Efficient Computation of Collatz Sequence Stopping Times: A Machine Learning Guided Algorithmic Approach

By: **Eyob Solomon Getachew**

Advisor: **Dr. Beakal Gizachew Assefa**

A thesis submitted to the School of Information Technology and Engineering in partial fulfillment of the requirements for the degree of **Master of Science in Artificial Intelligence**

College of Technology and Built Environment

Addis Ababa University, Addis Ababa, Ethiopia

September, 2025

Approval

This is to certify that this thesis titled ***Efficient Computation of Collatz Sequence Stopping Times: A Machine Learning Guided Algorithmic Approach*** is prepared by ***Eyob Solomon Getachew*** and submitted in partial fulfillment of the thesis-option requirements for the Degree of Master of Science in Artificial Intelligence at the School of Information Technology & Engineering, Addis Ababa University.

Advisor

Dr. Beakal Gizachew		
Name	Signature	Date

Examiner (External)

Dr. Mesfin Abebe		13/10/25
Name	Signature	Date

Examiner (Internal)

Dr. Adane Letta		15/10/25
Name	Signature	Date

Chairperson

Dr. Sileshi Demesie		
Name	Signature	Date

Publication Notice

This thesis incorporates content that has been published as part of the author's MSc research conducted under the supervision of Dr. Beakal Gizachew Assefa at Addis Ababa University. The research results presented herein were originally disseminated as a preprint on *arXiv* and subsequently peer-reviewed and published in *IEEE Access*.

- E. S. Getachew and B. G. Assefa, "Efficient Computation of Collatz Sequence Stopping Times: A Novel Algorithmic Approach," *IEEE Access*, vol. 13, 2025. DOI: 10.1109/ACCESS.2025.3548031 (Originally released as *arXiv preprint* arXiv:2501.04032)

The published article and this thesis originate from the same body of research. The thesis version provides an expanded and unified presentation, including additional analyses, experiments, and contextual discussion beyond the scope of the publication.

All necessary permissions for inclusion have been obtained. The reuse of this material complies with IEEE and arXiv self-archiving policies, which allow authors to incorporate their own published work into academic theses and dissertations with proper acknowledgment.

Abstract

The Collatz conjecture, first proposed in 1937, remains one of the most iconic unsolved problems in mathematics. It concerns sequences generated by repeatedly applying a simple iterative rule: halving even numbers and mapping odd numbers to $3n + 1$. The conjecture asserts that every natural number, when subjected to this process, eventually reaches the number 1. Although deceptively straightforward, the problem has resisted proof for nearly nine decades despite extensive computational verification. This thesis introduces a novel *machine-learning-guided, structure-aware algorithm* for computing Collatz stopping times. By analyzing statistical patterns and hierarchical regularities identified through regression and clustering experiments, the algorithm exploits the inverse Collatz tree to bypass redundant even paths and minimize repetitive computation. The resulting method achieves a consistent **28% reduction in iteration count** relative to state-of-the-art algorithms, and an **average execution-time improvement of about 57%**. Building on this foundation, the algorithmic concept was further adapted into a novel *Collatz-based regularization* framework for deep learning. The approach introduces a bounded, deterministic penalty derived from the stopping time of the mean absolute model weights and applies it as a norm-like term in the loss function. When evaluated across image (CNN), tabular (FCNN), and time-series (RNN) benchmarks, the proposed regularizer achieved stable and superior or comparable performance under varying regularization strengths—maintaining convergence where conventional ℓ_1, ℓ_2, and Elastic Net penalties degraded significantly at higher strengths. Overall, the findings demonstrate that integrating machine learning insights into algorithmic design can yield substantial computational efficiency, and that deterministic number-theoretic dynamics can serve as a robust, mathematically interpretable regularization mechanism for modern neural networks.

Keywords: Collatz conjecture, stopping time, structure-aware algorithm, execution-time efficiency, scalability, ML-guided algorithm design, bounded regularization, deep learning.

Acknowledgements

First and foremost, I offer my deepest gratitude to the **Holy Trinity** and **Saint Virgin Mary** for granting me strength, wisdom, and perseverance throughout this journey. I also give heartfelt thanks to all of **God's army**, including the **Angels** and **Saints**, whose intercession, protection, and unseen guidance have accompanied me in every step. Their divine support has been my constant source of courage, clarity, and peace.

I wish to express my profound appreciation to my advisor, Dr. Beakal Gizachew, for his invaluable guidance, constructive feedback, and continuous encouragement throughout this research. His mentorship has greatly shaped both the technical depth and scholarly rigor of this thesis.

My heartfelt thanks go to my friends, Dr. Kibret Yilak and Mr. Amanuel Negash, whose steadfast encouragement and belief in my ability played a decisive role in motivating me to complete this work. I am deeply grateful to my wife, Elizabeth Kassaye, whose patience, love, and support gave me strength during the most demanding stages of this journey. Without her encouragement, this thesis would not have been possible. I also dedicate this achievement to my beloved mother, whose love, prayers, and lifelong sacrifices built the foundation for all I have accomplished.

I would also like to extend my sincere gratitude to the SITE Dean, Dr. Fantahun Bogale, and the Postgraduate Coordinator, Dr. Sileshi Demesie, for their technical and administrative guidance. I am equally thankful to the external examiner, Dr. Mesfin Abebe, and the internal examiner, Dr. Adane Letta, for their insightful comments and constructive feedback, which significantly improved the quality of this thesis. My appreciation also goes to all the lecturers who taught and guided me during my postgraduate studies.

To everyone who contributed, directly or indirectly, to this journey—thank you, and may God bless you all.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Adam Adaptive Moment Estimation.

