL allocates fewer computations to simple frames with fewer objects as compared to complex scenes with multiple objects and cluttered backgrounds. Figure 5.3e shows a histogram of AdaMTL's computational complexity per example for all the images in PASCAL validation set. The red line represents the average number of FLOPS needed to process all the images. We can notice that AdaMTL adapts to the large variation in the computational complexity requirement by various images in the dataset.

To gain more insights into the decisions made by our policy network, we visualize a sample of the generated tokens masks in Figure 5.4. Column 5.4a represents the input frames, while columns 5.4b, 5.4c, and 5.4d represent the generated tokens masks by our policy network at different layers, such that the white areas represent activated tokens. We can notice that the policy network tends to activate more tokens in the earlier layers to understand the global features of the input frame. Then, it narrows down its scope in the later layers, focusing on the most informative patches (i.e., patches with the main objects) in the input frame.



(a) Input Frame (b) Layer 1 (c) Layers 12-15 (d) Layer 18

Figure 5.4: Sample of the generated masks by AdaMTL tokens policy network. The white areas in (b)-(d) represent activated tokens.

Table 5.4: Performance Analysis on Vuzix Augmented Reality Glasses [151]. We analyze the percentage of the inference latency and the energy consumption by our AdaMTL model compared to its corresponding static model.

Method	Backbone	Inference Latency (%)	Energy Consumption (%)
AdaMTL	Swin-T	-20.6%	-35.3%
AdaMTL	Swin-S	-21.8%	-37.5%

5.4.5 Deployment on Vuzix M4000 AR glasses

We compile our AdaMTL model using PyTorch for Android [121], and we deploy it on the Vuzix M4000 AR glasses [151]. The Vuzix glasses have an 8 Core 2.52Ghz Qualcomm XR1 board with 6GB RAM. It operates using Android 11.0. Using the *Battery Historian* tool [64] to profile the energy consumption on the device, we record the average latency and energy consumption across random samples from the PASCAL validation dataset. Table 5.4 shows that AdaMTL reduces the inference latency and the energy consumption by up to 21.8% and 37.5%, respectively, compared to its corresponding static MTL model.

5.4.6 Ablation Study

Quality of the learnt inference policies: To analyze the quality of the learned inference policies by our task-aware policy network. Table 5.5 compares the accuracy-efficiency trade-off achieved by our task-aware policy network (i.e., referred to as Task-Aware) to three other baselines: (1) Random where we activate random blocks and tokens from the static MTL model, (2) Random+ where we again activate random blocks and

Table 5.5: Comparison between the quality of our task-aware policy, the task-agnostic, and the random execution policy.

Policy	**Backbone**	Δ **m** (ST) (%) $\uparrow$	Δ **FLOPS** (ST) $\downarrow$
Random	Swin-T	-34.53	0.24×
Random+		-8.64	0.24×
Task-Agnostic		-5.68	0.24×
Task-Aware		**-3.86**	**0.26**×
Random	Swin-B	-34.86	0.82×
Random+		-8.57	0.82×
Task-Agnostic		+0.77	0.84×
Task-Aware		**+1.12**	**0.83**×

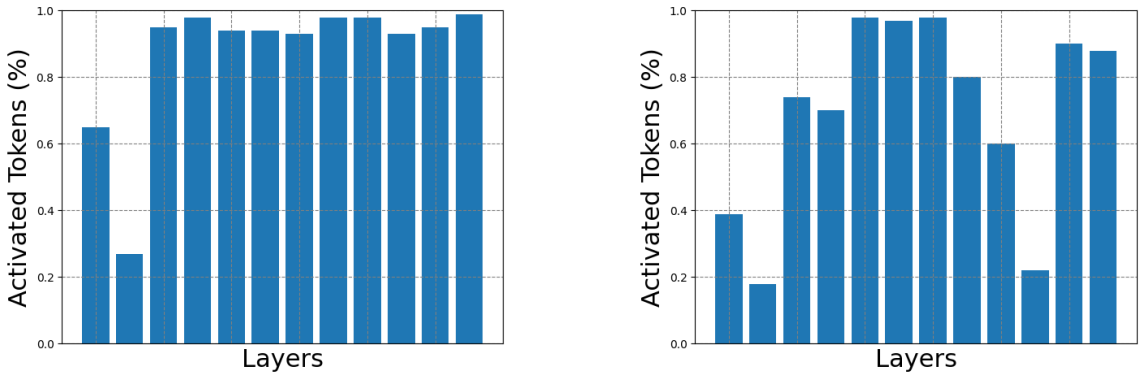
Table 5.6: The contribution of different adaptive dimensions to AdaMTL. Δ m is measured relative to the static MTL model.

Adaptive Blocks	Adaptive Tokens	Δ m %	FLOPS (G)
×	×	-0.00	5.8
✓	×	-1.66	5.23
×	✓	-1.46	5.08
✓	✓	**-0.85**	5.37

tokens from the static MTL model but after performing our adaptive training pipeline, and (3) Task-Agnostic policy network explained in Subsection 5.3.2. The results in Table 5.5 show that the policies learned by our task-aware policy network outperform the other baselines. We can also notice that performing adaptive training gives the MTL model robustness towards sparsification (i.e., Using a random policy on the static model reduced the accuracy by 34%, while it only reduced the accuracy by 8% when applied to the adaptively trained MTL model).

Analysis of the adaptation along each component of our policy network To understand the contribution of both components of our policy network (i.e., blocks policy network and tokens policy network) to the accuracy-efficiency trade-off of AdaMTL, we compare the results from four different settings: (1) the static model where neither component of the policy network is activated, (2) AdaMTL while activating the block policy network only, (3) AdaMTL while activating the tokens policy network only, and (4) AdaMTL where both policy networks are activated. Table 5.6 shows that enabling both components enhances the accuracy-efficiency trade-off of AdaMTL.

Analysis of the behavior of our loss function To analyze the importance of the weight factor ω_d in Equation 5.6 of our loss function, we visualize the average percentage of activated tokens across the blocks of Swin-T encoder with and without ω_d in Figure 5.5. We can notice that adding the ω_d factor to the loss function leads to a more distributed FLOPS reduction across different blocks, which is essential for the effectiveness of our adaptive MTL framework.



(a) Without weighted tokens

(b) With weighted tokens

Figure 5.5: Loss function ablation: The figure illustrates the average percentage of activated tokens over the 12 blocks in Swin-T using (a) non-weighted tokens loss and (b) weighted tokens loss.

5.5 Conclusion

In this chapter, we propose AdaMTL - an adaptive framework that learns task-aware inference policies for the MTL models in an input-dependent manner. We achieve this by co-training a lightweight policy network along with our MTL model. During runtime, our policy network recognizes the unnecessary computations and dynamically chooses an execution strategy depending on the input complexity and the target computational budget. Our experiments on PASCAL dataset demonstrate that AdaMTL reduces the computational complexity by 43% while improving the accuracy by 1.32% compared to the single task models. Combined with SOTA MTL components, AdaMTL boosts the accuracy by 7.8% while improving the SOTA MTL model efficiency by 3.1×. Finally, we deployed AdaMTL on Vuzix M4000 AR glasses showing up to 21.8% and 37.5% reduction in inference latency and energy consumption, respectively, compared to the static MTL model.

Chapter 6

Alternative Parameter-Efficient Platform for Machine Learning through Acid-base Networks

6.1 Introduction

Semiconductor devices have led the information technology revolution over the past several decades. However, there are good reasons to explore alternative methodologies for information storage and data processing, such as their potential for greater power efficiency, greater affordability, biological compatibility, and ability to adapt to different environmental conditions that may be inhospitable to conventional semiconductor technologies. One interesting research direction is a molecular computing paradigm [2][24] that relies on chemical reactions, which can be considerably more space- and energy-efficient than digital approaches [54]. Various methodologies have been proposed for molecular computation. For example, some demonstrations have leveraged chemical reaction-diffusion processes, often using the Belousov–Zhabotinsky (BZ) reaction [50] as a chemical oscillator [52]. While these oscillators' dynamics can be used to perform tasks such as classification [116], they incorporate complex spatio-temporal signals rather than stable endpoints, and they are sensitive to factors such as active sample mixing and temperature. Other hybrid approaches utilize electronic actuators to assist in encoding the inputs for reaction initiation [116]. A different approach is proposed by Arcadia *et al.* where they model the autocatalytic reactions as an activation function used in artificial neural networks [13]. They demonstrate an autocatalysis-based digital image recognition model with the images encoded in the concentration of a catalyst. In this work, we use acid-base concentration to encode digital information. DNA computing has been the most widely studied form of molecular computing [24][119][122][138]. It relies on DNA strands to carry information and execute different logic and arithmetic operations. Sequence-specific hybridization among networks of DNA sequences is used to model the algorithm chosen to solve the given problem. Nevertheless, DNA has limitations such as temperature

sensitivity, long computation times, and the need for large numbers of custom reagents and optimizations to perform experimental demonstrations of computation. In this chapter, we use a chemical approach that employs acid-base reactions as a means for universal computations orchestrated by an acoustic liquid handler. The proposed chemical system is straightforward. A mixture of a strong acid (HX) and a strong base (YOH) can be summarized as reaching an equilibrium between three reactions:

$$\mathrm{HX} + \mathrm{H_2O} \rightarrow \mathrm{H_3O^+} + \mathrm{X^-} \tag{6.1}$$

$$\mathrm{YOH} \rightarrow \mathrm{Y^+} + \mathrm{OH^-} \tag{6.2}$$

$$\mathrm{H_3O^+} + \mathrm{OH^-} \rightleftarrows 2\mathrm{H_2O} \tag{6.3}$$

An interesting property of this simple chemical system is provided by the neutralization reaction (Eq. 6.3), which couples the results of the acid and base reactions, primarily leaving behind only the more abundant species. Let x_i denote the type of droplet i (e.g., acid or base). If we consider a mixture of an odd number of n droplets with equal volume and concentrations, with m of these droplets being acids, then:

$$\mathrm{Majority}(x_1, x_2, \ldots, x_n) = \begin{cases} \text{Acid } (=+1) & \text{if } 2m > n \\ \text{Base } (=-1) & \text{if } n > 2m, \end{cases} \tag{6.4}$$

which is analogous to the majority function in Boolean logic.

Importantly, the majority function combined with the inversion operation form a functionally complete set [11], which means these operations are sufficient to model any Boolean logic function. However, the inversion operator is challenging to realize in chemical computation since inversion would require a conditional bistable chemical network which can be toggled without knowledge of the current state. As best we know, there is no simple reaction that would both change an acid to a base, and change a base to an acid. To obviate the need for complex reaction networks, we can take advantage of the symmetry of Equations (1) and (2), and recognize an opportunity to encode information in the difference between acidic and basic solutions. Thus, we propose a dual-rail encoding that represents data values as complementary pairs, e.g., (acid, base) or (base, acid). An encoded value will be represented by a concentration of [H_3O^+]=x ions ($\mathrm{pH} = -\log x$) in one rail and a concentration of [OH^-]=x ions ($\mathrm{pH} = 14.0 - \log x$) in the complementary rail. Differential encoding supports the inversion operation by simply exchanging the roles of the complementary rails during computations. Computations performed in the dual-rail form conveniently produce both the output and its logical complement, similar to some common digital structures like the D Flip-Flop [125]. Figure 7.1b illustrates some of the fundamental building blocks of our methodology. The concept of complementary representations, such as dual-rail encoding, has been used to solve relevant problems in other computational systems [79][123]. For example, Jiang *et al.* used complementary solutions with different concentrations to represent bits within their approach [79]. Similarly, Qian *et al.* used the notion of complementary inputs and gates to design seesaw gates [123]. Our approach extends the dual-rail information concept to universally-relevant acid/base chemistry.

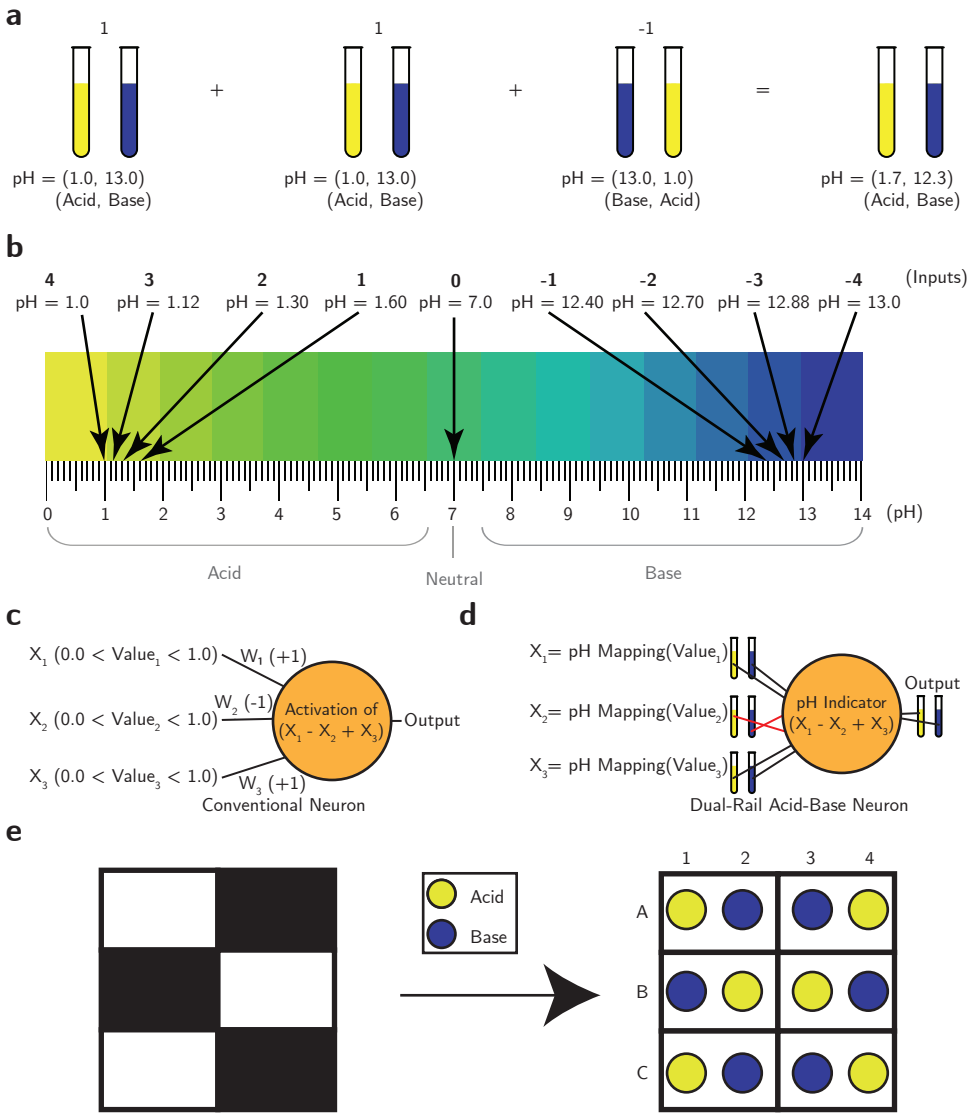


Figure 6.1: **Method overview. a** The dual-rail acid-base encoding for an addition followed by a subtraction (using the inverted representation). Each "1" is represented by an acidic solution (on the pH rail) and a basic solution (on the complementary pH rail); a "-1" is represented by the same solutions in reverse order. **b** Illustration of the mapping of discretized inputs into their corresponding pH values. **c** Conventional neuron with its default operations. **d** Equivalent neuron using an acid-base encoding with analog input values mapped to the corresponding pH value from the calculated range. Crossing lines indicate multiplying by -1. **e** Encoding of a 2x3 binary array of pixels into a well plate of acids (yellow) and bases (blue), where each pixel value is mapped to two wells.

6.2 Results

Encoding data in a dual-rail, acid-base representation.

As illustrated in Figure 7.1b, our encoding approach represents each bit using a pair of complementary

solutions where the pair (Acid, Base) represents the value "1" or TRUE, while the pair (Base, Acid) represents the value "-1" or FALSE. A pair with neutral pH 7 represents the midpoint of the full scale, which is the boundary between TRUE and FALSE, or the value "0" when applicable. In some applications, we can further extend the encoding to a larger set of discrete values by controlling the concentration of the encoded values. For example, to build a 3-bit encoding system with eight possible non-zero values, we can discretize the data symmetrically into four negative values {-4, -3, -2, -1} and four positive values {1, 2, 3, 4}. The positive values would be encoded as (Acid, Base), and the negative values would be encoded as (Base, Acid).

For each data value, we dilute the solution by adding an amount of water that is equal to:

$$\text{added water volume} = (\text{initial volume} \times \frac{\text{maximum encoded value}}{\text{encoded value}}) - \text{initial volume} \tag{6.5}$$

We make these solutions experimentally by providing an acoustic liquid handler (Echo 550, Labcyte) with stocks of acids, bases, and deionized water. Each computational operation involves transporting portions of the input solutions to an output well plate. The role of the liquid handler is analogous to the wires in an electronic circuit. As shown in Figure 7.1bc, for each value in the information sequence being encoded, if the value is positive, we instruct the liquid handler to dispense an acid droplet followed by a base droplet in the next available slots in the destination grid. Similarly, if the value is negative, the liquid handler dispenses a base droplet followed by an acid droplet, instead. Finally, if the computation involves more discretized values, the liquid handler dispenses water droplets into the corresponding wells to dilute the solutions as described in equation 6.5. Finally, we add a pH indicator to the final outputs of the computation to read the results. If the indicator measures an (Acid, Base) pair, this signifies that the output is TRUE, while a (Base, Acid) pair would indicate FALSE.

Modeling digital circuits using acid-base reactions. As discussed in the Introduction, when mixed, acids and bases naturally execute the majority function and inversion can be realized by exchanging the roles of the complementary pair. Since the majority and inversion operations are functionally complete [11], they can be used to represent any logic function. Figure 6.2 depicts primitive AND, OR, INV, NAND, and NOR logic gates represented in terms of acid-base operations. In order to avoid neutral pH outputs, all of the building blocks should have an odd number of inputs. To construct two-input logic gates, the third input is a constant bias. Table 6.1 shows the truth table for the AND gate and the corresponding representation using acid-base reactions with a dual-rail encoding.

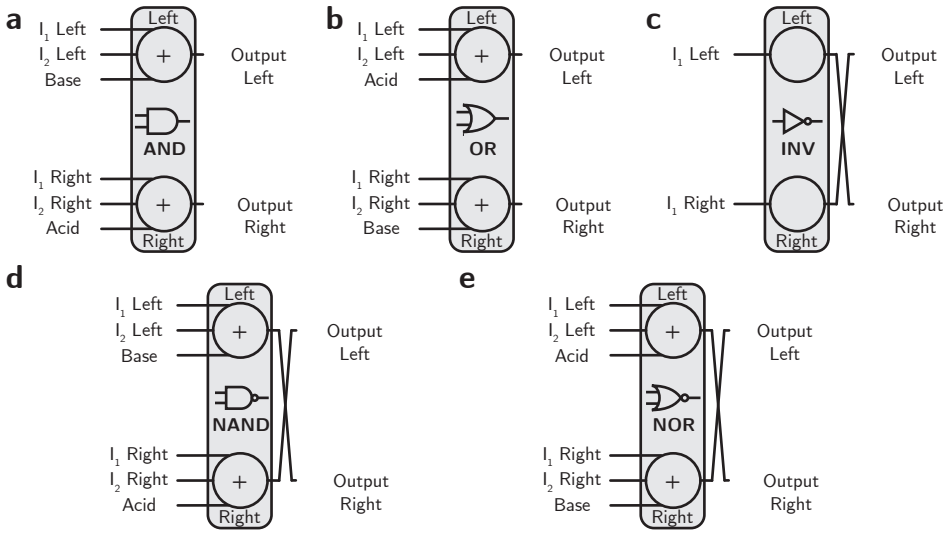


Figure 6.2: **Primitive logic gates represented by acid-base blocks. a** AND gate, **b** OR gate, **c** INV gate, **d** NAND gate, **e** NOR gate. Each gate with an even number of inputs requires a third biasing value to implement its function and eliminate neutrality. The + sign indicates that the input solutions are mixed. The crossing output lines indicate that an inversion has been performed.