AdamW Adam with Weight Decay.

AI Artificial Intelligence.

CNN Convolutional Neural Network.

CORR-Reg Correlation-Based Regularization.

DBSCAN Density-Based Spatial Clustering of Applications with Noise.

EBR Evidence-Based Regularization.

ECG Electrocardiogram.

EEG Electroencephalogram.

FCNN Fully Connected Neural Network.

FS-EB Function-Space Evidence-Based Regularization.

HTR Hessian Trace Regularization.

INCA Iterative Neighborhood Component Analysis.

kNN k-Nearest Neighbors.

L1 L1 Regularization (Lasso).

L2 L2 Regularization (Ridge/Weight Decay).

MNIST Modified National Institute of Standards and Technology database.

MSE Mean Squared Error.

PCC Proof-of-Collatz Conjecture.

PoW Proof-of-Work.

ReLU Rectified Linear Unit.

RMSE Root Mean Square Error.

RNN Recurrent Neural Network.

ST Stopping Time (Collatz sequence length).

Chapter 1 Introduction

1.1 Background of the Study

The *Collatz conjecture*, also known as the *3n + 1 problem*, is a longstanding unsolved question in mathematics first proposed by Lothar Collatz in 1937 [1]. Despite its deceptively simple formulation, the problem has resisted proof for nearly a century and continues to intrigue mathematicians, computer scientists, and hobbyists alike.

At its core, the conjecture concerns the behavior of sequences generated by a simple iterative rule applied to any positive integer n, defined as:

$$C(n) = \begin{cases} \dfrac{n}{2}, & \text{if } n \text{ is even,} \\ 3n+1, & \text{if } n \text{ is odd.} \end{cases} \tag{1.1}$$

Repeatedly applying this rule produces a sequence known as the *Collatz sequence*. The central question is whether every such sequence, regardless of the starting value n, eventually reaches 1.

For example, beginning with $n = 13$, the sequence evolves as:

$$13 \to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$$

Once the sequence reaches 1, it enters the well-known **trivial cycle**:

$$1 \to 4 \to 2 \to 1$$

To date, no other (non-trivial) cycles have been discovered, and no starting value has been shown to diverge to infinity. Despite extensive empirical validation across enormous input ranges, a formal proof or counterexample remains elusive. This paradox of simplicity and depth has made the Collatz conjecture one of the most iconic unsolved problems in mathematics.

1.2 Why It Is a Conjecture

In mathematics, a *conjecture* is a proposition that is widely believed to be true but has not yet been formally proven. The Collatz conjecture asserts that every positive integer, when repeatedly transformed according to Equation (1.1), eventually reaches the number 1. Despite extensive numerical testing and widespread confidence in its validity, no general proof exists.

To confirm the conjecture, one must prove that:

- Every starting value $n \in \mathbb{N}$ eventually leads to 1.

Conversely, to disprove it, one of the following would suffice:

- The existence of a **non-trivial cycle**—a sequence that loops without including 1.
- A **divergent trajectory**—a sequence that grows indefinitely without returning to 1 or forming a cycle.

So far, no such counterexamples have been found. Computational efforts have verified the conjecture for input values up to $2^{100,000} - 1$ [2], but such tests—while impressive—cannot constitute a general proof. This reliance on exhaustive computation highlights both the practical importance of algorithmic efficiency and the deeper mathematical challenge of the problem.

1.3 Statement of the Problem

Despite extensive computational efforts, existing approaches for computing Collatz stopping times remain constrained by redundant traversal and limited structural reuse. State-of-the-art implementations such as the Codeword Encoding approach [2] and bitwise variants [3] improve arithmetic efficiency but still iterate through every intermediate value without exploiting the hierarchical regularities of the Collatz tree. As a result, these algorithms perform

large numbers of unnecessary operations—particularly during long even sequences—leading to high iteration counts and limited scalability for large input ranges.

Recent structural investigations [4, 5, 6] reveal that Collatz sequences exhibit repetitive and hierarchical behavior, including overlapping subpaths and patterned even-odd alternations. However, these findings have not been operationalized into practical algorithms capable of leveraging these regularities. This disconnect between structural understanding and computational design highlights a key methodological gap: the lack of an algorithm that translates these insights into measurable computational efficiency. A related limitation lies in the absence of **machine learning–guided** approaches to Collatz computation. While most studies rely purely on deterministic or symbolic reasoning, preliminary ML experiments (as explored in this thesis) suggest that stopping times follow learnable, non-random patterns. Yet, no prior work has systematically used ML-derived insight to inform algorithmic design—an opportunity to merge pattern discovery with deterministic computation.

Finally, in the context of machine learning itself, classical regularization techniques such as ℓ_1, ℓ_2, and Elastic Net [7, 8, 9] are widely used to control overfitting but often degrade under strong penalty strengths or require sensitive hyperparameter tuning. The deterministic yet non-linear nature of Collatz dynamics offers a mathematically grounded and bounded alternative signal that remains unexplored as a regularization mechanism.

Accordingly, this thesis addresses the following two interconnected problems:

1. **Algorithmic inefficiency:** Develop a structure-aware algorithm for computing Collatz stopping times that reduces redundant computations and scales efficiently, guided by empirical and ML-based analysis of sequence regularities.

2. **Regularization stability:** Formulate and evaluate a deterministic, Collatz-derived regularization method that improves model stability and generalization relative to classical norm-based penalties.

1.4 Research Questions

The following research questions guide this study:

1. How can machine learning models be used to identify and interpret structural patterns in the Collatz sequence, and how can these patterns be leveraged to design an algorithm that minimizes redundant computations in stopping time calculations?

2. What is the extent of computational efficiency gained by the proposed algorithm compared to existing brute-force and bitwise methods, in terms of iteration count and runtime performance?

3. How can features or dynamics derived from the Collatz sequence, such as stopping times, be leveraged within deep learning algorithms to enhance model performance?

1.5 Research Objectives

1.5.1 General Objective

To develop an optimized algorithm for computing the stopping time of natural numbers in the Collatz sequence, and investigate its potential applications in machine learning.

1.5.2 Specific Objectives

- To analyze structural patterns within the Collatz sequence through computational and visual exploration.

- To design an efficient algorithm that minimizes redundant operations during stopping time computation by leveraging those patterns.

- To evaluate the correctness, performance, and scalability of the proposed algorithm against existing methods.

- To investigate whether machine learning models can detect structural patterns and predict stopping times with reduced computational effort.

- To explore the integration of Collatz-based features or regularization strategies into deep learning models and assess their impact on predictive performance.

1.6 Significance of the Study

This study advances the computational investigation of the Collatz conjecture by introducing an algorithm that significantly reduces the number of iterations required to compute stopping times. Unlike existing methods that process each number step-by-step, the proposed algorithm leverages structural regularities in the Collatz tree to eliminate redundant computation. This enables more scalable verification of the conjecture, especially for extremely large inputs, and offers a foundation for future high-throughput explorations in number theory.

Beyond computational efficiency, the research demonstrates that the structural dynamics of the Collatz sequence can also be integrated into machine learning workflows. Specifically, we introduce a Collatz-based regularization term derived from the *mean of the absolute values of model weights*. Empirical results across three modalities (image classification (MNIST, CNN), tabular regression (California Housing, FCNN), and time-series classification (ECG5000, RNN)) show that this regularizer consistently outperforms ℓ_1, ℓ_2, and Elastic Net. The improvements are most pronounced at higher regularization strengths, where conventional methods often collapse, while the Collatz regularizer remains stable. This interdisciplinary link suggests that deterministic sequence properties can be repurposed as effective, robust tools for enhancing model performance in machine learning algorithms.

More broadly, this research illustrates how machine learning can uncover structural patterns that inspire more efficient rule-based algorithms, while those same algorithms, rooted in number theory, can, in turn, be used to improve the performance of machine learning models.

1.7 Contributions

This research makes significant algorithmic and interdisciplinary contributions to computational number theory and machine learning:

1. **Structure-Aware Collatz Algorithm**: A novel computational method that reduces iteration counts by approximately 28% compared to state-of-the-art methods; maintaining this gain even at extreme input magnitudes. (Section 3.4). The algorithm:
   - Leverages hierarchical patterns in the Collatz tree
   - Achieves logarithmic complexity in both average ($O(\log n)$) and worst-case scenarios
2. **Theoretical Framework**: Comprehensive complexity analysis proving the algorithm's efficiency gains, with empirical validation across one billion test cases showing exact agreement with brute-force results (Section 4.1).
3. **Empirical Verification Breakthrough**: Extended Collatz verification to $[2^{100,000}, 2^{100,000} + 100,000]$, demonstrating:
   - Consistent stopping behavior beyond the previous upper bound of $2^{100,000} - 1$
   - Algorithm stability with inputs of $\sim$ 30,000 digits
   - No non-trivial cycles detected in the extended range

   The Julia implementation (Section 4.3) processed all 100,000 numbers with 100% agreement against theoretical predictions.
4. **Machine Learning Integration**: A new regularization technique derived from Collatz dynamics that:
   - Uses mean absolute weights for stability and a deterministic stopping-time transformation
   - Outperforms ℓ_1, ℓ_2, and Elastic Net across all tested scenarios

- Remains robust at higher regularization strengths, where conventional methods often collapse
- Demonstrates cross-domain effectiveness (images, tabular data, time-series)

Full experimental results are presented in Section 4.4.

5. **Interdisciplinary Methodology**: A replicable framework for:
   - Converting mathematical structures into efficient algorithms
   - Applying number-theoretic concepts to ML challenges
   - Validating hybrid approaches through rigorous experimentation

1.8 Scope of the Study

This research establishes a foundation for Collatz sequence analysis through two synergistic objectives: advancing computational efficiency in fundamental operations and demonstrating novel applications of number theory in machine learning. The study prioritizes algorithmic purity by developing a structure-aware approach that achieves efficiency gains through mathematical insights rather than implementation-level optimizations.

Scalar Computation Framework: The algorithm is deliberately designed for exact integer arithmetic in sequential execution environments, omitting memoization and parallelization to isolate the impact of structural optimizations. This constrained implementation serves as both a performance baseline and a template for future enhancements, having successfully extended verification to $2^{100,000} + 100,000$ without batch-processing optimizations. The design explicitly leaves integration of distributed computing techniques and result caching (e.g., for contiguous number ranges) as open opportunities for large-scale verification efforts beyond the current upper bounds.

Mathematical Boundaries: The work is strictly confined to the canonical Collatz formulation, excluding generalized variants (e.g., $3n + k$ systems) and theoretical proof attempts.

This deliberate constraint enables focused analysis of computational patterns while maintaining verifiable correctness up to $2^{100,000} + 100,000$, extending the prior verification limits by 100,000 numbers.