Table 6.1: The truth table for the AND gate in digital systems and its corresponding representation in our dual-rail acid-base method.

Digital AND Gate			**Dual-Rail Acid-Base Operation**			
In 1	**In 2**	**Out**	**Input 1**	**Input 2**	**Bias**	**Output**
0	0	0	Base/Acid	Base/Acid	Base/Acid	Base/Acid
0	1	0	Base/Acid	Acid/Base	Base/Acid	Base/Acid
1	0	0	Acid/Base	Base/Acid	Base/Acid	Base/Acid
1	1	1	Acid/Base	Acid/Base	Base/Acid	Acid/Base

The first two columns correspond to the input permutations. The following fourth and fifth columns represent the acid-base encoding. The sixth column represents the constant bias input used with the AND gate. The last columns on either side show the result (output) of mixing the inputs in the acid-base AND gate.

Figure 6.3 depicts a complete logic circuit for the digital decoder using acid-base blocks. Figure 6.3a shows the original logic circuit, while Figure 6.3b shows the corresponding circuit using the equivalent acid-base logic blocks. Finally, Figure 6.4 depicts the implementation of the decoder circuit using the liquid handler. Figure 6.4a) shows the truth table for the 2-bit decoder circuit, Figure 6.4b) shows the first level of the circuit that includes the original input and its complements in addition to acid and base stocks to be used as biases for the AND gates, and Figure 6.4c) shows the final outputs of the circuits after applying the AND gate according to the circuit design.

One limitation of these acid-base logic gates is that the outputs of successive cascaded logic stages have progressively more neutral pH because of the neutralization reaction. Conceptually, one way to resolve this is to simply replace the intermediate results with fresh acid or base solutions. For simplicity this is what we

assume in logical simulations; however, it implies interrupting the computation. Other future alternatives to restore the pH logic levels could include incorporating mechanical valves with pH-sensitive materials, which we highlight in the Discussion.

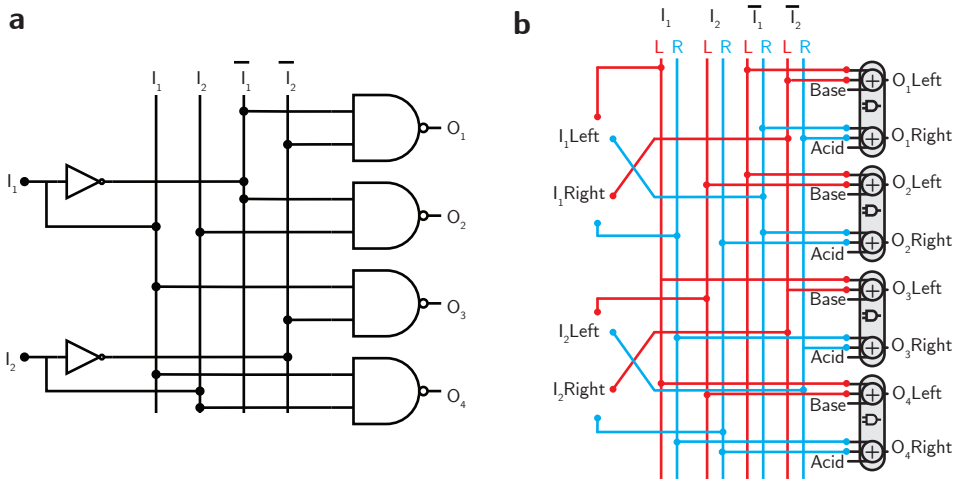


Figure 6.3: **Digital decoder and its equivalent acid-base implementation.** **a** The decoder circuit using digital logic blocks. **b** The equivalent decoder circuit using acid-base blocks.

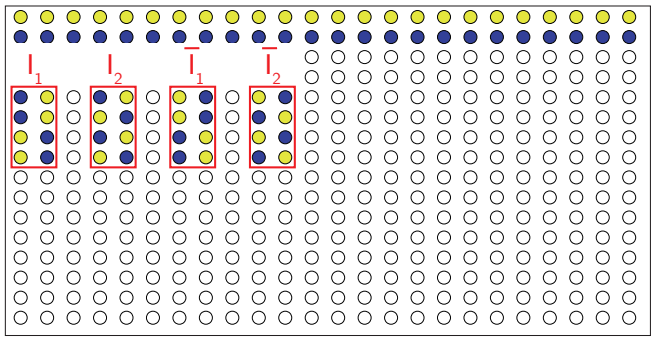
MNIST digit classifier using acid-base reactions. Neural networks are computing systems that derive relations from available datasets through weighted sums of input features executed through connected neurons. Classification is a common task for neural networks to classify inputs (e.g., handwritten digit image) into a set of classes (e.g., the corresponding digit number). Additionally, the network is customized for different conditions, such as 1) The input image format: colored, grayscale, or black and white. We ran our experiment on black and white, and 3-bit grayscale images. 2) The type of training weights. We chose binary weights, which are limited to "1" and "-1," since we can readily represent a "1" as a direct transfer of an input and a "-1" as an inverted transfer that flips our rails in our encoding. We used softmax [136] as an activation function for all our models. Finally, the neural network is split into two phases: a training phase and an inference phase. The training phase is a computationally-intensive phase that involves using a prepared dataset to find optimal weights that generate the desired labels with high accuracy. The inference phase uses the pre-trained weights to predict labels for new inputs outside of the dataset. Much like previous DNA computing demonstrations,[122][30] our work focuses on using acid-base chemistry to realize the inference phase based on pre-trained weights, which is the main phase in most practical applications. The acid-base encoding presented here computes the weighted sum performed by each neuron by mixing strong acids and bases that represent the encoded image based on the binary weights of the network. A black pixel is represented by a (Base, Acid) pair, while a white pixel is represented by an (Acid, Base) pair. Our approach performs the multiplication by "1" (through a direct liquid transfer without inversion) or by "-1" (through swapping the order of the encoded dual-pair value during the transfer). The resulting solution is acidic or basic, which maps to the classification result of the network where (Acid, Base) pairs represent positive outputs, while (Base, Acid) pairs represent negative outputs. Additionally, the pH indicator serves as a non-linear activation

layer that computes the neuron's output based on the pH threshold.

a

Inputs		Outputs			
I_1	I_2	O_1	O_2	O_3	O_4
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

b



c

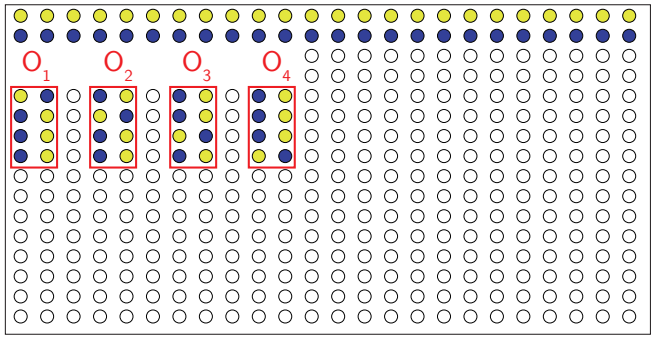


Figure 6.4: **Implementation of the 2-Bit Decoder using the liquid handler. a** The truth table for the 2-bit decoder. **b** The initial acid (yellow) and base (blue) stocks of the encoded inputs and their complements. **c** The final output of the acid-base circuit after applying the AND gate.

We trained a binary neural network consisting of two fully-connected neurons with the softmax activation function to recognize images that represent hand-written digits. In our experiments, we choose images corresponding to the hand-written 0 and 1 digits from the MNIST (Modified National Institute of Standards and Technology) digit classification dataset [37]. The pixels of the input image are flattened sequentially (left to right, and top to bottom) into a 1D vector. A copy of the 1D vector representation of the image is fed into each neuron. Figure 6.5 shows the 2-neuron network for the classification of 8×8 images into zero or one. After training the weights, we perform our classification as shown in Figure 6.6. Figure 6.6a) shows an

example image from the validation dataset. We start by encoding the image into its acid-base representation shown in Figure 6.6b) using the acid-base stock. We encode the image as many times as the number of neurons (classes) in our approach (we have two classes/neurons since we are classifying the digits "0" and "1"). After encoding and applying the weights, we compute the output from each neuron to decide whether it is a "1" or "-1." Finally, as shown in Figure 6.6c), for each neuron, we pool (sum) all of the pH rails of the dual-rail to compute the pH rail of the neuron's output, and all of the complementary pH rails of the dual-rail to compute the complementary pH rail of the neuron's outputs. We add a pH indicator to the neuron outputs to determine whether the outputs are acidic or basic based on their color. A neuron with an output pair (Acid, Base) indicates that the input image matches the digit this neuron is supposed to recognize. For instance, the first neuron is supposed to recognize the digit 0, while the second neuron is supposed to recognize the digit 1. In contrast, a neuron with an output pair of (Base, Acid) indicates that the input image does not match the digit this neuron is supposed to recognize.

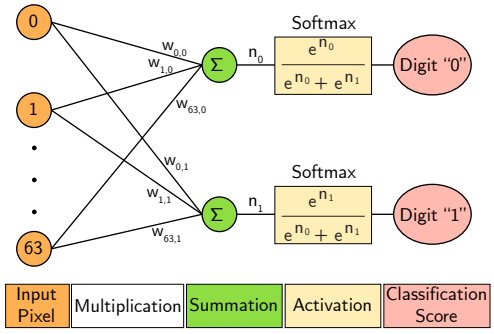


Figure 6.5: **Network architceture.** Neural network for the classification of 8×8 images into the digits zero or one.

We ran our model in simulation using various parameters (image size, discretization levels, and the number of classes) across the full validation set and calculated the match percentage between our experimental reactions and our theoretical pH simulations. Additionally, we picked ten random samples from the validation dataset to evaluate the model experimentally using the liquid handler. The minor reduction in accuracy from the acid-base model compared to the *in-silico* model arises from relying on the pH color indicator for reading the outputs. We rely on the color to determine the state of each neuron where (Acid, Base) = +1 and (Base, Acid) = -1; which is sufficient when the *in-silico* model produces scores that have a positive sign for the correct label and a negative sign for the wrong label. However, for a few samples, the *in-silico* model and the acid-base model will generate matching results, except that all outputs are either (Acid, Base) or (Base, Acid). However, when we measure the actual pH levels, the output with the highest concentration always matches the prediction label from the *in-silico* model. Supplementary Figure B.1 shows an 8×8 image example that cannot be classified correctly with the acid-base network; the pixel values for the image are shown in Supplementary Table B.1.

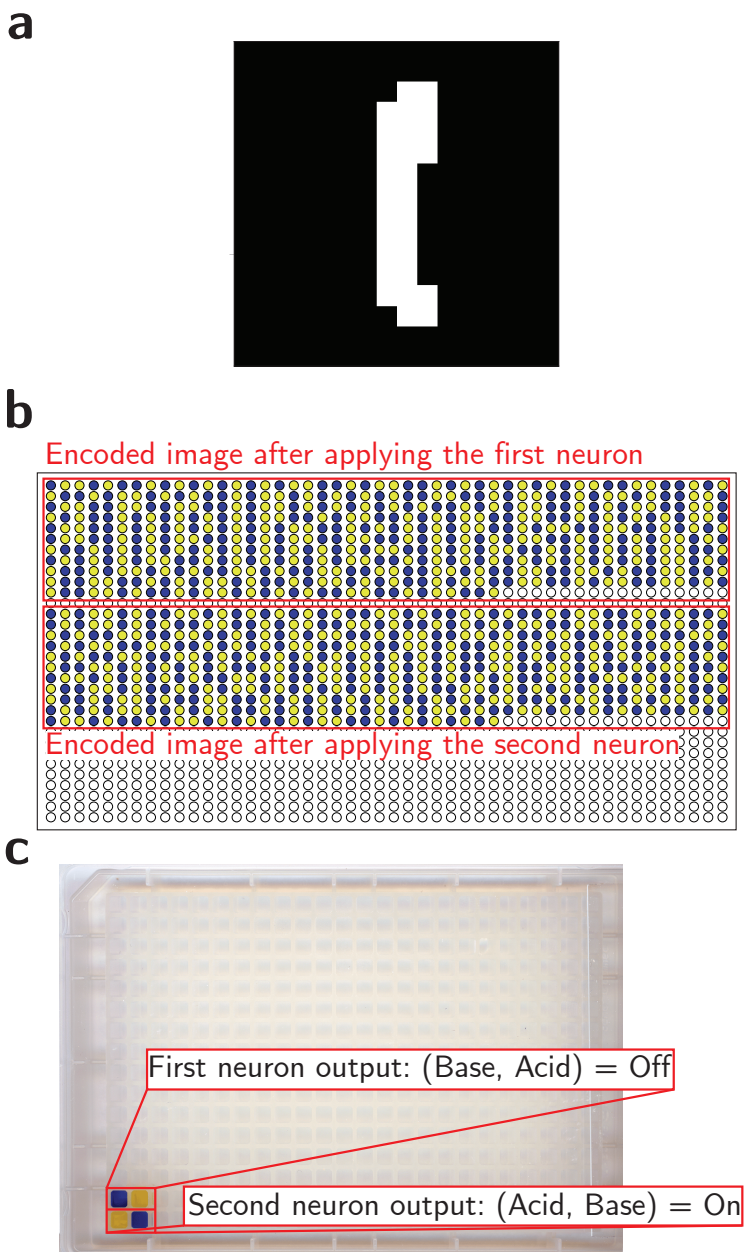


Figure 6.6: **Classification of the digit "1" using an acid-base network.** **a** The original 16×16 image to be classified. **b** The encoding of the image and inversions using acids (yellow) and bases (blue). **c** The final classification output image from the lab experiment.

For the first set of results, we ran our experiment using binary pixel values and two classes using dataset images of sizes 8×8, 12×12, 16×16, and 28×28. As shown in Table 6.2, our chemical-based classifier is able to match theoretical results from the digital network with 99.61% similarity on average across different input sizes. The model's accuracy is calculated over the validation dataset, and it is defined as the ratio of number of correct classifications to the number of samples in the validation dataset. The match similarity is

calculated over the validation dataset using pH simulation, and it is defined as the ratio of the number of of samples where the *in-silico* and acid-base networks match to the number of samples in the validation dataset.

Table 6.2: Matching accuracy for 2-class, binary image classifier.

Image Size	**Model Accuracy (%)**	**Acid-Base Network Match (%)**
8×8	95.45	99.81
12×12	95.59	99.39
16×16	95.82	99.39
28×28	96.00	99.86
Average	95.72	99.61

Image size, classification accuracy of the digital model, and matching accuracy between the digital model and the theoretical pH values. Source data are provided as a Source Data file.

For the second evaluation, we extended our experiment to classify 3-bit images (8 grayscale levels) instead of binary values. We use a network that is similar to the previous experiment but uses 3-bit values instead of binary ones. We discretized the input pixel values into eight different integers where each pixel value x is mapped to the integer: $round(8 \times \frac{x}{255.0}) - 4$. We then utilized the dilution equation explained before to represent the given integer. We re-trained the network with the updated representation. The remaining flow matches the previous experiment. As shown in Table 6.3, our chemical-based classifier is able to match the theoretical results from the digital network with 98.94% similarity on average across different input sizes.

Table 6.3: Matching accuracy for 2-class, 3-bit image classifier.

Image Size	**Model Accuracy (%)**	**Acid-Base Network Match (%)**
8×8	93.28	97.26
12×12	95.15	99.15
16×16	96.28	99.48
28×28	96.37	99.86
Average	95.27	98.94

Image size, classification accuracy of the digital model, and matching accuracy between the digital model and the theoretical pH values. Source data are provided as a Source Data file.

For the final set of results, we extended our experiment by using three fully-connected neurons instead of two to classify binary images belonging to three different classes/digits (“0”, “1”, and “2”). As shown in Table 6.4, our chemical-based classifier is able to match the theoretical model’s results from the digital network with 98.3% similarity on average across different input sizes.

Table 6.4: Matching accuracy for 3-class, binary image classifier.

Image Size	**Model Accuracy (%)**	**Acid-Base Network Match (%)**
8×8	95.68	96.12
12×12	95.86	98.82
16×16	96.00	98.82
28×28	96.05	99.62
Average	95.90	98.35

Image size, classification accuracy of the digital model, and matching accuracy between the digital model and the theoretical pH values. Source data are provided as a Source Data file.

6.3 Discussion

In this work, we explored the usage of acid-base reactions as a basis for complex computation orchestrated by a robotic fluid handler to perform the encoded reactions. We introduced a dual-rail encoding method that suits the natural complementarity of acids and bases and enables our method to support the negation operation, which is essential for realizing universal computation. We leveraged the fact that the pH of a strong acid/base system is dominated by whichever species is more concentrated to realize the majority function in chemistry. Our approach presents a system for modeling various computations without using the complex spatio-temporal encodings presented in other unconventional computing methods. Furthermore, the reactions enable rapid computation, given the speed of strong acid/base reactions, without expensive equipment or intricate handling to execute the operations. Straightforward computation using acid-base chemistry can provide new ways of interacting with molecular data storage systems, for example, by interacting with pH-dependent protection/deprotection groups that could be incorporated into DNA [114, 122] or small-molecule information [14, 84, 128] storage systems. As an illustration of our approach, we implemented an image classification neural network using a system of acid-base reactions. We utilized binary neural networks and binary cross-entropy loss functions to generate complementary outputs compatible with our method that can be distinguished using a pH indicator. As a result, we managed to classify the MNIST digits dataset (for two and three digits). Simulated experiments based on our approach classified images with a 99% accuracy. We also extended our encoding and computational approach to work with non-binary inputs. Using the dynamic range of possible concentrations, we can generate more interesting representations for our data to support more efficient computations [15]. Looking forward, we are investigating the possibilities of other nonlinear chemistries. This work presented only strong acids/bases, but incorporating weaker acids and bases as pH buffers [21] could introduce nonlinearity, thresholds, and plateaus in the pH curves. Introducing nonlinear pH operators could produce more complex primitives, such as activation functions [13], which are a critical part of multilayer neural networks. By developing a cascadable chemical nonlinear activation function, we would be able to extend our method to support deep neural networks with multiple hidden layers. In addition to neural networks, we also presented an approach for using the acid-base reaction to represent digital logic circuits. We designed multiple digital logic primitives using the acid-base system while introducing a third biasing input to the blocks to guarantee a non-neutral output and the desired functionality. Since acid-base reactions can be diluted when propagated across multiple layers or gates, we proposed solving

dilution problems by manually replacing the intermediate results with fresh non-diluted stocks of acids or bases. However, this step can be automated by combining with other existing concepts, such as the usage of pH-sensitive hydrogel valves [171] to build a restoration mechanism to maintain the computation across a longer chain of reactions. Additionally, digital logic optimization algorithms can be incorporated to improve the quality of our approach. For example, we can optimize the computations to reduce the number of logic layers in the digital circuit to minimize the dilution that results from cascading multiple reactions during the computation. Furthermore, our procedures can be incorporated into other binary systems of a similar nature. For example, the Hadamard transform [66] uses operations analogous to those employed in this work. Our dual-rail representation provides a methodology to solve the challenges of representing negation in chemistry, that can be extended beyond acid-base reactions. A complementary representation can serve as a useful alternative for any computational system that can not readily negate values. Furthermore, the relationship between the two (or more) rails does not have to be inverse but rather any desired relationship hard to realize in a single rail system. For example, the associated rails can represent such operations as multiplication by a scalar or a pre-computed trigonometric function.

6.4 Methods

Materials and reagents. Hydrochloric Acid (HCl, 36.5% to 38.0%, Fisher Chemical) and Sodium Hydroxide (NaOH, $\geq 97\%$, Fisher Chemical) were dissolved using deionized water as a solvent (Millipore Milli-Q) to a concentration of 100 mM. Bromothymol blue ($C_{27}H_{28}Br_2O_5S$, 0.04%, VWR Chemicals) was used as an indicator for the pH of the solutions. Deionized water was made available to the liquid handler for dilution of the reagents in the 3-bit image neural network classification experiments.

pH simulator. In order to verify our results on the full dataset, an automated pH simulator was designed to simulate the reaction outcomes for the full dataset. The simulator predicts the pH by calculating the hydrogen ion concentration [H^+] that results from mixing the two or more aqueous solutions involved in our computations. The simulator was designed to directly accept the liquid handler's programming files to recreate the liquid handler's transfers and eliminate related human errors.