Machine Learning Integration: The study pioneers the application of Collatz dynamics as a novel regularization strategy, evaluated across three foundational architectures (CNNs, FCNNs, RNNs) and benchmarked against ℓ_1, ℓ_2, and Elastic Net. All experiments followed a consistent protocol of 20 training epochs at two regularization strengths ($\lambda \in \{0.01, 0.10\}$), averaged over 10 independent runs for robustness. Results demonstrated that the Collatz-based regularizer consistently outperformed the baselines, with particularly strong stability under higher penalty strengths where ℓ_1 and Elastic Net collapsed. These controlled experiments establish proof-of-concept efficacy and motivate further exploration of number-theory-inspired techniques for tackling core machine learning challenges such as overfitting mitigation.

1.9 Organization of the Study

The thesis is structured into six chapters. The first chapter introduces the Collatz conjecture and explains the motivation for studying its stopping time problem from both algorithmic and machine learning perspectives. It also outlines the research questions and objectives that guide the investigation. Relevant studies on Collatz sequence computation and classical regularization techniques in machine learning are reviewed in Chapter 2, establishing the theoretical and methodological foundation for the proposed approaches. The methodological framework, presented in Chapter 3, combines exploratory analysis, structural modeling, and algorithm design. This section also describes the formulation of the Collatz-based regularization method and the computational setup used for experiments. Experimental findings are discussed in Chapter 4, where both the proposed algorithm and the regularization approach are evaluated. Finally, Chapter 5 summarizes the key results, provides recommendations for further research, and outlines directions for extending the algorithm and regularization approach to other mathematical and learning contexts.

Chapter 2 Literature Review

The literature relevant to this study encompasses two distinct bodies of work. The first concerns the Collatz conjecture itself, which can be broadly categorized into three main areas:

(A) **Theoretical insights and structural analysis** – examining the conjecture's mathematical foundations and the structural intricacies of its behavior, with the aim of uncovering patterns, proposing generalizations, and assessing probabilistic models.

(B) **Computational methods and algorithmic efficiency** – exploring algorithmic designs that push the limits of numeric verification through optimized traversal strategies, bitwise operations, and symbolic encoding schemes.

(C) **Practical applications and implications** – leveraging the conjecture's iterative properties for real-world problems such as cryptography, blockchain consensus, and digital watermarking, demonstrating its relevance beyond pure mathematics.

The second body of literature addresses **regularization techniques in machine learning**, which is included here to provide the necessary background for evaluating the proposed Collatz-based regularization method developed in this thesis. This body of work reviews foundational and advanced approaches for improving model generalization. The following sections provide a detailed review of each category.

2.1 Theoretical Insights and Structural Analysis

Recent investigations into the Collatz conjecture have employed both structural and probabilistic approaches, seeking patterns and behaviors that might help reveal or approximate a proof. Techniques such as bottom-up analysis, sequence generalization, probabilistic modeling, and logical frameworks have been explored to better understand convergence properties, stopping times, and potential counterexamples.

The bottom-up approach constructs Collatz sequences in reverse, starting from $n = 1$ and applying inverse transformations to map all integers back to this origin. By doubling even numbers and modifying odd numbers with $(n-1)/3$, this method forms a tree-like structure that visualizes convergence paths to 1. This analysis yields algebraic formulas generalizing the sequence's behavior, offering insight into its convergence properties [4].

An approximation method analyzes stopping times in the Collatz sequence by defining transformations as $T(m) = m/2$ for even m and $T(m) = (3m + 1)/2$ for odd m. This method introduces probabilistic bounds, showing that average stopping times fall within a logarithmic framework and providing insights into potential asymptotic behaviors of the conjecture. The model provides lower bounds, contributing insights into the asymptotic behavior of stopping times if the conjecture holds [5].

A framework introduced by Ebnenasir [6] conceptualizes the Collatz process as a concurrent program composed of *convergence stairs*, where each stair groups numbers sharing the same stopping time. This formulation provides a formal specification of convergence using self-stabilizing program theory and represents stairs as infinite binary trees, offering an analytical view of the sequence's structural organization. While primarily theoretical in its motivation, the study also establishes the logical foundations for algorithmic generation and verification of stair membership, reinforcing the view that Collatz stopping times exhibit clustering behavior.

The generalization $3n+3k$, where $k \in \mathbb{N}$, extends the Collatz sequence's behavior. For $k = 0$, the sequence mirrors the familiar 3n + 1 pattern with the cycle 4, 2, 1. Higher values of k produce new periodic sequences, with formulas detailing their stopping times, presenting a broader framework for Collatz-like transformations [10].

An extensive review of computational and theoretical efforts to find Collatz counterexamples shows no success in uncovering a counterexample. The analysis also suggests that iterative reduction strategies might either contribute to proving the conjecture or demonstrate its

potential as an unprovable truth in mathematics, enriching its theoretical study [11].

A statistical analysis assesses the probability of infinite stopping times in the Collatz sequence, which would contradict the conjecture. The results indicate that the likelihood of divergence is astronomically low, analogous to selecting a specific atom from the entire observable universe, reinforcing the assumption that all sequences eventually converge to 1 [12].

A logical and probabilistic approach to proving the Collatz conjecture addresses the absence of non-trivial infinite loops and posits that all natural numbers eventually converge to 1. A probabilistic model further supports this by demonstrating that the likelihood of divergence approaches zero as iterations increase, providing structured reasoning for the conjecture's validity [13].

Link to this work: Building on these bottom-up and hierarchical perspectives, this thesis treats the inverse Collatz tree as an explicit computational object and, in Chapter 3, develops a structure-aware traversal that alternates inverse $3n+1$ steps with collapsing even runs in a single move by dividing by the largest power of two dividing n, thereby turning structural regularities into concrete iteration savings evaluated in Chapter 4.

The studies reviewed in this subsection primarily explore the theoretical, structural, and probabilistic dimensions of the Collatz conjecture. Table 1 summarizes key contributions from these works, emphasizing their analytical focus, limitations, and their relevance to the present study. Collectively, these theoretical investigations highlight hierarchical and probabilistic patterns within the Collatz sequence that motivate the algorithmic and empirical directions pursued in subsequent sections.

Table 1: Summary of major theoretical and structural studies on the Collatz conjecture.

Study	**Methodology**	**Limitations**	**Relevance to This Work**
Abascal et al. [4]	Constructs the inverse Collatz tree to analyze convergence structure.	Provides theoretical insight but does not offer a complete proof of convergence.	Forms the conceptual basis for the structure-aware traversal proposed in this thesis.
Inselmann et al. [5]	Develops a probabilistic model estimating expected stopping times.	Offers statistical bounds without establishing a rigorous analytical resolution.	Supports regression-based analysis of logarithmic growth patterns.
Ebnenasir et al. [6]	Conceptualizes the Collatz process as a concurrent program ("convergence stairs") that groups numbers sharing the same stopping time, revealing clustering patterns in the sequence.	Provides a formal specification but does not deliver a complete theoretical resolution of the conjecture.	Highlights the clustering nature of stopping times and establishes the logical foundation for their later algorithmic generation.
Boulkaboul et al. [10]	Extends the conjecture to generalized $3n + 3k$ forms to study periodicity.	Examines variants but does not resolve the classical conjecture.	Demonstrates mathematical generalizability of Collatz dynamics.
Clay et al. [11]	Reviews theoretical and computational efforts on the conjecture.	Analytical synthesis only; confirms unresolved theoretical status.	Frames the motivation for further algorithmic exploration.
Nicola et al. [12]	Uses probabilistic reasoning to estimate divergence likelihood.	Probabilistic approach; lacks a formal theoretical proof.	Justifies the assumption of convergence in experimental design.
Zhou et al. [13]	Combines logical and probabilistic reasoning to argue convergence.	Provides a plausibility argument rather than a complete proof.	Strengthens theoretical confidence in assuming universal convergence.

2.2 Computational Methods and Algorithmic Efficiency

Significant computational advances have allowed the Collatz conjecture to be empirically verified for extremely large numbers, employing a variety of algorithmic approaches to push the limits of computational capabilities. Techniques such as bitwise operations, binary representation, and automata-based processing have emerged as essential tools in this verification.

Leveraging bitwise operations and disk-based storage, a high-capacity algorithm represents numbers in binary to efficiently perform $3n+1$ computations on a bit-by-bit basis. This method has enabled verification for numbers up to 100,000 bits, confirming the conjecture for values up to $2^{100000} - 1$ and demonstrating its effectiveness for unprecedented numeric ranges [2]. Similarly, another binary-based algorithm transforms integers to binary form, applying bitwise operations and reverse shifts to verify the conjecture for sequences up to 3,000 bits. This approach confirms bounded growth, with no cycles beyond expected bounds, reinforcing the conjecture's validity [3]. (Full implementation details of these bitwise methods are discussed in Section 2.5.)

In addition to binary-based techniques, novel algorithms and supporting theorems have been developed to analyze structural relationships within Collatz sequences. These include directed graph visualizations, iteration analyzers, and peak-value finders that collectively explore the interplay between natural numbers, peak values, and iteration counts, offering computational insights into the conjecture's complex structure [14].

An additional computational framework, known as the *Convergence Stairs* approach [6], formalizes the Collatz process as a concurrent program and introduces verified algorithms that generate all numbers belonging to a specific stair—sets of integers sharing the same stopping time. Each stair is defined and validated using Binary Verification Codes, enabling recursive enumeration of its members through provably sound and complete procedures. This algorithmic realization of the theoretical model demonstrates that stopping times form natural clusters rather than random distributions, providing both structural insight and

computational verification that complement the bitwise and automata-based strategies.

Automata-based verification methods provide further efficiency by processing transformations in batches rather than individually. One such method organizes a "Collatz tree" by residue classes, enabling certain values to be processed directly without performing all intermediate transformations. By segmenting repetitive patterns, this design significantly optimizes the verification process and offers a scalable framework for large-scale computation [15].

Table 2: Summary of major computational approaches developed for large-scale verification and analysis of the Collatz conjecture.

Method	**Core Idea / Mechanism**	**Limitations**
Codeword Encoding [2]	Encodes Collatz sequences into symbolic "code words" using bitwise logic to compress even runs and reduce storage requirements.	Time complexity remains proportional to sequence length; high memory usage due to storage of encoded symbols.
Bitwise Approach [3]	Implements $3n+1$ and $n/2$ transformations using hardware-level bitwise operations for faster computation.	Processes every step sequentially without redundancy elimination or structure reuse.
Automata / Residue Class Method [15]	Groups inputs by residue classes and precomputes transitions via finite automata for batched evaluation.	Heavy preprocessing overhead; limited flexibility for arbitrary or dynamic inputs.
Convergence "Stairs" [6]	Groups natural numbers by identical stopping times into layered "stairs," revealing clustering behavior and enabling algorithmic generation within each stair.	Algorithmic formulation remains exponential in time and lacks scalability for high-depth enumeration.
Other Structural Analyses [14, 4]	Introduce graph-based and inverse-tree representations to study structural relationships among sequence elements.	Primarily theoretical; do not directly yield performance improvements for large-scale computation.

Link to this work. While these approaches achieve impressive computational scale, the method proposed in this thesis combines bit-level efficiency with structural pruning strategies derived from the inverse Collatz tree. This hybrid design aims to retain the speed advantages of binary manipulation while reducing unnecessary computations through targeted branch elimination, as elaborated in Chapter 3.