Data preparation and network training. The MNIST dataset was used for training the acid-base digit classifier due to its well-understood properties and clear benchmarks. The dataset provides 70,000 grayscale hand-written digits ("0"-"9") of size 28×28 (784 total values, where each value is between 0-255). Data points for the digits "0", "1", and "2" were used for the network including 5923, 6742, and 5958 points for training, and 980, 1135, and 1032 points for validation, respectively. The whole dataset was used for training and simulation *in silico*, while a random sample from the validation dataset was used for the experimental verification as described in the next section. Each experiment was evaluated using four different image scales: 8×8, 12×12, 16×16, and 28×28. Two variants were provided for each image: the binary variant where each pixel was given a value of 1 or -1 based on a threshold of 128 where pixels of intensity less than 128 were assigned an encoding of (Base, Acid) while pixels of value greater than or equal to 128 were assigned an encoding of (Acid, Base), and the 3-bit variants where each pixel was discretized to a value from {-4, -3, -2, -1, 1, 2, 3, 4} representing the pixel intensity ranges {0:32, 33:64, 65:96, 97:128, 129:160, 161:192,

193:224, 225:256}, respectively. The same processed dataset image files were also used by the pH simulator to compute the accuracy of the acid-base experiments by computing the resultant pH after running each image through our acid-base neural network and comparing it to the expected classification. The dataset was split into training and validation sets according to an 85/15 ratio. The neural network model was trained with the weight binarization technique by Courbariaux *et al.* [32] to constraint the weights to "+1" and "-1", as explained above. A learning rate of 0.001 and a $sigmoid$ activation function were used to train a neural network consisting of one classification layer for 30 epochs. In addition, the Binary Cross Entropy was used as a loss function. After training multiple networks for the different permutations of the image scale and data scale variants, the weights were exported into text files to be used to guide our acid-base experiments.

Experimental verification of the acid-base neural networks. The experiments were conducted using the acoustic liquid handler. The handler was provided with an initial stock 384-well polypropylene plate (minimum volume 10 μL, maximum volume 60 μL) that is manually prepared with rows of acidic solutions (HCl) followed by rows of basic solutions (NaOH) and then by rows of deionized water. Each well was filled with 50 μL of the designated solution. The liquid handler has a maximum deviation of 10% from its target volume. For the first phase, using the image exported from the processed MNIST dataset, the liquid handler is programmed to encode the image using a series of transfers from the initial stock plate into a 1536-well low dead volume plate (minimum volume 1 μL, maximum volume 4 μL). The process is repeated according to the number of neurons in the neural network model. For each pixel in the image, if the value is negative and the neuron weight is +1 or the value is positive, and the neuron weight is -1, the handler transfers 2.4 μL of basic solution into the first empty well, followed by 2.4 μL of acidic solution into the next well. Similarly, if the value is negative and the neuron weight is -1 or the value is positive, and the neuron weight is +1, the handler transfers 2.4 μL of acidic solution into the first empty well, followed by 2.4 μL of basic solution into the next well. Afterward, if more values other than "1" and "-1" are supported, the handler dispenses quantities of the deionized water, as described in equation 6.5, into the corresponding wells in the destination to dilute the solutions according to their discretized pixel values.

The final phase is computing the output of each neuron by pooling the pH and complementary pH rails of each dual-rail pair. The summation is carried out from the encoded image in the 1536-well plate back into the 384-well plate. For each neuron, we transfer 200 nL from the pH rail of each pixel's dual-pair (starting from the first well, scanning to the end of the neuron wells, and skipping every second well) into the well assigned to the pH rail of the output of that neuron. We do the same for the well representing the complementary pH rail of each pixel's dual pair. The overall time to pool the neuron outputs was around 5 minutes. Table 6.5 gives the breakdown of the run-time of the two stages of the experiment: image encoding and result pooling. The shown times are for the three different experiments presented: 2-class classification of binary images, 3-class classification of binary images, and 2-class classification of 3-bit images. The run-time of the experiment depends on the number/volume of liquid transfers needed to execute the experiment. The 2-class binary Image takes the same amount of time as the 3-bit grayscale variant since the transferred volumes are equivalent (the 3-bit grayscale variant only substitutes some of the transferred acid/base with water for dilution). On the other hand, the 3-class variant has an extra run-time overhead since it requires more liquid transfers for the extra neuron (the 3-class network has three neurons, compared to two neurons

for the 2-class network). Supplementary Figure B.2 shows an illustration of encoding 3-bit grayscale images using the approach described earlier. The color of each well corresponds to the pH color shown in Figure 7.1bb. All the validation datasets with different scale permutations were verified by running through the pH simulator for theoretical validation (without noise or experimental errors). We selected ten random images from the validation dataset for the real experimental evaluation by the Echo liquid handler as follows: one 8×8 binary image (for the 2-class network), one 8×8 binary image (for the 3-class network), one 12×12 binary image (for the 2-class network), one 12×12 binary image (for the 3-class network), and two 16×16 binary images (for the 2-class network) two 16×16 3-bit images (for the 2-class network). Supplementary Figure B.3 show a walkthrough of the experiment's setup. Additionally, we have performed an experiment of digital networks on a different platform described in the appendix.

Table 6.5: The run-time in minutes for a single input of the three experiments: 2-class classification for binary images, 3-class classification for binary images, and 2-class classification for 3-bit images.

Image Size	**2-Class Binary Image**		**2-Class 3-Bit Image**		**3-Class Binary Image**	
	Encoding	**Pooling**	**Encoding**	**Pooling**	**Encoding**	**Pooling**
8×8	13 min	1 min	13 min	1 min	19 min	2 min
12×12	29 min	3 min	29 min	3 min	44 min	4 min
16×16	52 min	5 min	52 min	5 min	78 min	8 min
28×28	158 min	16 min	158 min	16 min	237 min	24 min

Reading the classification values. The final phase of the experimental validation is reading out the neuron output. 5 μL of the Bromothymol blue pH indicator were added to the dual-rail output of each neuron; a blue color indicated a basic solution, while a yellow color indicated an acidic solution. Each neuron's dual colors represented its final classification where a neuron's output of (Acid/Yellow, Base/Blue) indicated it is a "1" (the predicted classification) while (Base/Blue, Acid/Yellow) indicated it is a "-1."

6.5 Data availability

Source data are provided with this chapter. The processed MNIST datasets are available at https://doi.org/10.6084/m9.figshare.21753545.v4[6].

6.6 Code availability

The software is available at https://doi.org/10.6084/m9.figshare.21753551.v4[7].

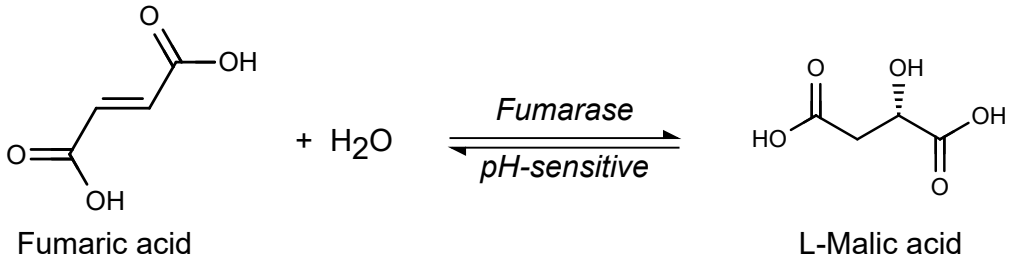
Chapter 7

Enzymatic Computing for Parameter-Efficient Analog Neural Networks and Digital Circuits

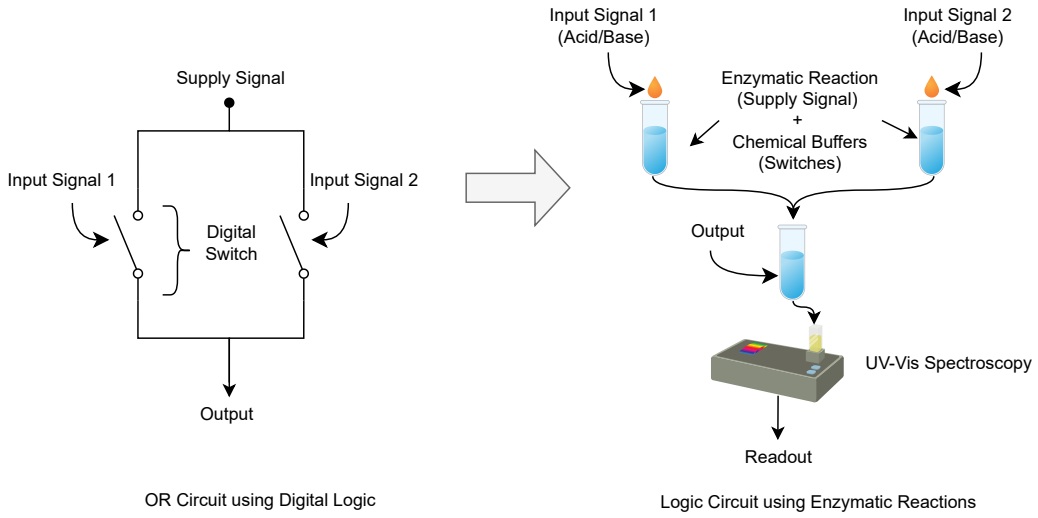
7.1 Introduction

The evolution of computing technology has experienced several paradigm shifts, transitioning from the initial mechanical systems to the current advancements in electronic computing. These developments have paved the way for exploring unconventional methods of information processing. Unconventional computing explores novel computing mechanisms, often inspired by biological, chemical, or physical phenomena [2, 1, 19, 3, 52]. While some advancements, such as DNA computing and quantum computing, utilize the intrinsic capabilities of DNA molecules for data storage and the principles of quantum mechanics for enhanced computational power [44, 23, 126, 134, 124], there remains a vast potential in other biological systems. Among these, we observe that enzymes offer a unique opportunity for developing computational systems due to their inherent biological properties and behaviors.

In this chapter, we introduce a computational model based on enzymatic reactions. Enzymes are catalysts that accelerate biochemical reactions and offer a high degree of specificity [47]. They are not only environmentally benign but also demonstrate energy-efficient characteristics [5]. Additionally, their kinetic properties can be influenced and controlled by external factors such as pH levels, which motivates our approach to using pH-controlled enzymatic reactions for information processing. Building on this premise, we use the enzyme fumarase to build our system. Fumarase is an enzyme involved in the citric acid cycle, catalyzing the hydration of fumarate to L-malate, as shown in Fig. 7.1a. This reaction is a critical step in the metabolic processes that contribute to energy production within cells. Fumarase's activity is influenced by the pH of its environment [48], a feature that allows for the regulation of the reaction's direction based on pH levels. The enzyme's response to pH changes enables a degree of control over its catalytic function, making it a subject



(a) Conversion of fumaric acid to L-malic acid catalyzed by the fumarase enzyme.



(b) Modeling logic circuits using Enzymatic reactions.

Figure 7.1: (a) Enzymatic reaction for converting fumaric acid to L-malic acid. (b) Overview of modeling logic circuits using enzymatic reaction showing a traditional circuit for an OR gate comprised of two switches compared to the same circuits modeled using enzymatic reactions. The enzymatic reaction and the buffer provide the switching mechanism; the substrate represents the supply signal, the acids and bases are used to model the input signal, and the reaction product models the output signal. The final readout is done using UV-Vis spectroscopy to detect the presence or absence of the product of the enzymatic reactions (indicating an output of 1 or 0, respectively).

of interest for applications extending beyond its biological role. In this study, we utilize the pH-sensitive nature of fumarase to develop a computational model. In our system, we promote the forward reaction by elevating the buffer's pH above 7.2, while lowering the pH results in inhibition or reversal of the reaction. We use acids and bases for signal encoding to modulate the pH of the reaction medium, ensuring controlled activation or inhibition of fumarase activities to model computational logic. Building on this foundation, we propose a new class of digital logic circuits and analog computing. A major advantage of our proposed method compared to many existing approaches [9, 82] is decoupling the propagated supply signal (modeled as enzymatic reaction's substrate) and the input signals (encoded as acids and bases), providing extensibility

in modeling logic functionalities and cascading computations. Moreover, in computational systems, it is often preferred to have a non-linear relation between the inputs and the outputs for its ability to enable functions beyond what is achievable with simple linear operations, such as the mere addition of two inputs or reagents. In digital circuit design, the ability to incorporate non-linear relationships between inputs and outputs facilitates the creation of more complex and adaptable computing architectures [58, 159]. Our model introduces non-linearity through two components. The first involves chemical buffers that host the enzymatic reactions. These buffers are characterized by attributes such as buffering capacity and range, which impart a non-linear relationship with the input that affects the solution's pH. Additionally, the enzymatic reactions themselves contribute to the non-linearity of the system. The activity of enzymes, including fumarase, demonstrates a non-linear relation to the pH of the medium[48]. We provide more details about the non-linearity of our system in the supplementary material. Our contributions can be summarized as follows:

- We introduce a novel computational paradigm that uses pH-sensitive enzymatic reactions in a chemical buffer to model digital logic gates. Fig. 7.1b shows that traditional logic circuits are comprised of a supply signal (e.g., power signal) that gets propagated through electronic switches controlled by the input signals. Similarly, in our model, the enzymatic reaction models a switching mechanism; the substrate acts as the supply signal, while the acid/base encoding models the input signal that controls the pH of the chemical buffers, which regulates the direction of the enzymatic reaction (supply signal). For the final readout, we apply an established detection method to generate NADH if the forward enzymatic reaction is successful. We use UV-vis spectroscopy to detect the presence or absence of NADH (which indicates an output of 1 corresponding to the success of the enzymatic reaction or 0 otherwise).

- We formulate the problem of finding the proper encoding for the input signals (the acids and bases that are used to represent the signal), and we propose a corresponding optimization approach. Our method identifies the concentrations of acids and bases to control pH levels in the buffer solution (promoting or inhibiting enzymatic reactions) to align with the behavior of the digital design.

- We experimentally model different logic circuits as enzymatic computation using our framework. We also demonstrate examples of cascading logic gates. We implemented and evaluated the different circuits in the laboratory conducted using chemical reactions.

- We extend our enzymatic computation framework to design and model a machine-learning classifier that we carry out through chemical reactions conducted in the laboratory. The implemented model performs binary classification of data points into positive or negative groups based on their respective coordinates. Our framework translates the traditional weights and inputs of a neural network perceptron into concentrations of chemical buffers while the acids and bases are used to encode the input coordinates.

7.2 Related Work & Preliminaries

Molecular Computation

Molecular computation extends beyond DNA computing to encompass a broad spectrum of molecular interactions used for computational tasks. This field exploits the specific and affinity-driven interactions of various molecular substrates, including peptides, proteins, and small molecules like neurotransmitters, to perform complex computations [133, 72, 147]. While these molecules offer unique computational capabilities due to their inherent properties, they often lack the robustness or flexibility required for complex computational architectures.

For instance, peptides and proteins are particularly notable for their structural versatility, which allows them to fold into specific three-dimensional shapes that can act as molecular computing devices. Researchers have utilized these properties to develop protein-based logic gates and self-assembling peptide structures, which respond to external stimuli to execute computational operations [107, 133]. However, these systems typically face challenges in stability and complexity, which are crucial for computing applications. Intrinsically Disordered Proteins (IDPs) showcase another avenue for unconventional computing, offering unique features due to their conformational heterogeneity. Unlike structured proteins, IDPs lack a rigid 3D structure and exist as dynamic ensembles capable of sampling a range of conformations. This flexibility allows IDPs to exhibit context-dependent behavior and physicochemical multiplicity, features that are particularly suitable for implementing fuzzy sets and processing fuzzy logic [88, 51]. The ability of IDPs to adapt their conformation in response to environmental factors provides flexibility and adaptability in computing systems, suggesting a promising direction for further exploration in molecular computing. Another example is small molecules, including neurotransmitters, which have been used to mimic neural network behaviors in bio-inspired computing systems [65]. These molecules can participate in signal transduction pathways, serving as messengers that relay information within and between cells. By harnessing these signaling molecules, researchers have developed computational models that mimic neural networks, leveraging the natural communication pathways of the brain for information processing. This approach highlights the computational capacity of small molecules and their potential for creating bio-inspired computing systems that emulate the efficiency and complexity of biological processes. Nevertheless, the complexity and control of these molecular interactions remain limited, restricting their application to simple signal-processing tasks. Moreover, the field of molecular computing has been enriched by the integration of synthetic biology techniques, which enable the design and construction of novel biological circuits with predefined functionalities. This has led to the development of biosensors, memory storage devices, and even computational systems capable of decision-making processes based on molecular inputs [85, 158, 120, 129, 14, 13]. Enzymes, such as Fumarase, provide a promising alternative to other biomolecules due to their specificity and efficiency. Enzymes are highly specific to their substrates, which reduces the likelihood of side reactions and allows for precise control over the biochemical reactions necessary for computation [47]. Additionally, enzymes operate effectively under controlled conditions where parameters such as temperature, pH, and ionic strength can be regulated to enhance their activity and stability. This specificity and controllability make enzymes

suitable for constructing logic gates and performing complex computational tasks. However, the use of enzymes as practical components in computing devices also presents challenges. Enzymes can be sensitive to environmental changes, leading to issues with stability and consistency over long periods. However, there are also techniques to chemically modify enzymes to enhance their stability, which remains a potential avenue for future research to further improve enzyme stability for practical applications [16]. While there are some criticisms of molecular computing based on chemical inputs, such as the progressive dilution of solutions and the difficulty in cascading logic gates, our approach tries to mitigate these issues. The use of buffered solutions maintains consistent pH levels, minimizing the impact of dilution. Additionally, the non-linearity in our model makes the threshold values more reliable. By decoupling the supply signal from the input signal using enzymatic reaction substrates and acids/bases, our method mitigates signal neutralization, facilitating the construction of more complex binary circuits. For example, existing work [9, 82] demonstrates modeling a single gate or a decoder circuit (which does not involve cascading computation) since, beyond the first gate, the acid/base input/supply signals can get neutralized (destroying the propagated signal), and only the biasing input signal is left to represent the output. However, in our method, we model the supply signal using the enzymatic reaction's substrate while the input signals are encoded as variations in acid/base concentrations that modulate the enzymatic activity by altering the buffer's pH, which reduces the effect of dilution or neutralization, providing further flexibility and extensibility for the system as we demonstrate cascading more than one gate.

Enzymatic Reactions

Enzymes serve as biological catalysts, enabling chemical reactions to proceed at rates that would otherwise be unattainable under mild conditions. Governed by the principles of enzymatic kinetics, these biomolecules have been widely studied in biochemistry and molecular biology. Their ability to facilitate reactions is controlled by a range of factors, including but not limited to temperature, ionic strength, and pH levels. However, their potential use in computational systems has been less explored. While enzymes offer the advantage of highly specific and energy-efficient catalytic actions, their sensitivity to various environmental conditions presents challenges that need to be carefully managed.

Digital Circuits

Digital circuits serve as the fundamental infrastructure powering all electronic systems. These circuits operate on the principles of Boolean algebra, employing binary logic where signals are designated as either "0" or "1". The elemental components in these circuits are logic gates, such as AND, OR, and NOT gates, which execute basic Boolean functions. These gates accept binary signals as inputs and produce a binary output based on predetermined logical operations. Complex digital systems are constructed by assembling these basic gates into hierarchical arrangements that can perform sophisticated computational tasks. Over the years, technological advancements have facilitated the miniaturization and increased efficiency of these circuits, contributing to the development of electronic devices that are both potent and compact. Although traditional digital circuits predominantly employ silicon-based components, the limitations inherent to these materials

have spurred interest in alternative computational paradigms [19, 162, 54, 9]. Among these alternatives, systems modeled on biological or chemical phenomena are subjects of active research, as they offer novel methods of signal processing and computation.

7.3 Results & Discussion

Modeling Logic Gates using Enzymatic Reactions

As mentioned earlier, our approach utilizes the enzymatic conversion of fumarate to L-malate, catalyzed by the fumarase enzyme as shown in Equation 7.1, as a biochemical proxy for switches in digital logic gates. This enzymatic reaction offers a convenient feature: its directionality is modulated by the pH level of the chemical buffer in which it occurs [48].

$$\mathrm{Fumarate} + \mathrm{H_2O} \overset{\text{fumarase}}{\rightleftharpoons} \mathrm{L{-}malate} \tag{7.1}$$

The initial step is to map each switch in a conventional digital logic gate to a chemical buffer. Each buffer, containing the enzyme and its substrates, acts as a medium for the enzymatic reaction (the supply signal). Initially, these buffers are prepared with a pH value situated approximately at $\mathrm{pH} = 6.8$, the midpoint of the ranges that favor forward and reverse reactions. This intermediate pH serves as the starting point, from which we can shift the reaction's course by adding either strong acids or bases (the input signals). For the realization of these enzymatic gates, the concentration levels of strong acids and bases, corresponding to the binary signals "0" and "1", are determined through an optimization process that we explain in the following subsection. The concentrations thus derived serve as the inputs for each chemical buffer. The introduction of these strong acids or bases drives the enzymatic reaction either forward or backward, in accordance with the pH-modulated reaction kinetics. Subsequent to this input phase, the reactions are propagated from one buffer to the next, following the switches' connections in the original digital design. The enzymatic reactions mimic the signal flow in digital design, moving from one buffer to the next in a manner analogous to signal transmission across switches in the conventional circuit. The presence or absence of the malic acid at the terminal buffer serves as the chemical analog of the binary output signal ("0" or "1") from the digital circuit.

Computing the Concentrations for the Input Signals

An important aspect of our system is the computation of concentrations necessary for representing the signals "0" and "1". The goal is to adjust the input signal, in terms of concentration, to make the buffer fall within a pH range that either promotes or inhibits the forward enzymatic reaction, corresponding to the desired outputs of "1" or "0", respectively. To model the buffer's pH, we employ the Henderson-Hasselbalch equation, shown in Eq 7.2, which serves to calculate the expected pH of the buffers where pKa represents the acid dissociation constant of the material forming the buffer, $[HA]$ is the concentration of the acid, $[A^-]$denotes the concentration of the conjugate base, and pH is the final buffer's concentration.

$$\mathrm{pH} = \mathrm{pKa} + \log\left(\frac{[A^-]}{[HA]}\right) \tag{7.2}$$

Determining the requisite concentrations is approached as a linear programming problem, where the objective is to identify values that satisfy a series of inequalities that describe the target pH range for each input pattern. For instance, consider a basic design incorporating a single buffer (switch) for an inverter circuit, which necessitates that an input of "0" produces an output of "1", and an input of "1" results in an output of "0". Let $\text{pH}_{\text{forward}}$ be the minimum pH required for the enzymatic reaction to proceed (so we need the buffer's pH to be greater than $\text{pH}_{\text{forward}}$ to execute the forward reaction), and $\text{pH}_{\text{reverse}}$ be the pH at which the reaction reverses. $\Delta[OH^-]$ represents the change in concentration of $[OH^-]$ ions when applying the concentration of input signal 0, while $\Delta[H^+]$ represents the change in concentration of $[H^+]$ ions when applying the concentration of input signal 1. The objective is for a "0" signal to set the buffer's pH within the $\text{pH}_{\text{forward}}$ range and conversely for a "1" signal. To find these concentrations, we solve the inequalities 7.3 and 7.4. Realizing the envisioned enzymatic circuit relies on the feasibility of solving these inequalities.

$$\text{pKa} + \log\left(\frac{[A^-] + \Delta[OH^-]}{[HA] - \Delta[OH^-]}\right) \geq \text{pH}_{\text{forward}} \tag{7.3}$$

$$\text{pKa} + \log\left(\frac{[A^-] - \Delta[H^+]}{[HA] + \Delta[H^+]}\right) \leq \text{pH}_{\text{reverse}} \tag{7.4}$$

As the design complexity increases with additional buffers, the inequalities are expanded to model the buffer states in response to varying input patterns. While we can try to find the concentrations experimentally, we prefer to formulate the problem based on the characterization of the enzyme, making it easier to design different gates. Using this formulation, the concentrations are determined by solving the given inequalities. One method to solve these inequalities is through iterative optimization techniques through neural networks and gradient descent, which are further detailed in the supplementary material.

Signal Readout

The output in our system is defined as the presence or absence of the reaction's product, which also acts as a thresholding mechanism. Hence, we measure the system's state by detecting the presence or absence of the final output (as a binary state) defined by the absorbance level at 340nm. We use the common method of detecting L-malic acid by applying assay reactions to generate NADH (which can then be detected through UV-Vis Spectroscopy). The reaction details are highlighted in the supplementary material. After the reaction, an increase in absorbance at 340 nm serves as a reliable proxy for the presence of L-malic acid and, by extension, the binary output of the original logic gate. This readout mechanism provides a quantifiable method for translating the chemical activity into discernable digital signals. Moreover, alongside qualitative assessments derived from the absorbance curve, we established a quantitative method to analyze the presence of NADH. Considering the baseline absorbance at 340nm typically sits at around 0.2, and NADH's peak absorbance occurs near 0.4 at 340 nm, our goal is to refine these observations to binary-like values, closely mapping the presence or absence of NADH to 1 or 0, respectively. To facilitate this, we employ the following equation for quantifying the output based on the absorbance at 340 nm:

$$\text{Absorbance Score} = |A_{340\text{nm}} - 0.2| \times 5.0 \tag{7.5}$$

In this formula, $A_{340\text{nm}}$ represents the measured absorbance at the 340 nm wavelength. Applying a threshold value of 0.5 allows for a binary interpretation of the data: readings below 0.5 are classified as indicating an absence of NADH (quantified as 0), and those above 0.5 as indicating its presence (quantified as 1). Thus, an absorbance measurement at 340 nm close to 0.2 translates to an absorbance score near 0, reflecting minimal or no NADH presence. Meanwhile, an absorbance measurement at 340 nm around 0.4 results in an absorbance score close to 1, indicating a significant presence of NADH. This approach enhances the precision of our model's output interpretation by quantitatively reflecting the enzymatic reaction outcomes.

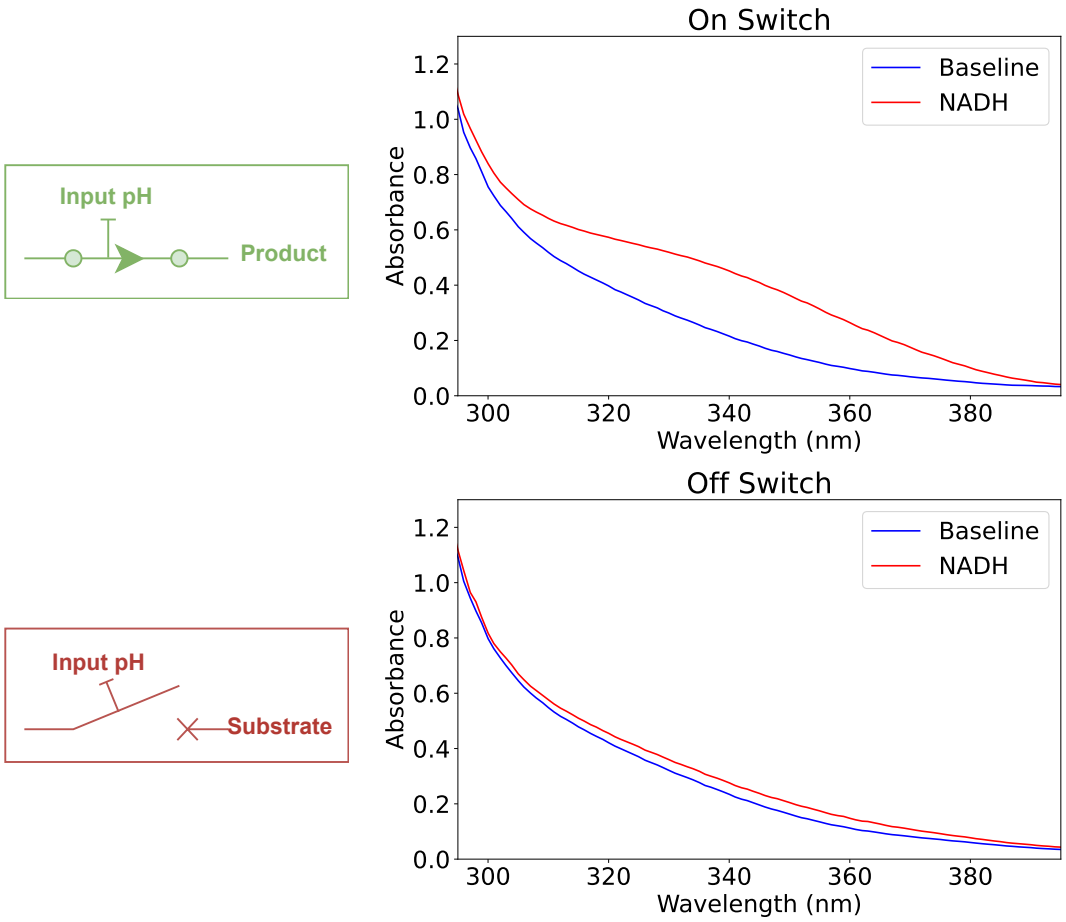


Figure 7.2: UV-vis spectroscopy for the outputs of a switch mechanism modeled using enzymatic reactions.

Modeling Switch Mechanism

The concept of a digital switch is modeled by using a single enzymatic reaction. In this model, the binary state of the switch is determined by the presence or absence of L-malic acid, with its presence indicating an "on" state ("1") and its absence denoting an "off" state ("0"). We use one buffer solution to modulate the enzymatic reaction, where the application of a strong base or acid determines the switch's state. Specifically, the introduction of a strong base into the buffer induces a pH increase, catalyzing the enzymatic reaction to

State	pH Encoding	Abs.	Abs. Score	Normalized
On	14.00	0.427	1.137	**1**
Off	0.60	0.215	0.077	**0**

Table 7.1: Absorbance score for modeling a switch mechanism using enzymatic reactions. The "State" column indicates the state of the switch (on or off). The "pH Encoding" column indicates the pH used to set the switch into the corresponding state. The "Abs." column lists the measured absorbance at 340 nm corresponding to NADH presence. The "Abs. Score" is calculated from the absorbance using the provided equation, and the "Normalized" column presents the final binary output after applying the thresholding value.

produce L-malic acid and effectively turning the switch "on." Conversely, adding a strong acid decreases the pH, inhibiting the enzymatic reaction and switching the device "off." Fig. 7.2 demonstrated the experimental validation of the model showcasing controlling the switch state, while table 7.1 shows the corresponding Absorbance Scores.

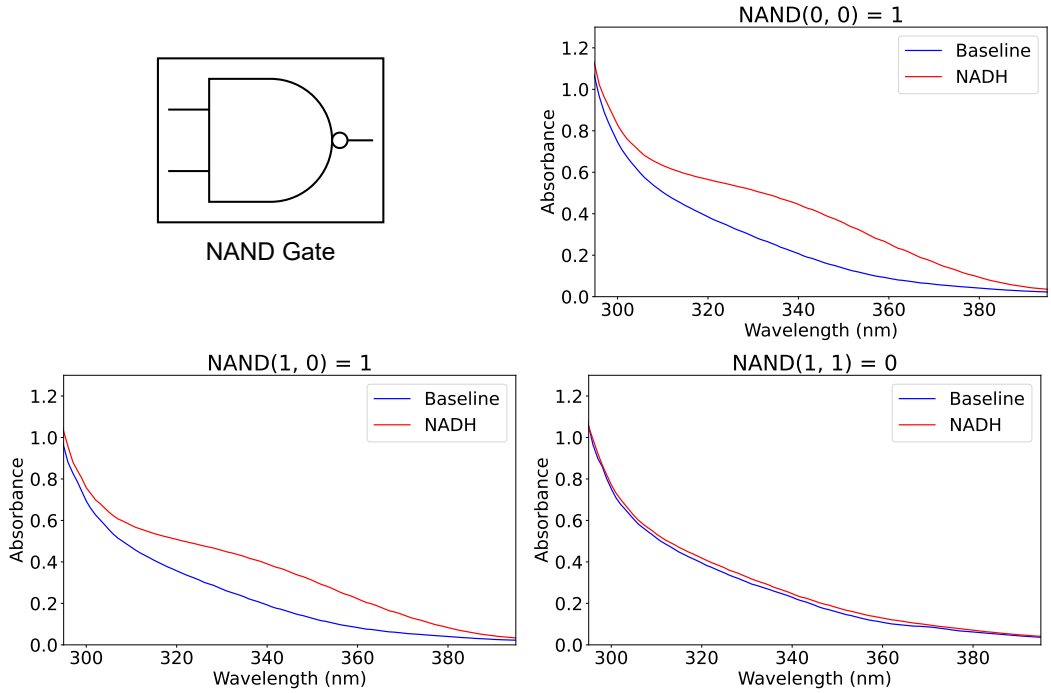


Figure 7.3: UV-vis spectroscopy for the outputs of the NAND circuit modeled using enzymatic reactions.

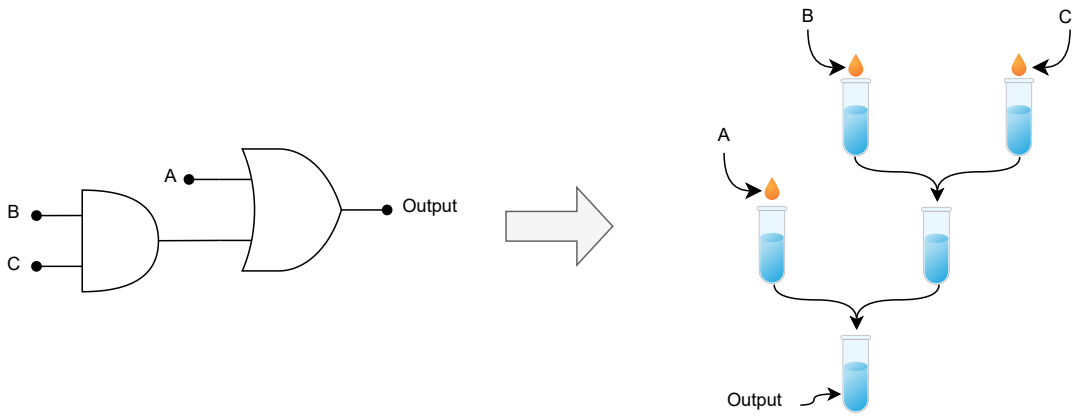
Modeling NAND Gate

The NAND gate, a fundamental two-input logic gate, is designed to output a "1" for all input combinations except when both inputs are "1". To model this behavior using our enzymatic reaction framework, the input signals are configured to inhibit the enzymatic reaction when both are "1", allowing the reaction to proceed in all other cases. The encoding of binary signals into chemical concentrations uses a pH of 14.27 for encoding a signal of value "0" and a pH of 0.62 for encoding a signal of value "1". This approach ensures that the

enzymatic reaction is selectively inhibited based on the input combination, mirroring the logical function of a NAND gate. The reactions were carried out in $2.5mL$ of phosphate buffer with an initial concentration of $1.01gm/m$L containing $100\mu L$ of the Fumarase enzyme and $128\mu L$ of the Fumaric acid. The encoded signal was applied using $10\mu L$ of the given concentration; we expect the pH of the buffer to be above 7.2 (promoting the forward reaction) for all cases except for the input signals of all "1s". Hence, Fig. 7.3 illustrates the absorbance curves for the outputs generated by our enzymatic NAND gate for various input patterns, demonstrating the gate's ability to replicate the expected logical outcomes through biochemical means. Additionally, Table 7.2 shows the absorbance score for the modeled computation and its alignment with the expected output from the equivalent digital computation. The table shows three entries only for the two-input function (which should have four input patterns) because the missing fourth entry is symmetric to the second entry.

In 1	In 2	Input pH Encoding	Digital Output	Abs.	Abs. Score	Normalized
0	0	(14.27, 14.27)	**1**	0.435	1.177	**1**
1	0	(14.27, 0.62)	**1**	0.406	1.030	**1**
1	1	(0.62, 0.62)	**0**	0.230	0.148	**0**

Table 7.2: Quantitative analysis of NAND gate operation using enzymatic reactions. The "Digital Output" column indicates the expected binary output based on the input combinations "In 1" and "In 2," while "Input pH Encoding" represents the pH values used to encode the input signals. The "Abs." column lists the measured absorbance at 340 nm, which corresponds to NADH presence. The "Abs. Score" is calculated from the absorbance using the provided equation, and the "Normalized" column presents the final binary output after applying the thresholding value.



(a) Digital circuit for AND gate followed by an OR gate

(b) Enzymatic circuit for AND gate followed by an OR gate

Figure 7.4: (a) Conventional digital circuit comprised of cascading AND gate with OR gate. (b) Representing the same circuit using the enzymatic reactions model.

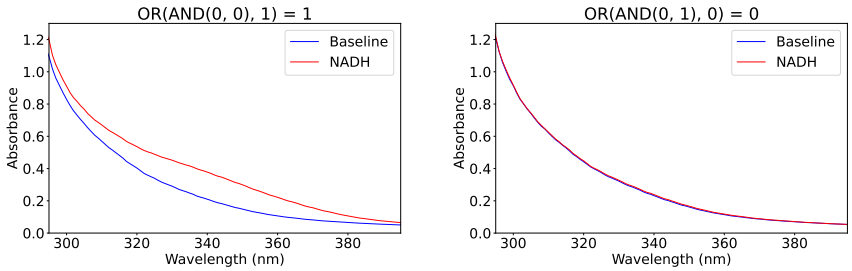


Figure 7.5: UV-vis spectroscopy for the outputs of the AND-OR circuit modeled using enzymatic reactions.

Modeling NOR Gate

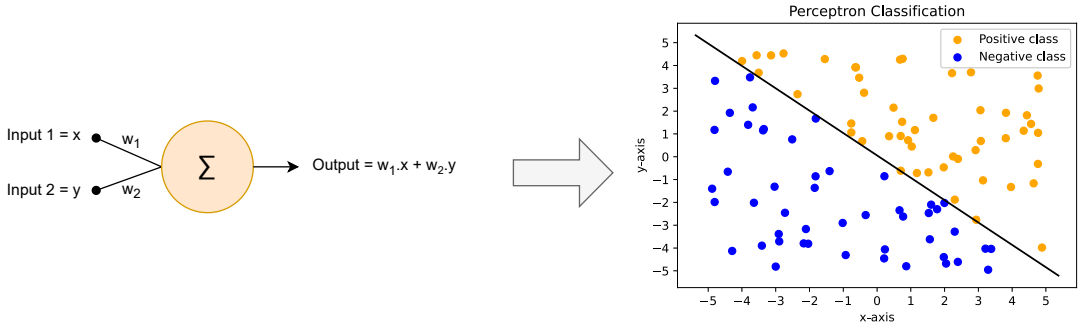
The NOR gate, another essential two-input logic gate, outputs a "1" only when both inputs are "0". In our enzymatic model, this principle is applied by configuring the input signals to prevent the enzymatic reaction from proceeding unless both inputs are "0". For the chemical encoding of binary signals, we use a pH of 12.87 for encoding a signal of value "0" and a pH of 0.3 for encoding a signal of value 1. This setup is designed to ensure that the enzymatic reaction is inhibited in the presence of any "1" input, effectively simulating the NOR gate's logical operation. Table 7.3 presents the absorbance scores for outputs observed from our enzymatic NOR gate when subjected to different input patterns. The corresponding UV-vis absorbance curves are shown in the supplementary material.