The computational strategies reviewed above are summarized in Table 2. Each method is characterized by its underlying mechanism and limitations, highlighting the evolution from low-level bitwise optimization to structure-aware formulations such as automata-based and convergence-stair algorithms. These studies collectively reflect the ongoing effort to extend Collatz verification toward larger input scales while exploring the sequence's inherent structural regularities.

2.3 Applications and Practical Implications

Beyond mathematical verification, the conjecture's iterative properties have found applications in various areas, including blockchain consensus mechanisms, cryptographic security, digital watermarking, image encryption, and medical diagnostics.

Studies on Proof-of-Collatz Conjecture (PCC) algorithms in blockchain demonstrate the conjecture's potential as a consensus mechanism, effectively replacing traditional Proof-of-Work (PoW). By leveraging the unpredictable stopping times of Collatz sequences, the PCC approach achieves stable execution times and reduces computational costs, offering an efficient alternative to PoW in private blockchain environments [16].

In cryptographic security, Collatz sequences are applied in both audio and image encryption, as well as hash function design. Collatz-based algorithms in audio encryption convert speech signals into unintelligible forms using variable-length encoding, achieving high entropy and low data correlation with the original audio to enhance transmission security [17]. Similarly, an encrypted audio dataset leverages these sequence transformations to provide high-entropy audio signals that obscure original content, creating a valuable resource for cryptanalysis

testing [18]. For image encryption, a chaotic logistic map-based scheme incorporates Collatz sequences to generate high-entropy encryption keys, significantly improving throughput and resilience against cryptographic attacks [19]. Additionally, a Collatz-based hash function merges chaotic properties with sequence unpredictability, achieving enhanced randomness, strong diffusion, and improved collision resistance, outperforming traditional hash functions like SHA-2 and SHA-3 [20].

In digital watermarking, a novel method embeds watermarks by calculating the Collatz degree of each pixel in an image, enhancing visual quality and providing a secure, low-complexity solution suitable for real-time applications. This approach supports efficient image authentication through a pseudo-random number generator for block-wise embedding [21]. Another image encryption scheme transforms images into encrypted audio by encoding pixel values through Collatz transformations. This combined scrambling and diffusion process yields high entropy and strong key sensitivity, offering robust protection against cryptographic attacks while allowing reversible recovery [22].

In medical diagnostics, the Collatz pattern is applied to schizophrenia detection by transforming EEG signals into unique feature vectors. Using maximum absolute pooling and iterative neighborhood component analysis (INCA), this technique selects clinically significant features and classifies them with k-nearest neighbors (kNN), providing a high-accuracy, efficient tool for automated EEG-based diagnosis [23].

Link to this work and transition. The diversity of these applications illustrates how properties of the Collatz iteration can be embedded into computational frameworks that are otherwise unrelated to its original mathematical formulation. This conceptual adaptability is directly relevant to the present study, where Collatz-derived dynamics are incorporated into a regularization term within machine learning models. Although no prior work connects the Collatz conjecture to machine learning regularization, understanding existing regularization strategies is essential for positioning and evaluating the proposed method. Accordingly, the next section reviews foundational and advanced regularization techniques that form the

methodological backdrop for the machine learning component of this thesis.

2.4 Regularization in Machine Learning

Regularization is a key mechanism for improving the generalization of machine learning and deep learning models by mitigating overfitting and controlling model complexity. Over time, a wide variety of regularization strategies have emerged, ranging from classical parameter-norm penalties to adaptive, curvature-based, and augmentation-driven approaches. At a high level, these can be grouped into: (1) norm-based methods such as ℓ_1 (Lasso) [7] and ℓ_2 (Ridge) [8], (2) structured and adaptive penalties that modulate regularization strength based on data or network statistics [24, 25], (3) techniques that shape the optimization trajectory via curvature or constraint-based penalties [26, 27], and (4) methods that inject diversity through data or feature augmentation [28, 29]. These categories differ in formulation and intended effect (some promote sparsity, others encourage flat minima or robustness to perturbations) and are often combined in practice. While many advanced strategies exist, the experimental evaluation in this thesis focuses on ℓ_2 regularization as a baseline to isolate and clearly assess the effect of the proposed Collatz-based penalty.

2.4.1 Norm-Based and Adaptive Regularization

Norm-based penalties remain among the most widely used forms of regularization due to their simplicity and effectiveness. The ℓ_1 penalty encourages sparsity by driving certain weights to zero, yielding interpretable and compact models. The ℓ_2 penalty (weight decay) shrinks weights toward zero without inducing sparsity, helping to reduce variance and stabilize training. The Elastic Net combines these two effects [9], recovering groups of correlated variables more effectively than either method alone.

Recent work has explored adaptive extensions to these classical penalties. CORR-Reg [24] modulates regularization strength using neuron activation correlations, penalizing less informative weights more aggressively while preserving critical connections. Evidence-Based Regularization (EBR) [25] applies stronger penalties to neurons lacking sufficient data sup-

port, improving robustness in noisy or low-signal settings. FS-EB [30] extends this idea to function space, introducing structured priors over model outputs to guide learning.

Table 3: Summary of major regularization techniques in machine learning and deep learning.