In 1	In 2	Input pH Encoding	Digital Output	Abs.	Abs. Score	Normalized
0	0	(12.87, 12.87)	**1**	0.411	1.057	**1**
1	0	(12.87, 0.30)	**0**	0.223	0.116	**0**
1	1	(0.30, 0.30)	**0**	0.233	0.163	**0**

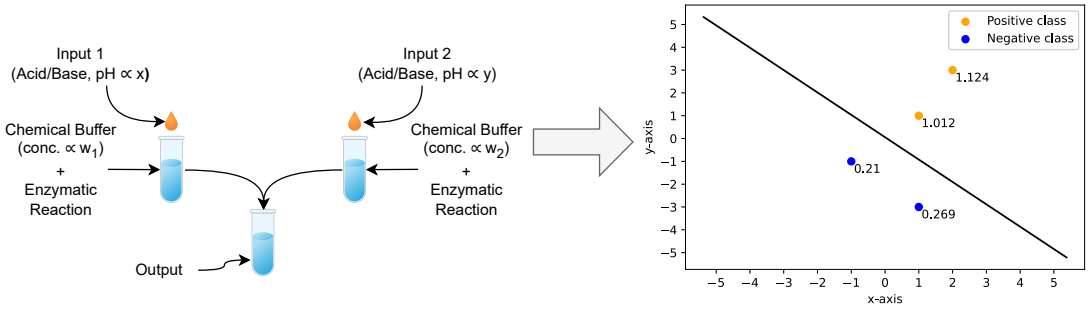
Table 7.3: Quantitative analysis of NOR gate operation using enzymatic reactions. The "Digital Output" column indicates the expected binary output based on the input combinations "In 1" and "In 2," while "Input pH Encoding" represents the pH values used to encode the input signals. The "Abs." column lists the measured absorbance at 340 nm corresponding to NADH presence. The "Abs. Score" is calculated from the absorbance using the provided equation, and the "Normalized" column presents the final binary output after applying the thresholding value.

Modeling OR Gate

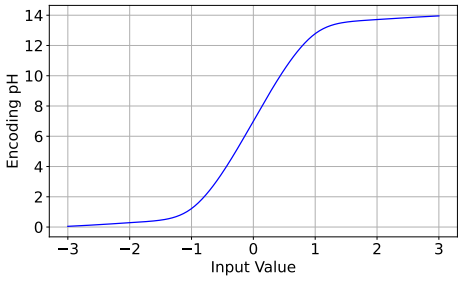
The OR gate, fundamentally the inverse of the NOR gate, outputs a "1" if at least one of its inputs is "1". To model this gate effectively within our enzymatic system, we incorporate a third buffer to model the additional computational stage. This addition necessitates a re-evaluation of the concentration levels for the signals "0" and "1", ensuring that the overall system accurately reflects the OR gate's logic. Hence, we use a pH of 0.6 for encoding a signal of value "0" and a pH of 14.39 for encoding a signal of value "1", respectively. By adjusting these concentrations, we align the enzymatic reactions to mirror the expected behavior of an OR gate. Table 7.4 presents the absorbance scores for outputs observed from our enzymatic OR gate when subjected



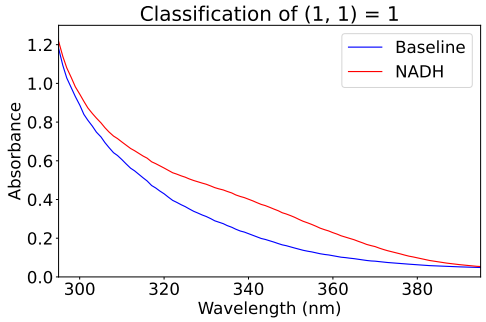
(a) Conventional Machine Learning Perceptron classifying points into positive and negative classes.



(b) Enzymatic Machine Learning Perceptron classifying points into positive and negative classes.



(c) Mapping of input coordinates into pH values.



(d) Absorbance curves for classifying the point (1,1) using enzymatic reactions.

Figure 7.6: Enzymatic machine learning perceptron. (a) A conventional machine learning perceptron classifies points into positive and negative classes. (b) An equivalent enzymatic machine learning perceptron for point classification. The chemical buffers encode the weights, while the acids/bases encode the inputs. (c) Mapping of the input coordinates into pH values to be applied to the perceptron. (d) Sample absorbance curves for classifying the point (1,1) using the enzymatic machine learning perceptron.

to different input patterns. The corresponding UV-vis absorbance curves are shown in the supplementary material.

In 1	In 2	Input pH Encoding	Digital Output	Abs.	Abs. Score	Normalized
0	0	(0.60, 0.60)	**0**	0.235	0.175	**0**
1	0	(14.39 0.60)	**1**	0.404	1.018	**1**
1	1	(14.39, 14.39)	**1**	0.419	1.094	**1**

Table 7.4: Quantitative analysis of OR gate operation using enzymatic reactions. The "Digital Output" column indicates the expected binary output based on the input combinations "In 1" and "In 2," while "Input pH Encoding" represents the pH values used to encode the input signals. The "Abs." column lists the measured absorbance at 340 nm corresponding to NADH presence. The "Abs. Score" is calculated from the absorbance using the provided equation, and the "Normalized" column presents the final binary output after applying the thresholding value.

Cascading AND-OR Gates

For the purpose of demonstrating the cascading capability of 2-input gates within our enzymatic system, we model a 3-input circuit configuration as depicted in Fig. 7.4. This arrangement consists of an AND gate followed by an OR gate. The AND gate is realized through two buffers, whose output is subsequently integrated with another input via a third buffer to emulate the OR gate functionality. We used a pH of 0.3 for encoding a signal of value "0" and a pH of 14.6 for encoding a signal of value "1". Fig. C.5 illustrates the resulting outputs from our enzymatic cascaded gate system for two sample inputs, additional absorbance curves are provided in the supplementary material, while Table 7.5 shows Absorbance Scores for different input points.

A	B	C	Digital Output	Abs.	Abs. Score	Normalized
0	0	0	**0**	0.224	0.118	**0**
0	1	0	**0**	0.239	0.195	**0**
1	0	0	**1**	0.391	0.956	**1**
0	1	1	**1**	0.396	0.982	**1**
1	0	1	**1**	0.411	1.055	**1**
1	1	1	**1**	0.420	1.101	**1**

Table 7.5: Quantitative analysis of the AND-OR circuit gate modeled using enzymatic reactions. The "Digital Output" column indicates the expected binary output based on the input combinations "A," "B," and "C," where we use a pH of 14.6 to encode a signal of value "1" and a pH of 0.3 to encode a signal of value "0". The "Abs." column lists the measured absorbance at 340 nm corresponding to NADH presence. The "Abs. Score" is calculated from the absorbance using the provided equation, and the "Normalized" column presents the final binary output after applying the thresholding value.

Enzymatic Machine Learning Model

Building upon the foundational principles outlined in preceding sections, we extend the application of enzymatic reactions to the realm of machine learning, specifically for the task of binary classification using a perceptron model. A perceptron is a classification model that can decide whether or not an input vector is a member of a specific class, as shown in Fig 7.6a. Hence, we propose a model analogous to a neural network

perceptron, as illustrated in Fig. 7.6b. In this model, what would traditionally be weights of a perceptron are instead represented by the concentration levels of chemical buffers. Input signals are modeled through variations in the concentrations of acids and bases within these buffers. The primary objective of this model is to classify a set of data points (x,y) into either positive or negative categories. For example, if a coordinate point (1, 1) is represented by the acid/base concentrations (x, y), then the point (2, 3) would be encoded with concentrations (2x, 3y) to reflect the proportional relationship between the points. Additionally, if we use concentrations of bases to represent positive coordinates, we choose acids with the same concentrations to represent their negative counterparts. Fig. 7.6c shows the mapping of different values into their corresponding pH encoding. This transformation allows for a straightforward mapping of input data into the biochemical domain, enabling the enzymatic network to process the information. Similar to how we compute the concentration for encoding the input signal, we adopt the same logic to calculate the correct buffer concentrations that mimic the neural network's weights, where we want to promote the forward reaction for positive samples and inhibit it for negative samples. Once these concentrations are computed, the model can classify data points based on the presence or absence of L-malic acid, respectively, wherein the presence of L-malic acid signifies a positive classification outcome, whereas its absence denotes a negative classification. To model our perceptron, two buffers were designated to handle the x and y coordinates, with their pH levels adjusted to 6.6 and 7.0 to model the network's weights. We validated 100 sample points with $in-silico$ simulation of the reaction. In addition, we evaluated four distinct points chemically, encompassing two positive samples and two negative samples. The positive samples were represented by coordinates $(1, 1)$ and encoded with pH values of $(13.0, 13.0)$ for the first point and $(2, 3)$ encoded as $(13.3, 13.48)$ for the second point, respectively. Conversely, the negative samples were represented by coordinates $(-1, -1)$ and encoded with pH values of $(1.0, 1.0)$ for the first point and $(1, -3)$ encoded with pH values of $(13.0, 0.52)$ for the second point. Fig. 7.6d provides a visual representation of the classification outcome of a sample point, while Table 7.6 shows the absorbance scores for the four points, showcasing the capability of our enzymatic model to distinguish between positive and negative classes.

Input	**Input pH Encoding**	**Digital Label**	**Abs.**	**Abs. Score**	**Normalized Label**
(1,1)	(13.00, 13.00)	**1**	0.402	1.012	**1**
(-1, -1)	(1.00, 1.00)	**0**	0.242	0.210	**0**
(2,3)	(13.30, 13.48)	**1**	0.425	1.124	**1**
(1, -3)	(13.00, 0.52)	**0**	0.254	0.269	**0**

Table 7.6: Absorbance score for classification results of 4 different coordinate points using enzymatic machine learning classifier. The "Input" column presents the input point coordinates, and the "Abs." column lists the measured absorbance at 340 nm, which corresponds to NADH presence. The "Abs. Score" is calculated from the absorbance using the provided equation, and the "Normalized" column presents the final label after applying the thresholding value.

7.4 Conclusion

In this chapter, we introduced our approach to digital computation that utilizes enzymatic reactions to emulate the functionalities of digital logic gates and machine learning neurons. Our method leverages the pH sensitivity of the enzymatic reaction to encode binary signals through the concentrations of strong acids or bases. We explained the problem formulation to tune the concentrations of acids and bases to encode the input signals properly. Our empirical evaluations demonstrated the capability of our framework through the construction of digital gates and machine learning perceptron in the laboratory setting, illustrating the practical application of enzymatic reactions in computational tasks. The potential of enzymatic reactions in computing opens avenues for future research, including refining and exploring the integration of biological and electronic components to enhance computational performance or improve biomedical devices.

7.5 Methods

All solutions for the experiment were prepared using deionized water as a solvent (Millipore Milli-Q). For the signal encoding, we used 20 μL (per input) of Hydrochloric Acid (HCl, 36.5% to 38.0%, Fisher Chemical) and Sodium Hydroxide (NaOH, $\geq$ 97%, Fisher Chemical). The enzyme utilized was Fumarase, obtained from a porcine heart (Sigma Aldrich, Natick, MA) dissolved in 0.1% Bovine Serum Albumin solution (Sigma Aldrich, Natick, MA). The chemical buffers were based on a Phosphate buffer solution (pH 7.00, Fluka). pH Meter Kit (PH700, Apera) was used to verify the pH across different reactions. Fumaric Acid (>99.0%, Sigma Aldrich, Natick, MA) was used to prepare the reagent solutions. To detect the enzymatic reaction outcomes, we employed NAD+ and a glutamate-oxaloacetate suspension (Megazyme, Lansing, MI). The final measurements were conducted using UV spectroscopy (Varian Cary 50 Bio UV-Vis Spectrophotometer) to assess the success of the enzymatic processes.

Chapter 8

Summary and Possible Extensions

8.1 Summary of Contributions

This dissertation investigated approaches to enhance the efficiency and adaptability of machine learning models, focusing on parameter-efficient architectures through multi-task learning optimization and unconventional computing platforms.

In Chapter 3, we introduced MTLoRA, a framework for parameter-efficient training of multi-task learning (MTL) models. This chapter presented the our approach to address parameter-efficient training specifically for MTL models, enabling efficient adaptation of a single shared backbone to multiple tasks. The framework incorporated Task-Agnostic (TA-LoRA) and Task-Specific (TS-LoRA) low-rank adaptation modules, designed to disentangle the parameter space in MTL fine-tuning. We applied MTLoRA to hierarchical-transformer-based MTL architectures for multiple downstream dense prediction tasks, achieving higher accuracy compared to fully fine-tuning the entire MTL model while reducing the number of trainable parameters by 3.6×. Furthermore, the chapter showed that MTLoRA established a Pareto-optimal trade-off between the number of trainable parameters and downstream task accuracy, outperforming existing parameter-efficient training methods. The effectiveness of MTLoRA was demonstrated across various backbone architectures, decoders, and pre-training datasets, indicating its flexibility and potential for wide application in parameter-efficient multi-task learning scenarios.

In Chapter 5, we introduced AdaMTL, an adaptive framework for input-dependent dynamic sparsification in multi-task learning models. AdaMTL addresses the challenge of optimizing computational efficiency in multi-task learning scenarios by dynamically adjusting model complexity based on input characteristics. The framework incorporates a lightweight, task-aware policy network that learns to recognize and skip unnecessary computations in the multi-task model. We designed AdaMTL with a multi-grained decision-making process, consisting of both block-level and token-level policy networks. This approach allows for fine-grained control over model activation, optimizing both accuracy and efficiency across multiple tasks. The policy network generates execution strategies at runtime, deciding which parts of the model to activate based on the input complexity and target computational budget. To effectively train this adaptive system, we developed a

multi-stage training recipe for co-training the policy network alongside the multi-task learning model. This includes a task-aware alternating training strategy to effectively balance the needs of multiple tasks without suffering from catastrophic forgetting. We evaluated AdaMTL on the PASCAL dataset, demonstrating its ability to reduce computational complexity while maintaining or improving accuracy compared to static multi-task models and single-task baselines. Our experiments showed that AdaMTL could reduce computational complexity by 43% while improving accuracy by 1.32% compared to single-task models. When combined with state-of-the-art multi-task learning techniques, AdaMTL boosted accuracy by 7.8% while improving efficiency by 3.1×. The work presented in this chapter extends our research on parameter-efficient multi-task learning, offering a dynamic approach that adapts to input complexity and computational constraints.

In Chapter 4, we introduced CRLoMTL, a novel approach designed to address the challenges of multi-task learning (MTL), particularly the issues of negative transfer and conflicting gradients. CRLoMTL leverages low-rank matrix adaptations to operate directly on the model's parameter space, offering a new approach to balancing task-specific needs with global model optimization. The framework incorporates a multi-pass training strategy that alternates between task-specific passes and global passes. During task-specific passes, each task updates its own set of low-rank matrices, while in global passes, the model learns an optimal combination of these task-specific adaptations. CRLoMTL employs trainable coefficients and a softmax function to compute a weighted average of task matrices, effectively integrating them into a single representative matrix that encapsulates task interactions while minimizing conflicts. Additionally, to prevent overfitting to any single task and promote generalization, CRLoMTL implements a feature regularization technique using L2 loss between task-specific and unified representations. This approach ensures a balanced learning process across all tasks. We evaluated CRLoMTL on the PASCAL-Context dataset using Swin Transformer backbones. The results demonstrated significant performance improvements, with CRLoMTL achieving a 2.82% improvement over single-task baselines while training less than 4% of the parameters when using a Swin-Tiny backbone. Moreover, CRLoMTL outperformed state-of-the-art MTL methods in both accuracy and parameter efficiency. Importantly, unlike some existing methods, CRLoMTL does not incur additional parameters or computational overhead in the final trained model, making it particularly suitable for resource-constrained applications or scenarios requiring rapid task adaptation.

In Chapter 6, we explored parameter-efficiency through unconventional computing by implementing neural networks and digital circuits using acid-base chemistry was introduced. We introduced a dual-rail encoding scheme, where acids and bases represented binary values, allowing us to perform essential logic functions such as majority and negation. These functions formed the basis for more complex computations, including the construction of digital circuits and neural network classifiers. Using a robotic fluid-handling device, we orchestrated acid-base reactions to encode data and execute computations. We applied our approach to build neural network classifiers for image recognition tasks, specifically using the MNIST dataset. The results showed that the acid-base neural networks closely matched their digital counterparts, with over 99% accuracy. This demonstrated that chemical reactions could be used to achieve parameter-efficient computation by leveraging the continuous nature of acid-base concentrations to represent data in a compact and efficient manner. This work contributed to the exploration of alternative, parameter-efficient computational platforms

by showcasing the potential of chemical systems to perform complex tasks with analog signals.

In Chapter 7, we extended the parameter-efficient computing framework introduced in Chapter 5 by incorporating enzymatic reactions to implement digital circuits and neural networks. Inpsired by the acid-base chemical systems previously explored, this chapter utilized pH-controlled enzymatic reactions as a novel approach to parameter-efficient computation. The enzymatic reactions, specifically using the enzyme fumarase, enabled binary signal encoding based on pH levels, which were controlled by varying concentrations of acids and bases. This allowed for the design and execution of digital logic gates and neural network classifiers in a chemically buffered environment. We demonstrated the construction of digital circuits, including NAND, NOR, and OR gates, using enzymatic reactions. The cascaded gates further showcased the potential for constructing more complex computational architectures. Additionally, we extended this model to implement a machine-learning classifier, where the weights of a neural network were encoded as chemical concentrations, and the system was evaluated using binary classification tasks. The results indicated that enzymatic reactions could provide a biologically-inspired, parameter-efficient computing framework with potential applications in unconventional computing. This chapter highlights the viability of enzymatic reactions as a continuation of our work on alternative computing methods and their potential for parameter-efficient computations.

8.2 Potential Extensions

The research presented in this dissertation lays the groundwork for several promising avenues of future investigation in the field of parameter-efficient ML paradigms. These potential extensions build upon the foundations established in the preceding chapters, aiming to further enhance the efficiency, adaptability, and applicability of the proposed methodologies. The following subsections outline four key directions for future research, each addressing critical aspects of model optimization:

8.2.1 Task-Adaptive Pruning in Low-Rank Multi-Task Learning

One potential extension involves the integration of task-adaptive pruning techniques with the aims to further enhance parameter efficiency by dynamically adjusting the sparsity of low-rank adaptation matrices based on task-specific requirements and model performance. The proposed method would involve developing a mechanism to assess the importance of different elements within the low-rank matrices for each task. This importance metric could be based on factors such as gradient magnitude, feature activation patterns, or task-specific performance metrics. Using this information, the system could adaptively prune less important elements, effectively creating task-specific sparse low-rank adaptations. Research in this direction could explore various pruning strategies, such as magnitude-based pruning, gradient-based pruning, or more sophisticated methods that consider the inter-task relationships. The pruning process could be implemented as a continuous operation during training, allowing the model to adapt its sparsity patterns as it learns. Furthermore, this approach could investigate the potential for knowledge transfer between tasks through the pruning patterns. For instance, the system might learn to retain certain elements across multiple tasks, indicating their general importance, while sparsifying task-specific elements. The development of efficient algorithms for updating and

maintaining these sparse low-rank structures would be crucial. This might involve exploring sparse matrix multiplication techniques or developing custom hardware acceleration for sparse low-rank operations.

8.2.2 Federated Learning for Parameter-Efficient Multi-Task Models

A promising direction for future research involves the integration of federated learning techniques with parameter-efficient multi-task learning frameworks. This approach aims to address the growing concerns surrounding data privacy and security while maintaining the benefits of multi-task learning and parameter efficiency. The proposed research would explore methods to adapt low-rank adaptation techniques and other parameter-efficient strategies to function effectively within a federated learning paradigm. This would involve developing algorithms that can aggregate model updates from distributed clients without compromising the parameter efficiency or task-specific adaptations of the central model. One potential area of investigation could be the design of communication-efficient protocols for transmitting low-rank updates in federated settings. This might involve developing compression techniques specifically tailored for low-rank matrices or exploring methods to selectively communicate only the most impactful updates. Another aspect of this research could focus on maintaining task performance and parameter efficiency in the face of non-IID data, a common challenge in federated learning scenarios. This might involve developing robust aggregation methods that can effectively combine task-specific adaptations from heterogeneous data distributions. Furthermore, this line of research could explore the potential for personalization in federated multi-task learning. This might involve developing techniques to fine-tune the shared parameter-efficient model for individual clients or groups of clients with similar task distributions, while still maintaining overall model coherence and efficiency. This could involve adapting differential privacy mechanisms or secure aggregation protocols to work effectively with low-rank adaptation matrices and other parameter-efficient structures.

8.2.3 Multi-Modal Integration in Parameter-Efficient Multi-Task Learning

Extending the current work on parameter-efficient multi-task learning to multi-modal contexts presents an avenue for future research. This direction aims to develop frameworks that can efficiently handle tasks spanning multiple modalities (e.g., vision, language, audio) while maintaining high parameter efficiency. The research could focus on designing novel architectures that allow for effective sharing of information across modalities without compromising the specific processing requirements of each modality. This might involve developing modality-specific low-rank adaptations that can interact and share information through a central, modality-agnostic representation. One potential approach could be to create a hierarchical structure where low-level, modality-specific features are processed independently, while higher-level features are integrated through shared low-rank adaptations. This structure could allow for efficient processing of modality-specific information while enabling cross-modal learning and generalization. Another area of investigation could be the development of attention mechanisms that can selectively focus on relevant information across modalities for each task. This could involve creating parameter-efficient cross-modal attention layers that can dynamically weight the importance of different modalities based on the current task and input.

8.2.4 Hybrid Digital-Chemical Systems for Parameter-Efficient AI

A direction for future research is the development of hybrid systems that combine traditional digital computing with chemical computing paradigms to achieve highly parameter-efficient AI models. This approach aims to leverage the strengths of both computing modalities: the precision and programmability of digital systems and the inherent parallelism and energy efficiency of chemical reactions. The research could focus on designing interfaces that allow information flow between digital and chemical components. This might involve developing novel analog-to-digital and digital-to-analog conversion techniques specifically tailored for chemical signals, enabling efficient translation between the two computing paradigms. One potential approach could be to offload certain computationally intensive or highly parallel operations to the chemical domain, while relying on digital components for operations that require high precision or complex control flow. For instance, the analog nature of chemical concentrations could be leveraged to perform efficient matrix multiplications or convolutions, which are common operations in neural networks. Investigation into hybrid architectures could also explore how to best partition neural network layers between digital and chemical components. This might involve developing optimization algorithms that can determine the most efficient distribution of computations across the two domains based on the specific requirements of each layer and the overall network architecture.

Appendix A

MTLoRA's Generalizability

A.1 Different Datasets: ImageNet-1K vs ImageNet-22K

We analyze the effect of using different pretraining datasets on the performance of MTLoRA. Figure A.1 shows the relative improvement in accuracy compared to the single task models of MTLoRA when applied to Swin-Tiny and Swin-Base pretrained on ImageNet-$1K$ and ImageNet-$22K$. The figure shows that the pretraining dataset can have a significant impact on the model's performance. Specifically, using a model pre-trained on a richer dataset (i.e., ImageNet-22K) results in a better performance on the downstream tasks without any additional cost on our parameter-efficient training methodology.

A.2 MTLoRA with different Adaptation Scales

The scale value α in Equation 2 determines how much the fine-tuned model can deviate from the original baseline model. For example, A scale value of 0 is the same as not using the LoRA weights and only using the base model weights, and a scale value of 1 means that both the LoRA weights as well as the base model weights have the same influence. It is common to use α in $1, 2$ for language models [69]; however, we experiment with different scales to analyze their effect when fine-tuning vision transformers for multiple tasks. Figure A.2 shows the overall accuracy at different scales. We found that empirically, scale $\mathbf{4}$ performs the best for the purpose of MTLoRA.

A.3 Different Backbones

To ensure the generalizability of MTLoRA, we apply it to another vision transformer backbone - Pyramid Vision Transformer [154]. We analyze the impact of using MTLoRA when applied on *PVT-Small*. Table A.1 shows that applying MTLoRA offers Pareto-optimal accuracy-efficiency trade-off compared to state-of-the-art paramter-efficient training techniques. Specifically, MTLoRA achieves higher accuracy (Δm) when compared to Hyperformer [101] while training $2\times$ fewer parameters.

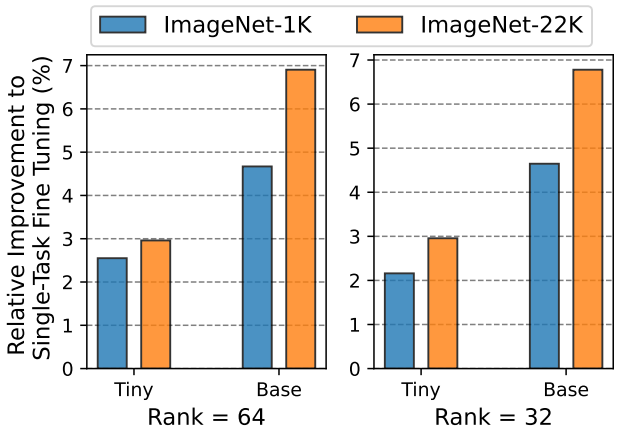


Figure A.1: Performance with MTLoRA with different pretraining datasets on various Swin-Transformer backbones.

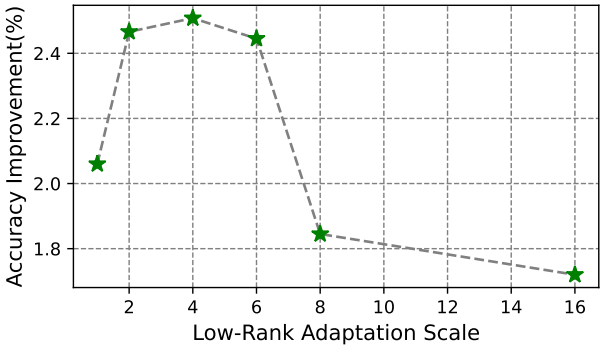


Figure A.2: Effect of the hyper-parameter α on the accuracy of MTLoRA on the downstream tasks.

A.4 Different Decoders in MTLoRA

We analyze the effect of using different decoders on the performance of MTLoRA. We choose 3 commonly used decoders for dense prediction tasks: (1) HRNet [152], which interpolates and concatenates multi-scale features from the hierarchical backbone and then passes them to a couple of MLP layers. (2) SegFormer [160], which uses MLP layers to combine the multi-scale features from the hierarchical backbone, then parse them to one last MLP layer for prediction. (3) Atrous Spatial Pyramid Pooling (ASPP) [27], which has a more complicated architecture utilizing spatial pyramid pooling capable of extracting multi-scale contextual information by probing the incoming features with filters or pooling operations at multiple rates and multiple effective fields-of-view. We plug those decoders into a pretrained Swin-Tiny backbone, then use our MTLoRA technique to train the model to perform multiple downstream tasks. Table A.2 shows that MTLoRA generalizes perfectly when different decoders are used. Different decoders can provide different accuracy-efficiency trade-offs, which provide flexibility to adapt the training budget depending on the application requirements and the available resources.

Table A.1: *MTLoRA* versus SoTA parameter-efficient training methods when applied on Pyramid Vision Transformer [154] backbone. The table summarizes the number of trainable parameters in each method. It also includes the accuracy of the downstream tasks as well as the average MTL model's accuracy (Δm).

Method	**SemSeg** ($mIoU \uparrow$)	**Human Parts** ($mIoU \uparrow$)	**Saliency** ($mIoU \uparrow$)	**Normals** ($rmse \downarrow$)	$\Delta m(\%)$	**Trainable Parameters** (M)
Single Task	68.81	61.27	62.67	17.55	0.00	97.51
MTL - Fine Tune Decoders	64.86	51.18	61.54	19.55	-8.85	2.11
Compactor++ [81]	70.29	54.80	63.16	18.82	-3.71	2.20
BitFit [168]	71.41	55.71	64.08	18.69	-2.38	2.34
LoRA [69]	71.89	56.9	64.27	18.48	-1.35	2.41
Adapter [60]	71.94	56.38	64.16	18.75	-1.97	2.90
Polyhistor [99]	71.00	57.52	65.83	17.83	+0.13	7.32
Hyperformer [101]	70.81	57.76	65.49	17.75	+0.14	16.14
MTLoRA ($r = 64$)	69.74	58.08	65.62	17.35	**+0.50**	8.69

Table A.2: Evaluating *MTLoRA* with $rank = 32$ when applied on Swin-Tiny backbone pretrained on ImageNet-22K dataset and various decoders for the downstream dense prediction tasks. The table summarizes the number of parameters for each decoder as well as the total number of parameters. It also includes the accuracy of the downstream tasks as well as the average MTL model's accuracy (Δm).

Decoder	**SemSeg** ($mIoU \uparrow$)	**Human Parts** ($mIoU \uparrow$)	**Saliency** ($mIoU \uparrow$)	**Normals** ($rmse \downarrow$)	$\Delta m(\%)$	**Trainable Parameters** (M) $Decoder/All$
HR-Net [152]	69.44	61.08	63.24	16.47	+2.93	1.94 / 6.08
SegFormer [160]	69.59	61.13	63.74	16.62	+3.00	2.08 / 6.22
ASPP [27]	72.32	60.98	63.04	16.51	+3.83	12.44 / 16.58

Appendix B

Additional Examples for Acid-Base Networks

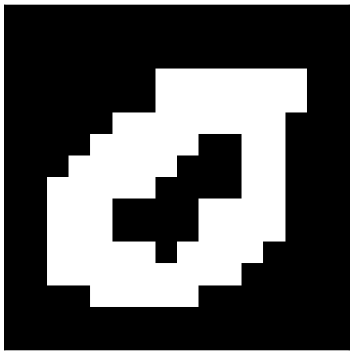


Figure B.1: An 8×8 example image that results in the wrong classification using acid-base computations. The *in-silico* model generates scores of {4.0, 2.0} (before the softmax); consequently, the acid-base model generates pH values of {(2.2, 11.8), (2.51, 11.49)}, which is correct but indistinguishable by color only.

Table B.1: Pixel values for the 8×8 example image that results in the wrong classification using acid-base computations.

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0
2	0	0	0	1	1	0	0	0
3	0	0	0	1	0	1	0	0
4	0	0	0	1	0	1	0	0
5	0	0	0	1	0	1	0	0
6	0	0	0	1	1	1	0	0
7	0	0	0	0	0	0	0	0

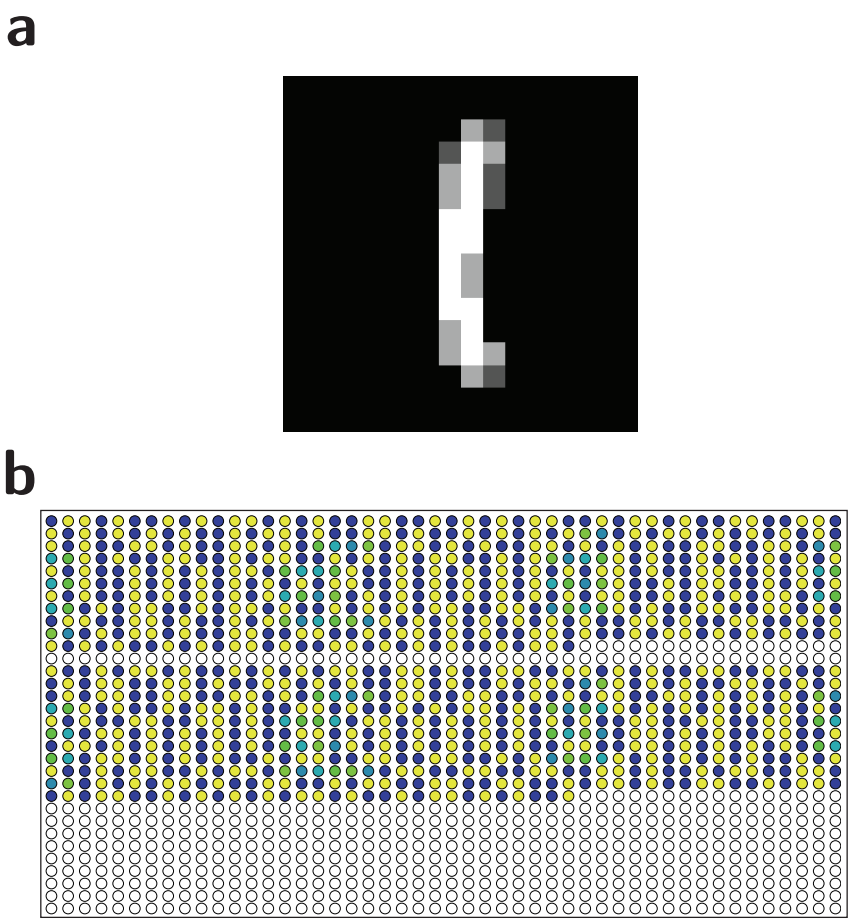


Figure B.2: Encoding of grayscale image using acid-base dual-rail encoding. **a** Grayscale encoded image to be classified. **b** Encoded image using acids (yellow) and bases (blue). The color intensity and shade of each well correspond to the level of the dilution based on the encoded value following to the pH scale colors shown in Figure 1b.

Simulation of Digit Classifier on Microfluidic Board OpenDrop v4

In order to demonstrate that our approach can work on different platforms, we designed a simulation for a similar experiment on OpenDrop v4 [10], a digital microfludics platform that uses electro-wetting technology and is more affordable than the acoustic liquid handler. The board has a 14x8 grid, which we used to encode a 7x7 image with the weights applied as previously described. The simulation then combines the droplets from the positive side of the dual-rail and moves the output to the lower-left cell of the grid, which represents the positive part of the dual-rail neuron's output. Similarly, the negative sides of the dual-rail are combined and moved to the lower-right corner, which represents the negative part of the dual-rail neuron's output. The final output is shown as the pH/color of the dual-pair of the droplet in the lower-left corner of the grid and the droplet in the lower-right corner, where acid-base would indicate +1, and base-acid would indicate -1/0. Supplementary Figure B.4 shows the first instance of the simulation with the encoded image and the last

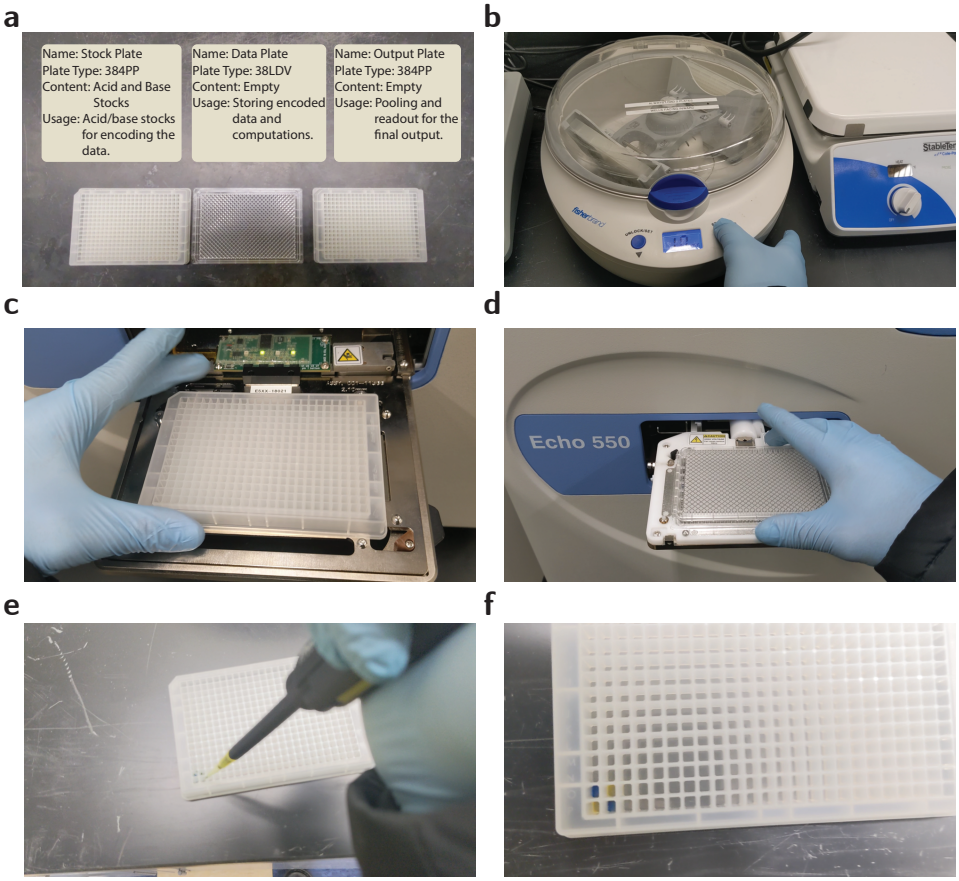


Figure B.3: Working Setup of the Classification of an 8×8 image. **a** The three plates used for running the experiment. **b** Centrifuging the stock plate. **c** Inserting the stock plate and the data plate into the liquid handler, after which we start the encoding protocol. **d** Retrieving the data plate, followed by centrifugation. Similarly, we insert the data plate and the output plate into the liquid handler, after which we start the pooling protocol. **e** Adding the pH indicator to the pooling plate, followed by centrifugation. **f** Final readout of the network's classification.

instance of the simulation after collecting the results of one neuron. We also ran a full digital logic gate on the physical board, but due to limitations on the volume of the liquid the board can handle, it was not feasible to run the full classification.

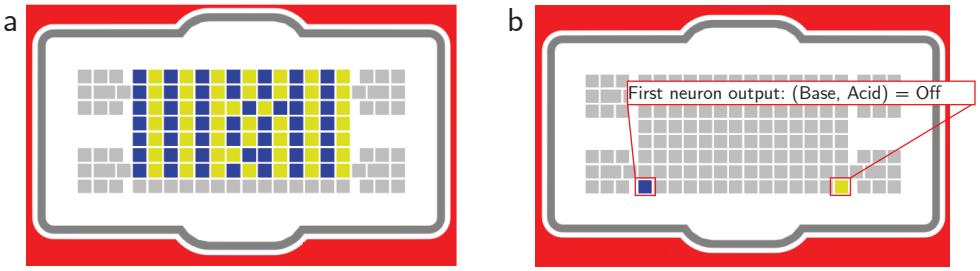


Figure B.4: Simulation of the Digit Classifier on a Microfluidic Device. **a** The encoding of a 7×7 image of the digit one using acids (yellow) and bases (blue) with network weights applied on a microfludic device grid. **b** The final output of the microfluidic simulation of one neuron.

Appendix C

Additional Enzymatic Circuit Designs

Preliminary: Gradient Descent Optimization

Gradient Descent has become a fundamental algorithm in the optimization of complex systems, notably within the context of machine learning. The algorithm operates by iteratively adjusting parameters in the direction of the steepest descent, or "gradient," of the function being optimized. Given its iterative nature, the algorithm has demonstrated utility in optimizing problems where the search space is high-dimensional and the objective function is non-convex [130][86]. In computational modeling, Gradient Descent has been applied to a range of applications, predominantly for neural network training [91]. However, gradient descent application is not confined to any single field, and it has made inroads into bioinformatics and systems biology for applications such as parameter estimation in complex biological models [110].

Enzyme Network for Computing Signal Concentrations

Computing the concentrations needed for the signals “0” or “1” is a main component of our system. We formulate an optimization problem modeled as a neural network, as depicted in Fig. C.1. The architecture of this network comprises two weights: one for the concentration corresponding to a binary “0” and the other for the concentration corresponding to a binary ’1.’

The input to the neural network consists of the truth table of the logic function under design. During the forward pass, the network computes the resultant pH levels for each combination of input signals based on their corresponding concentrations, as modulated by the reactions in the buffers.