Regularization	**Methodology**	**Limitations**
ℓ_1 Regularization (Lasso) [7]	Adds the absolute values of model weights to the loss function, promoting sparsity and feature selection.	May discard correlated features and cause instability at high penalty values.
ℓ_2 Regularization (Ridge) [8]	Penalizes squared weight magnitudes to reduce variance and prevent overfitting.	Uniform shrinkage can over-smooth parameters and slow convergence.
Elastic Net [9]	Combines ℓ_1 and ℓ_2 penalties to balance sparsity and smoothness.	Requires tuning two hyperparameters; trade-off sensitive to dataset scale.
CORR-Reg [24]	Adjusts penalty strength using neuron activation correlations to suppress redundant connections.	Relies on accurate activation statistics; less effective for small or noisy datasets.
Evidence-Based Regularization (EBR) / FS-EB [25, 30]	Applies stronger penalties to weights or functions with low data evidence, improving robustness.	Computationally expensive; depends on Bayesian or evidence estimation.
AdamW [31]	Decouples weight decay from gradient updates, preserving optimization dynamics.	Targets optimization stability rather than explicit model regularization.
Hessian Trace Regularization (HTR) [26]	Penalizes the trace of the Hessian to avoid sharp minima and encourage flat solutions.	Requires second-order derivative computation; computationally heavy for large models.

From an optimization perspective, Loshchilov and Hutter [31] proposed AdamW, which decouples weight decay from gradient-based updates to ensure consistent generalization benefits. Curvature-based methods such as Hessian Trace Regularization (HTR) [26] penalize

sharp minima by targeting the trace of the Hessian, leading to flatter and more generalizable solutions.

The principal regularization techniques reviewed in this section are summarized in Table 3. Each entry outlines the regularization strategy, its methodological foundation, and the key limitations observed in prior studies. Together, these approaches illustrate the evolution from classical norm-based penalties to adaptive and curvature-aware methods that enhance model generalization in modern machine learning frameworks.

2.4.2 Positioning of the Proposed Method

The regularization strategy proposed in this thesis is most closely related to norm-based approaches, but introduces a fundamentally different mathematical mechanism. Instead of directly penalizing parameter norms or blending shrinkage effects, the method:

1. Computes the mean of the absolute values of model weights.
2. Extracts the most significant digits of this mean value and maps them to an integer.
3. Evaluates the Collatz stopping time of the resulting integer.

This stopping time is normalized, scaled by a regularization strength, and added to the primary loss function. Unlike ℓ_1, ℓ_2, and Elastic Net—which apply penalties directly to weight magnitudes or their combinations—the Collatz-based approach introduces a structured, deterministic perturbation derived from number-theoretic dynamics. The design is deterministic, architecture-agnostic, and provides stability by avoiding sign cancellation through the use of absolute weight values.

Positioned in this way, the method bridges classical norm-based regularization and novel mathematically motivated strategies, enabling controlled comparisons against established baselines (ℓ_1, ℓ_2, Elastic Net) while demonstrating the feasibility of embedding discrete mathematical dynamics into optimization.

2.5 Related Work

This section reviews computational approaches for verifying the Collatz conjecture at large scales, as well as selected regularization strategies in machine learning that are relevant to the methodological and experimental contributions of this thesis. The computational methods aim to optimize traditional brute-force verification through logical encoding, hardware-efficient operations, or structural exploitation, while the regularization strategies provide the theoretical backdrop for incorporating Collatz-derived dynamics into machine learning.

The first computational approach, outlined in the paper *Collatz conjecture for* $2^{100,000} - 1$ *is true — Algorithms for verifying extremely large numbers* [2], replaces traditional arithmetic operations with logical and bitwise operations while encoding the Collatz sequence into compressed representations known as "code words." We refer to this method as the *Codeword Encoding Approach*.

The second, presented in the paper *Verification of Collatz Conjecture: An Algorithmic Approach* [3], enhances computational efficiency by substituting standard arithmetic operations with faster bitwise operations without altering the sequence's structure. We refer to this method as the *Bitwise Approach* in the following discussions.

2.5.1 Algorithmic Works

2.5.1.1 Codeword Encoding Approach

The Codeword Encoding Approach, proposed in [2], introduces logical computations and sequence encoding strategies to optimize the verification of the Collatz conjecture for extremely large input ranges. The key features of this method include:

- Utilization of an efficient encoding scheme for the Collatz sequence, referred to as "code words," to reduce computational and memory overhead.
- Replacement of traditional arithmetic operations with logical and bitwise operations to achieve faster computation.

- Scalability to handle extremely large numbers while maintaining reduced memory requirements.

In this approach, standard arithmetic computations in the $3x + 1$ operation are replaced by equivalent logical and bitwise operations. Instead of storing the entire Collatz sequence explicitly, the sequence is compressed into a symbolic representation known as a "code word."

Each code word groups together an odd number and its subsequent transformations: the $3x+1$ operation and any immediate divisions by two that follow. This compression significantly reduces the space required to represent the sequence. The algorithm tracks two key metrics during the encoding process:

- U: the number of $3x + 1$ computations performed.
- $D-U$: the number of additional division operations not immediately following a $3x+1$ step.

Overall, the design aims to minimize both computational time and memory usage, although the resulting time complexity remains proportional to the original sequence length.

Algorithm 1 presents a formalized pseudocode that summarizes the main ideas behind the Codeword Encoding Approach proposed in [2]. While the original paper describes the encoding process and supporting algorithms in multiple steps, the pseudocode here consolidates those concepts into a unified algorithm for clarity and consistency.

The algorithm takes an initial integer n as input and produces a compressed symbolic representation of its Collatz sequence, referred to as the "code word." Along with the code word, it computes two key quantities: U, the number of $3x + 1$ operations, and $D - U$, the number of additional division operations by two not immediately following a $3x+1$ step. To illustrate how the code word, U, and $D - U$ values are determined, we present two examples directly adapted from [2]. These examples demonstrate the behavior of the Codeword Encoding Approach as captured by Algorithm 1.