Since we want the network to learn the concentration that would cause the buffer to be in the pH that makes the enzymatic reaction move forward or backward, we define a ranged loss function, represented by Equation C.1. y represents the network’s output, while a and b are the range in which we need the network’s output (i.e., the buffer’s pH) to lie, which is based on the pH range for the forward or reverse reaction to happen. This function is designed to incentivize the network to adjust its weights (signal concentration) such that the resultant pH falls within specified ranges according to the expected output (0 or 1). These ranges are dependent on the expected output from the logic function’s truth table. Specifically, the loss function pushes

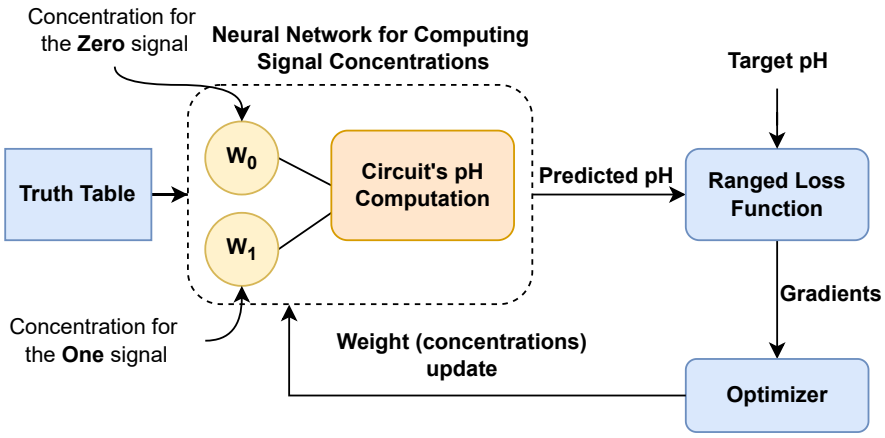


Figure C.1: Computing the concentrations for encoding the input signals using gradient descent. The concentrations are encoded as network weights. The pH computations capture the logic of the circuit's design and evaluate the pH at the different buffers. The ranged loss function updates the concentrations untill it meets the target range.

for a pH within the forward reaction range when the expected output is "1" and within the reverse reaction range when the expected output is '0.'

$$\text{Range Loss}(y,(a,b)) = \begin{cases} 0 & \text{if } a \le y \le b, \\ |y-a| & \text{if } y < a, \\ |y-b| & \text{if } y > b. \end{cases} \tag{C.1}$$

The neural network's training employs this specialized loss function, ensuring that the resultant pH for each logic condition aligns with the pre-specified ranges. Upon convergence, the network's optimized weights yield the concentrations that should be employed for signals "0" and "1" in the enzymatic logic gates.

Signal Readout

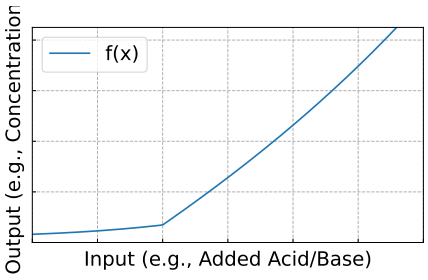
Detection of L-malic acid is usually done by applying a sequence of two catalyzed reactions to produce NADH.

$$\text{L}-\text{Malic acid} + \text{NAD}^+ \overset{\text{L}-\text{MDH}}{\rightleftharpoons} \text{oxaloacetate} + \text{NADH} + \text{H}^+ \tag{C.2}$$

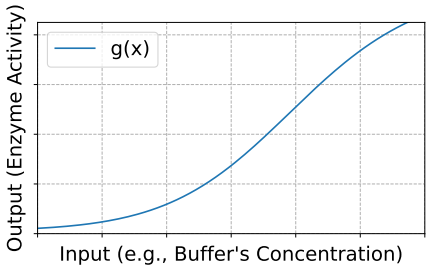
$$\text{Oxaloacetate} + \text{L}-\text{glutamate} \xrightarrow{\text{GOT}} \text{L}-\text{aspartate} + \text{2-oxoglutarate} \tag{C.3}$$

The first reaction, outlined in Equation C.2 is catalyzed by the enzyme L-malate dehydrogenase (L-MDH), whereby L-malic acid is oxidized to form oxaloacetate, with nicotinamide-adenine dinucleotide (NAD+) serving as the cofactor. The equilibrium of this reaction strongly favors the reactants L-malic acid and NAD+. To overcome this limitation and generate a detectable signal, a second reaction, outlined in Equation C.3 is induced to trap the resulting NADH product. This is accomplished by converting oxaloacetate to L-aspartate and 2-oxoglutarate, using an excess of L-glutamate as the substrate, in the presence of

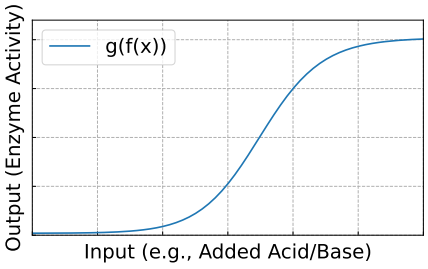
the enzyme glutamate-oxaloacetate transaminase (GOT). The amount of NADH generated corresponds to the original amount of L-malic acid present in the terminal buffer. The presence of NADH is subsequently quantified using its characteristic increase in absorbance at a wavelength of 340 nm. Measurement of this absorbance is executed with a UV spectrometer. An increase in absorbance at 340 nm indicates the presence of L-malic acid.



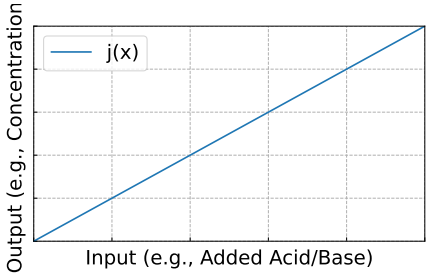
(a) Graph of the non-linear function $f(x)$, describing the behavior of chemical buffers in response to the addition of acid/base. The curve mimics a piecewise function corresponding to the behavior of the buffer within and outside the buffering range.



(b) Graph of the non-linear function $g(x)$, describing the enzyme activity at different concentrations, which increases as it approaches the optimal pH range.



(c) Graph of the non-linear composite function $(f \circ g)(x)$ describing the behavior of the enzyme while changing the buffer's concentration.



(d) Graph of the linear relation between input and output in linear computation systems, such as the change in the concentration of the system by varying the concentration of one of the substrates.

Figure C.2: Comparison between non-linear functions in our model (a) – (c), compared to the linear relation (d) induced by simpler computational models.

Visualizing Non-linear Components of the System

As we explained, our model introduces non-linearity through the chemical buffers that host the enzymatic reactions and the enzymatic reactions themselves.

If we model the buffer's change in pH in response to the concentration input of an acid or base through a non-linear function $f(x)$ and the enzyme's activity in response to the pH as a non-linear function $g(x)$. Thus, the system's overall response can be expressed as the composition of these two non-linear functions $(f \circ g)(x)$. This composition adds a layer of preferred non-linearity to our computational framework, allowing

for further flexibility in the framework. For instance, Fig. C.2a illustrates the non-linear characteristic of the function $f(x)$ where the curve captures the behavior of chemical buffers, represented by piecewise functions composed of linear segments to describe the buffer behavior inside and outside the buffer range. Fig. C.2b depicts the non-linearity within $g(x)$, which captures enzyme activity at different pH levels, showing how enzyme activity increases and peaks towards the optimal pH. Meanwhile, Fig. C.2c represents the composite non-linear function $(f \circ g)(x)$, which forms the foundational building block of our system, offering a more adaptable model in contrast to the conventional linearity, as indicated by Fig. C.2d, found in some current unconventional computation methods [9], where the system's output is changed by linearly varying the concentration of one of the substrates. Mathematically, our framework can be seen as an approximation model for a given logic design's functionality, where the function $(f \circ g)(x)$ serves as the building block. The logic functionality's structure dictates the architecture and how these non-linear functions are composed together, which translates into the enzymatic reactions we carry. Meanwhile, the determination of the input signal concentrations is akin to parameter optimization in a model, aiming to refine the system's behavior to match the intended computational function closely.

Absorbance Curves for the Digital Gates and the Machine Learning Perceptron

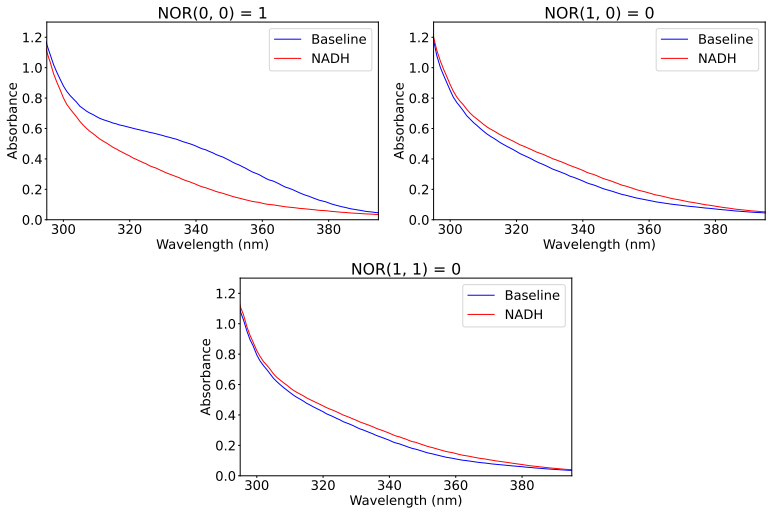


Figure C.3: UV-vis spectroscopy for the outputs of the NOR circuit modeled using enzymatic reactions.

We presented the quantitative output for our models of the digital gates and the machine learning perceptron. Fig. C.3, C.5, and C.6 show the corresponding UV-vis absorbance curves.

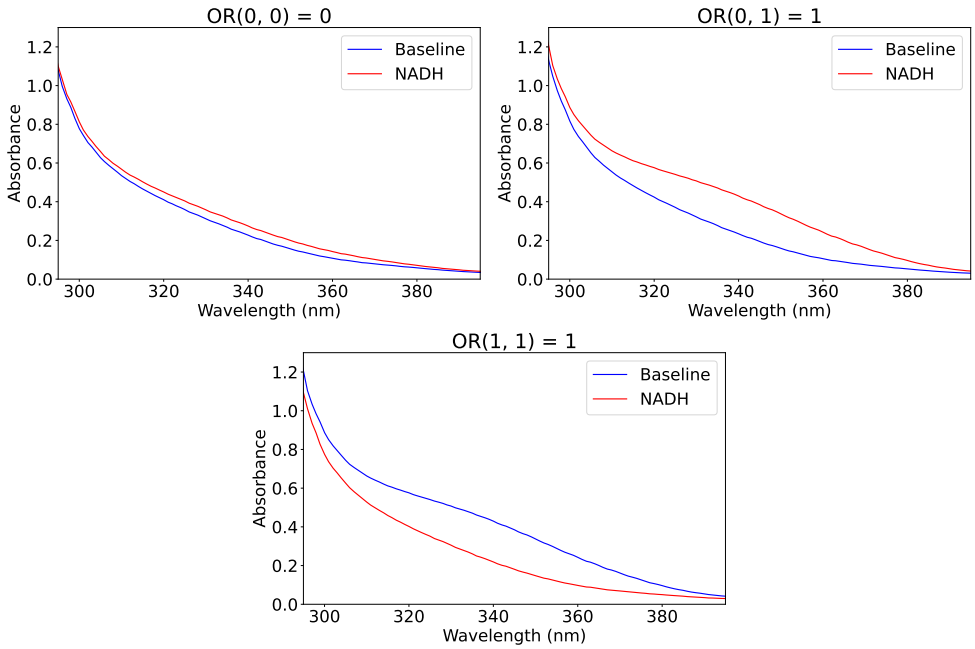


Figure C.4: UV-vis spectroscopy for the outputs of the OR circuit modeled using enzymatic reactions.

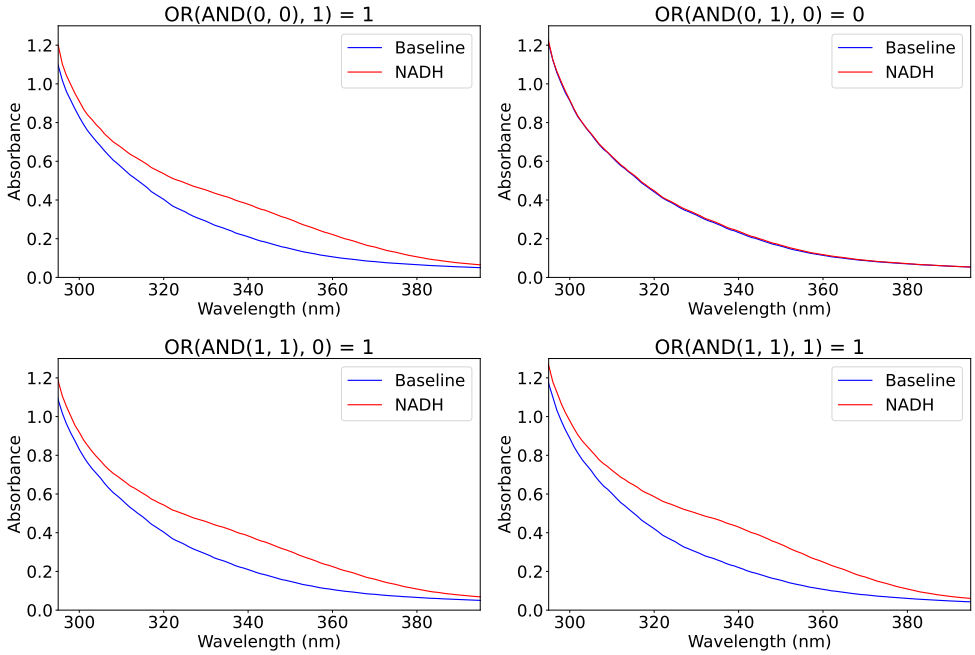


Figure C.5: UV-vis spectroscopy for the outputs of the AND-OR circuit modeled using enzymatic reactions.

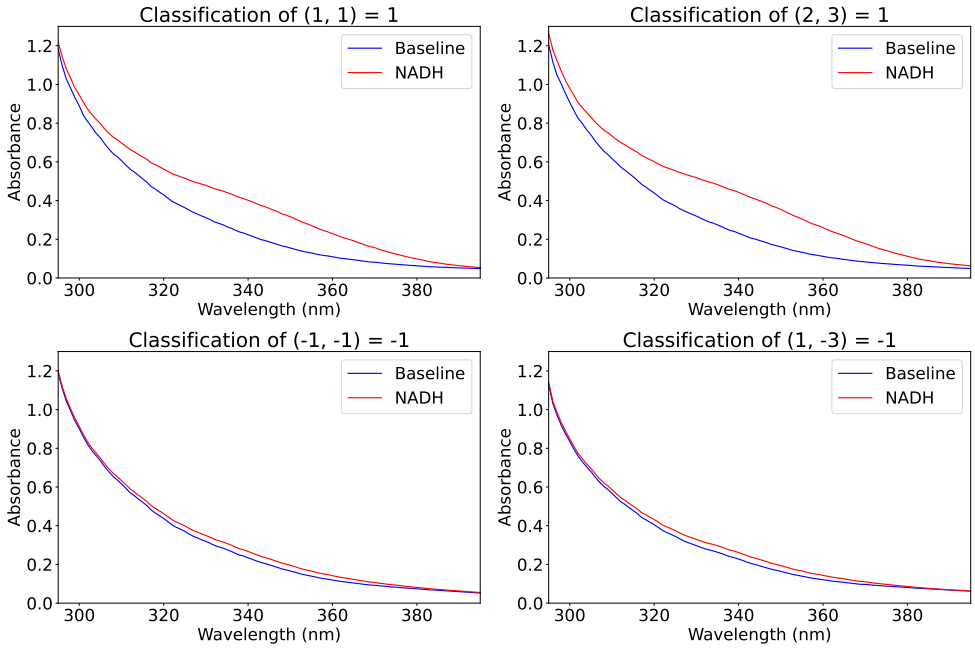


Figure C.6: Classification results for the machine learning perceptron modeled using enzymatic reactions.

Bibliography

[1] Andrew Adamatzky. *Advances in unconventional computing: Volume 1: Theory*, volume 22. Springer, 2016.

[2] Andrew Adamatzky, Larry Bull, and B De Lacy Costello. *Unconventional computing 2007*. Luniver Press, 2007.

[3] Andrew Adamatzky, Benjamin De Lacy Costello, and Tetsuya Asai. *Reaction-diffusion computers*. Elsevier, 2005.

[4] Leonard M Adleman. Molecular computation of solutions to combinatorial problems. *science*, 266(5187):1021–1024, 1994.

[5] Pratul K Agarwal. Enzymes: An integrated view of structure, dynamics and function. *Microbial cell factories*, 5:1–12, 2006.

[6] Ahmed Agiza. Discretized MNIST for Digital Circuits and Neural Networks Based on Acid-Base Chemistry Implemented by Dual-Rail Encoding and Robotic Fluid Handling. 12 2022.

[7] Ahmed Agiza. Source Code for Digital Circuits and Neural Networks Based on Acid-Base Chemistry Implemented by Dual-Rail Encoding and Robotic Fluid Handling. 12 2022.

[8] Ahmed Agiza, Marina Neseem, and Sherief Reda. Mtlora: Low-rank adaptation approach for efficient multi-task learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16196–16205, 2024.

[9] Ahmed A Agiza, Kady Oakley, Jacob K Rosenstein, Brenda M Rubenstein, Eunsuk Kim, Marc Riedel, and Sherief Reda. Digital circuits and neural networks based on acid-base chemistry implemented by robotic fluid handling. *Nature communications*, 14(1):496, 2023.

[10] Mirela Alistar and Urs Gaudenz. Opendrop: An integrated do-it-yourself platform for personal use of biochips. *Bioengineering*, 4(2):45, 2017.

[11] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2014.

[12] Alan Ansell, Edoardo Maria Ponti, Anna Korhonen, and Ivan Vulić. Composable sparse fine-tuning for cross-lingual transfer. *arXiv preprint arXiv:2110.07560*, 2021.

[13] Christopher E Arcadia, Amanda Dombroski, Kady Oakley, Shui Ling Chen, Hokchhay Tann, Christopher Rose, Eunsuk Kim, Sherief Reda, Brenda M Rubenstein, and Jacob K Rosenstein. Leveraging autocatalytic reactions for chemical domain image classification. *Chemical Science*, 12(15):5464–5472, 2021.

[14] Christopher E Arcadia, Eamonn Kennedy, Joseph Geiser, Amanda Dombroski, Kady Oakley, Shui-Ling Chen, Leonard Sprague, Mustafa Ozmen, Jason Sello, Peter M Weber, et al. Multicomponent molecular memory. *Nature communications*, 11(1):691, 2020.

[15] Christopher E Arcadia, Hokchhay Tann, Amanda Dombroski, Kady Ferguson, Shui Ling Chen, Eunsuk Kim, Christopher Rose, Brenda M Rubenstein, Sherief Reda, and Jacob K Rosenstein. Parallelized linear classification with volumetric chemical perceptrons. In *2018 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–9. IEEE, 2018.

[16] Frances H Arnold. Design by directed evolution. *Accounts of chemical research*, 31(3):125–131, 1998.

[17] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In *International conference on machine learning*, pages 254–263. PMLR, 2018.

[18] Giannis Bekoulis, Johannes Deleu, Thomas Demeester, and Chris Develder. Adversarial training for multi-context joint entity and relation extraction. *arXiv preprint arXiv:1808.06876*, 2018.

[19] Yaakov Benenson. Biomolecular computing systems: principles, progress and potential. *Nature Reviews Genetics*, 13(7):455–468, 2012.

[20] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[21] Robert J Beynon and John S Easterby. *Buffer solutions: The basics*. Taylor & Francis, 2004.

[22] Deblina Bhattacharjee, Tong Zhang, Sabine Süsstrunk, and Mathieu Salzmann. Mult: an end-to-end multitask learning transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12031–12041, 2022.

[23] Rohani binti abu Bakar and Junzo Watada. Dna computing and its applications: survey. *ICIC Express Letters ICIC International*, 2(1):101–108, 2008.

[24] Cris Calude and Gheorghe Paun. *Computing with cells and atoms: an introduction to quantum, DNA and membrane computing*. CRC Press, 2000.

[25] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *Computer Vision–ECCV 2020: 16th*

European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16, pages 213–229. Springer, 2020.

[26] Arnav Chavan, Zhuang Liu, Deepak Gupta, Eric Xing, and Zhiqiang Shen. One-for-all: Generalized lora for parameter-efficient fine-tuning. *arXiv preprint arXiv:2306.07967*, 2023.

[27] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018.

[28] Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. Longlora: Efficient fine-tuning of long-context large language models. *arXiv preprint arXiv:2309.12307*, 2023.

[29] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International conference on machine learning*, pages 794–803. PMLR, 2018.

[30] Kevin M Cherry and Lulu Qian. Scaling up molecular pattern recognition with dna-based winner-take-all neural networks. *Nature*, 559(7714):370–376, 2018.

[31] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, 2008.

[32] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[33] Michael Crawshaw. Multi-task learning with deep neural networks: A survey. *arXiv preprint arXiv:2009.09796*, 2020.

[34] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3150–3158, 2016.

[35] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee Micro*, 38(1):82–99, 2018.

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

[37] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[38] Jean-Antoine Désidéri. Multiple-gradient descent algorithm (mgda) for multiobjective optimization. *Comptes Rendus Mathematique*, 350(5-6):313–318, 2012.

[39] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2024.

[40] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[41] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[42] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[43] Mark Everingham, Luc Van Gool, Christopher Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88:303–338, 06 2010.

[44] Z Ezziane. Dna computing: applications and challenges. *Nanotechnology*, 17(2):R27, 2005.

[45] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[46] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[47] Perry A Frey and Adrian D Hegeman. *Enzymatic reaction mechanisms*. Oxford University Press, 2007.

[48] Carl Frieden and Robert A Alberty. The effect of ph on fumarase activity in acetate buffer. *J Biol Chem*, 212:859–868, 1955.

[49] Peng Gao, Jiaming Han, Renrui Zhang, Ziyi Lin, Shijie Geng, Aojun Zhou, Wei Zhang, Pan Lu, Conghui He, Xiangyu Yue, et al. Llama-adapter v2: Parameter-efficient visual instruction model. *arXiv preprint arXiv:2304.15010*, 2023.

[50] Ying Gao, Albert R Cross, and Robin L Armstrong. Magnetic resonance imaging of ruthenium-, cerium-, and ferroin-catalyzed belousov- zhabotinsky reactions. *The Journal of Physical Chemistry*, 100(24):10159–10164, 1996.

[51] Pier Luigi Gentili. The conformational contribution to molecular complexity and its implications for information processing in living beings and chemical artificial intelligence. *Biomimetics*, 9(2):121, 2024.

[52] Jerzy Gorecki, K Gizynski, J Guzowski, JN Gorecka, P Garstecki, G Gruenert, and P Dittrich. Chemical computing with reaction–diffusion processes. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 373(2046):20140219, 2015.

[53] Benjamin Graham, Alaaeldin El-Nouby, Hugo Touvron, Pierre Stock, Armand Joulin, Hervé Jégou, and Matthijs Douze. Levit: a vision transformer in convnet's clothing for faster inference. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 12259–12269, 2021.

[54] Lewis Grozinger, Martyn Amos, Thomas E Gorochowski, Pablo Carbonell, Diego A Oyarzún, Ruud Stoof, Harold Fellermann, Paolo Zuliani, Huseyin Tas, and Angel Goñi-Moreno. Pathways to cellular supremacy in biocomputing. *Nature communications*, 10(1):1–11, 2019.

[55] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, et al. Conformer: Convolution-augmented transformer for speech recognition. *arXiv preprint arXiv:2005.08100*, 2020.

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

[57] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

[58] David Harris and Sarah Harris. *Digital design and computer architecture*. Morgan Kaufmann, 2010.

[59] Hussein Hazimeh, Zhe Zhao, Aakanksha Chowdhery, Maheswaran Sathiamoorthy, Yihua Chen, Rahul Mazumder, Lichan Hong, and Ed Chi. Dselect-k: Differentiable selection in the mixture of experts with applications to multi-task learning. *Advances in Neural Information Processing Systems*, 34:29335–29347, 2021.

[60] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*, 2021.

[61] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[62] Xuehai He, Chunyuan Li, Pengchuan Zhang, Jianwei Yang, and Xin Eric Wang. Parameter-efficient model adaptation for vision transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 817–825, 2023.

[63] Geoffrey Hinton. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[64] Battery historian, August 2016. `https://github.com/google/battery-historian`.

[65] Allen Hjelmfelt, Edward D Weinberger, and John Ross. Chemical implementation of neural networks and turing machines. *Proceedings of the National Academy of Sciences*, 88(24):10983–10987, 1991.

[66] Kathy J Horadam. A generalised hadamard transform. In *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005.*, pages 1006–1008. IEEE, 2005.

[67] T Hospedales, A Antoniou, P Micaelli, and A Storkey. Meta-learning in neural networks: A survey ieee transactions on pattern analysis and machine intelligence 1, 2021.

[68] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR, 2019.

[69] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[70] Zhiqiang Hu, Yihuai Lan, Lei Wang, Wanyu Xu, Ee-Peng Lim, Roy Ka-Wei Lee, Lidong Bing, and Soujanya Poria. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*, 2023.

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

[72] Hubert Hug and Rainer Schuler. Strategies for the development of a peptide computer. *Bioinformatics*, 17(4):364–368, 2001.

[73] Keishi Ishihara, Anssi Kanervisto, Jun Miura, and Ville Hautamaki. Multi-task learning with attention for end-to-end autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2902–2911, 2021.

[74] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[75] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[76] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[77] Adrián Javaloy and Isabel Valera. Rotograd: Gradient homogenization in multitask learning. *arXiv preprint arXiv:2103.02631*, 2021.

[78] Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual prompt tuning. In *European Conference on Computer Vision*, pages 709–727. Springer, 2022.

[79] Hua Jiang, Marc D Riedel, and Keshab K Parhi. Digital logic with molecular reactions. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 721–727. IEEE, 2013.

[80] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[81] Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems*, 34:1022–1035, 2021.

[82] Evgeny Katz and Vladimir Privman. Enzyme-based logic systems for information processing. *Chemical Society Reviews*, 39(5):1835–1857, 2010.

[83] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7482–7491, 2018.

[84] Eamonn Kennedy, Christopher E. Arcadia, Joseph Geiser, Peter M. Weber, Christopher Rose, Brenda M. Rubenstein, and Jacob K. Rosenstein. Encoding information in synthetic metabolomes. *PLOS ONE*, 14(7):1–12, 07 2019.

[85] Ahmad S Khalil and James J Collins. Synthetic biology: applications come of age. *Nature Reviews Genetics*, 11(5):367–379, 2010.

[86] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[87] Iasonas Kokkinos. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6129–6138, 2017.

[88] Prakash Kulkarni, Vitor BP Leite, Susmita Roy, Supriyo Bhattacharyya, Atish Mohanty, Srisairam Achuthan, Divyoj Singh, Rajeswari Appadurai, Govindan Rangarajan, Keith Weninger, et al. Intrinsically disordered proteins: ensembles at the limits of anfinsen's dogma. *Biophysics Reviews*, 3(1), 2022.

[89] Thaddeus D Ladd, Fedor Jelezko, Raymond Laflamme, Yasunobu Nakamura, Christopher Monroe, and Jeremy Lloyd O'Brien. Quantum computers. *nature*, 464(7285):45–53, 2010.

[90] Z Lan. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[91] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[92] Dawei Li, Xiaolong Wang, and Deguang Kong. Deeprebirth: Accelerating deep neural network execution on mobile devices. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[93] Hanxue Liang, Zhiwen Fan, Rishov Sarkar, Ziyu Jiang, Tianlong Chen, Kai Zou, Yu Cheng, Cong Hao, and Zhangyang Wang. M^3vit: Mixture-of-experts vision transformer for efficient multi-task learning with model-accelerator co-design. *arXiv preprint arXiv:2210.14793*, 2022.

[94] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. *Advances in neural information processing systems*, 30, 2017.

[95] Bo Liu, Yihao Feng, Peter Stone, and Qiang Liu. Famo: Fast adaptive multitask optimization. *Advances in Neural Information Processing Systems*, 36, 2024.

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

[97] Shikun Liu, Edward Johns, and Andrew J Davison. End-to-end multi-task learning with attention. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1871–1880, 2019.

[98] Shikun Liu, Edward Johns, and Andrew J Davison. End-to-end multi-task learning with attention. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1871–1880, 2019.

[99] Yen-Cheng Liu, Chih-Yao Ma, Junjiao Tian, Zijian He, and Zsolt Kira. Polyhistor: Parameter-efficient multi-task adaptation for dense vision tasks. *Advances in Neural Information Processing Systems*, 35:36889–36901, 2022.

[100] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[101] Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. *arXiv preprint arXiv:2106.04489*, 2021.

[102] Kevis-Kokitsi Maninis, Ilija Radosavovic, and Iasonas Kokkinos. Attentive single-tasking of multiple tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1851–1860, 2019.

[103] Lingchen Meng, Hengduo Li, Bor-Chun Chen, Shiyi Lan, Zuxuan Wu, Yu-Gang Jiang, and Ser-Nam Lim. Adavit: Adaptive vision transformers for efficient image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12309–12318, 2022.

[104] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[105] David AB Miller. Are optical transistors the logical next step? *Nature Photonics*, 4(1):3–5, 2010.

[106] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3994–4003, 2016.

[107] Takafumi Miyamoto, Shiva Razavi, Robert DeRose, and Takanari Inoue. Synthesizing biomolecule-based boolean logic gates. *ACS synthetic biology*, 2(2):72–82, 2013.

[108] Pim Moeskops, Jelmer M Wolterink, Bas HM Van Der Velden, Kenneth GA Gilhuijs, Tim Leiner, Max A Viergever, and Ivana Išgum. Deep learning for multi-task medical image segmentation in multiple modalities. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2016: 19th International Conference, Athens, Greece, October 17-21, 2016, Proceedings, Part II 19*, pages 478–486. Springer, 2016.

[109] Eslam Mohamed and Ahmad El Sallab. Spatio-temporal multi-task learning transformer for joint moving object detection and segmentation. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 1470–1475. IEEE, 2021.

[110] Carmen G Moles, Pedro Mendes, and Julio R Banga. Parameter estimation in biochemical pathways: a comparison of global optimization methods. *Genome research*, 13(11):2467–2474, 2003.

[111] Aviv Navon, Aviv Shamsian, Idan Achituve, Haggai Maron, Kenji Kawaguchi, Gal Chechik, and Ethan Fetaya. Multi-task learning as a bargaining game. *arXiv preprint arXiv:2202.01017*, 2022.

[112] Marina Neseem, Ahmed Agiza, and Sherief Reda. Adamtl: Adaptive input-dependent inference for efficient multi-task learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4729–4738, 2023.

[113] Marina Neseem and Sherief Reda. Adacon: Adaptive context-aware object detection for resource-constrained embedded devices. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2021.

[114] Lee Organick, Siena Dumas Ang, Yuan-Jyue Chen, Randolph Lopez, Sergey Yekhanin, Konstantin Makarychev, Miklos Z Racz, Govinda Kamath, Parikshit Gopalan, Bichlien Nguyen, et al. Random access in large-scale dna data storage. *Nature biotechnology*, 36(3):242–248, 2018.

[115] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural networks*, 113:54–71, 2019.

[116] Juan Manuel Parrilla-Gutierrez, Abhishek Sharma, Soichiro Tsuda, Geoffrey JT Cooper, Gerardo Aragon-Camarasa, Kevin Donkers, and Leroy Cronin. A programmable chemical computer with memory and pattern recognition. *Nature communications*, 11(1):1–8, 2020.

[117] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.

[118] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[119] Alex Prokup, James Hemphill, and Alexander Deiters. Dna computation: a photochemically controlled and gate. *Journal of the American Chemical Society*, 134(8):3810–3815, 2012.

[120] Oliver Purcell and Timothy K Lu. Synthetic analog and digital circuits for cellular computation and memory. *Current opinion in biotechnology*, 29:146–155, 2014.

[121] Pytorch mobile: End-to-end workflow from training to deployment for ios and android mobile devices, March 2023. `https://pytorch.org/mobile/android`.

[122] Lulu Qian and Erik Winfree. Scaling up digital circuit computation with dna strand displacement cascades. *Science*, 332(6034):1196–1201, 2011.

[123] Lulu Qian, Erik Winfree, and Jehoshua Bruck. Neural network computation with dna strand displacement cascades. *Nature*, 475(7356):368–372, 2011.

[124] Somayeh Bakhtiari Ramezani, Alexander Sommers, Harish Kumar Manchukonda, Shahram Rahimi, and Amin Amirlatifi. Machine learning algorithms in quantum computing: A survey. In *2020 International joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2020.

[125] Patricia Remon, Magnus Balter, Shiming Li, Joakim Andreasson, and Uwe Pischel. An all-photonic molecule-based d flip-flop. *Journal of the American Chemical Society*, 133(51):20742–20745, 2011.

[126] Eleanor Rieffel and Wolfgang Polak. An introduction to quantum computing for non-physicists. *ACM Computing Surveys (CSUR)*, 32(3):300–335, 2000.

[127] Sara Robinson. Emerging insights on limitations of quantum computing shape quest for fast algorithms. *SIAM News*, 36(1), 2003.

[128] Jacob K Rosenstein, Christopher Rose, Sherief Reda, Peter M Weber, Eunsuk Kim, Jason Sello, Joseph Geiser, Eamonn Kennedy, Christopher Arcadia, Amanda Dombroski, et al. Principles of information storage in small-molecule mixtures. *IEEE Transactions on NanoBioscience*, 19(3):378–384, 2020.

[129] Jacob K Rosenstein, Christopher Rose, Sherief Reda, Peter M Weber, Eunsuk Kim, Jason Sello, Joseph Geiser, Eamonn Kennedy, Christopher Arcadia, Amanda Dombroski, et al. Principles of information storage in small-molecule mixtures. *IEEE Transactions on NanoBioscience*, 19(3):378–384, 2020.

[130] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[131] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[132] Sebastian Ruder, Joachim Bingel, Isabelle Augenstein, and Anders Søgaard. Latent multi-task architecture learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4822–4829, 2019.

[133] M Sakthi Balan and Helmut Jürgensen. On the universality of peptide computing. *Natural Computing*, 7:71–94, 2008.

[134] MM Savchuk and AV Fesenko. Quantum computing: Survey and analysis. *Cybernetics and Systems Analysis*, 55:10–21, 2019.

[135] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. *Advances in neural information processing systems*, 31, 2018.

[136] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *towards data science*, 6(12):310–316, 2017.

[137] Yichen Shen, Nicholas C Harris, Scott Skirlo, Mihika Prabhu, Tom Baehr-Jones, Michael Hochberg, Xin Sun, Shijie Zhao, Hugo Larochelle, Dirk Englund, et al. Deep learning with coherent nanophotonic circuits. *Nature photonics*, 11(7):441–446, 2017.

[138] Piro Siuti, John Yazbek, and Timothy K Lu. Engineering genetic circuits that compute and remember. *Nature protocols*, 9(6):1292–1300, 2014.

[139] Trevor Standley, Amir Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? In *International Conference on Machine Learning*, pages 9120–9132. PMLR, 2020.

[140] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for modern deep learning research. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 13693–13696, 2020.

[141] Robin Strudel, Ricardo Garcia, Ivan Laptev, and Cordelia Schmid. Segmenter: Transformer for semantic segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 7262–7272, 2021.

[142] Ximeng Sun, Rameswar Panda, Rogerio Feris, and Kate Saenko. Adashare: Learning what to share for efficient deep multi-task learning. *Advances in Neural Information Processing Systems*, 33:8728–8740, 2020.

[143] Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. Vl-adapter: Parameter-efficient transfer learning for vision-and-language tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5227–5237, 2022.

[144] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[145] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR, 2021.

[146] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[147] Ron Unger and John Moult. Towards computing with proteins. *Proteins: Structure, Function, and Bioinformatics*, 63(1):53–64, 2006.

[148] Simon Vandenhende, Stamatios Georgoulis, Wouter Van Gansbeke, Marc Proesmans, Dengxin Dai, and Luc Van Gool. Multi-task learning for dense prediction tasks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3614–3633, 2021.

[149] Simon Vandenhende, Stamatios Georgoulis, and Luc Van Gool. Mti-net: Multi-scale task interaction networks for multi-task learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV 16*, pages 527–543. Springer, 2020.

[150] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[151] Vuzix m4000 smart glasses, March 2023. `https://www.vuzix.com/products/m4000-smart-glasses`.

[152] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, et al. Deep high-resolution representation learning for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3349–3364, 2020.

[153] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, et al. Deep high-resolution representation learning for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3349–3364, 2020.

[154] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 568–578, 2021.

[155] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424, 2018.

[156] Zirui Wang, Zihang Dai, Barnabás Póczos, and Jaime Carbonell. Characterizing and avoiding negative transfer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11293–11302, 2019.

[157] Zirui Wang, Yulia Tsvetkov, Orhan Firat, and Yuan Cao. Gradient vaccine: Investigating and improving multi-task optimization in massively multilingual models. *arXiv preprint arXiv:2010.05874*, 2020.

[158] Wilfried Weber and Martin Fussenegger. Emerging biomedical applications of synthetic biology. *Nature Reviews Genetics*, 13(1):21–35, 2012.

[159] Neil HE Weste and David Harris. *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.

[160] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in Neural Information Processing Systems*, 34:12077–12090, 2021.

[161] Dan Xu, Wanli Ouyang, Xiaogang Wang, and Nicu Sebe. Pad-net: Multi-tasks guided prediction-and-distillation network for simultaneous depth estimation and scene parsing. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 675–684, 2018.

[162] Shuo Yang, Bas WA Bögels, Fei Wang, Can Xu, Hongjing Dou, Stephen Mann, Chunhai Fan, and Tom FA de Greef. Dna as a universal chemical substrate for computing and data storage. *Nature Reviews Chemistry*, pages 1–16, 2024.

[163] Hanrong Ye and Dan Xu. Inverted pyramid multi-task transformer for dense scene understanding. In *European Conference on Computer Vision*, pages 514–530. Springer, 2022.

[164] Hanrong Ye and Dan Xu. Taskprompter: Spatial-channel multi-task prompting for dense scene understanding. In *The Eleventh International Conference on Learning Representations*, 2022.

[165] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014.

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

[167] Zhongzhi Yu, Yonggan Fu, Sicheng Li, Chaojian Li, and Yingyan Lin. Mia-former: efficient and robust vision transformers via multi-grained input-adaptation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8962–8970, 2022.

[168] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*, 2021.

[169] Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3712–3722, 2018.

[170] Renrui Zhang, Jiaming Han, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, Peng Gao, and Yu Qiao. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *arXiv preprint arXiv:2303.16199*, 2023.

[171] Yan Zhang, Zishun Liu, Somsak Swaddiwudhipong, Haiyan Miao, Zhiwei Ding, and Zhengzhi Yang. ph-sensitive hydrogel for micro-fluidic valve. *Journal of Functional Biomaterials*, 3(3):464–479, 2012.

[172] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 34(12):5586–5609, 2021.

[173] Yuxin Zhang, Lirui Zhao, Mingbao Lin, Yunyun Sun, Yiwu Yao, Xingjia Han, Jared Tanner, Shiwei Liu, and Rongrong Ji. Dynamic sparse no training: Training-free fine-tuning for sparse llms. *arXiv preprint arXiv:2310.08915*, 2023.

[174] Quan Zhou, Wenbing Yang, Guangwei Gao, Weihua Ou, Huimin Lu, Jie Chen, and Longin Jan Latecki. Multi-scale deep context convolutional neural networks for semantic segmentation. *World Wide Web*, 22:555–570, 2019.

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

[176] Beier Zhu, Yulei Niu, Yucheng Han, Yue Wu, and Hanwang Zhang. Prompt-aligned gradient for prompt tuning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15659–15669, 2023.