Algorithm 1: Collatz Codeword Encoding Algorithm

Input: n: Starting integer

Output: Code word representing the encoded Collatz sequence, and metrics U & D

Initialize $U \leftarrow 0$, $D \leftarrow 0$, and an empty code word;

while $n \neq 1$ **do**

  if n *is odd* **then**

    Append a '-' symbol to the code word;

    $n \leftarrow 3n + 1$;

    $U \leftarrow U + 1$;

  else

    Initialize $count\_divisions \leftarrow 0$;

    while n *is even* **do**

      $n \leftarrow n/2$;

      $D \leftarrow D + 1$;

      $count\_divisions \leftarrow count\_divisions + 1$;

    if $count\_divisions > 1$ **then**

      Append $(count\_divisions - 1)$ number of '0' symbols to the code word;

return *Code word,* U, $D - U$

Examples from the Literature: The following two examples, adapted directly from [2], illustrate how the Collatz sequence can be encoded using code words and how the values U and $D - U$ are computed. These examples demonstrate the core behavior of the Codeword Encoding Approach. The code words and associated parameters shown here are consistent with the outputs that would be produced by Algorithm 1.

Example 1: For $x = 7$, the full Collatz sequence is:

$$7 \to 22 \to 11 \to 34 \to 17 \to 52 \to 26 \to 13 \to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$$

The total sequence length is $L_{\text{seq}} = 17$. The number of $3x + 1$ operations is $U = 5$, and the

number of additional division steps is $D - U = 6$. These are encoded into the code word:

```
"- - - 0 - 00 - 000"
```

This code word has length 11, which aligns with the relationship:

$$L_{\text{code}} = L_{\text{seq}} - N_{\text{odd}}, \tag{2.1}$$

where $N_{\text{odd}} = 6$.

Example 2: For $x = 63$, the Collatz sequence,

$$63 \to 190 \to 95 \to 286 \to 143 \to \cdots \to 1,$$

has a total sequence length of $L_{\text{seq}} = 108$. The algorithm tracks $U = 39$ $3x + 1$ operations and $D - U = 29$ additional divisions. The resulting code word is:

```
"- - - - - - 000 - - 0 - - - 0 - - - - 0 - 00 - - - 0 - - 0 - - - - - - - 00 -
- - - 000 - 0 - 0 - 000 - 00 - - - 0000 - 000"
```

The total code word length is 68. Since $N_{\text{odd}} = 40$, $L_{\text{code}} = 108 - 40 = 68$, again matching the formula in Equation 2.1.

Shortcomings of the Approach: A critical shortcoming of the algorithm is that the total number of iterations $(U + D + 1)$ is equal to the sequence length. For $x = 7$, the sequence length is 17, and the total number of iterations $U+D+1 = 5+11+1 = 17$. Similarly, for $x = 63$, the sequence length is 108, and the total number of iterations $U+D+1 = 39+68+1 = 108$. This shows that the time complexity of the algorithm remains proportional to the sequence length:

$$O(L_{\text{seq}})$$

where L_{seq} is the length/stopping time of the sequence.

An additional limitation arises from the need to store the generated code words in secondary memory for verification purposes. This significantly increases the space complexity compared

to simply storing the sequence length of a given number. For instance, storing the sequence length of 7 as a single number (17) is far more efficient than storing 11 symbols in the form of a "code word." The "code word" length given in Equation 2.1 shows that the space complexity of the algorithm is:

$$O(L_{\text{seq}} - N_{\text{odd}})$$

where L_{seq} is the length/stopping time of the sequence, and N_{odd} is the total Number of Odd Numbers in the sequence.

In summary, while the algorithm provides a novel perspective through logical operations and sequence encoding, the lack of improvement in time complexity and the higher space requirements for code words make it less optimal.

2.5.1.2 Bitwise Approach

The Bitwise Approach, proposed in [3], optimizes the brute-force computation of the Collatz sequence by replacing standard arithmetic operations with low-level bitwise operations. Specifically, division by two is implemented using a right bit-shift ($n \gg 1$), and the transformation $3n + 1$ is computed using the expression $(n \ll 1) + n + 1$, where $\ll$ denotes a left bit-shift. This substitution improves performance by leveraging hardware-level efficiency while preserving the semantics of the original Collatz transformation.

The pseudocode representation of this method is provided in Algorithm 2. It iteratively processes each term in the sequence without storing intermediate values, incrementing a counter to record the total number of steps until the sequence reaches one. This approach maintains a time complexity of $O(L_{\text{seq}})$, where L_{seq} is the length of the Collatz sequence, as each step is processed exactly once. It also achieves constant space complexity $O(1)$ by using only scalar variables to track the input and the sequence length. Unlike the Codeword Encoding Approach [2], which introduces memory overhead by storing symbolic representations of the sequence, the Bitwise Approach avoids all intermediate storage and is therefore

more memory-efficient.

Algorithm 2: Bitwise Collatz Sequence Length Algorithm

Input: n: Starting integer

Output: *length*: Total number of steps to reach 1

Initialize $length \leftarrow 1$;

while $n \neq 1$ **do**

  if n *is even* **then**

    $n \leftarrow n \gg 1$; `// Bitwise division by 2`

  else

    $n \leftarrow (n \ll 1) + n + 1$; `// Bitwise computation of` $3n + 1$

  $length \leftarrow length + 1$;

return *length*;

To provide a broader perspective, we now introduce additional relevant methods and summarize all approaches using a comparative table.

2.5.1.3 Additional Key Approaches

Beyond these two core computational techniques, several other studies have provided important insights that inform both the structural interpretation and algorithmic design adopted in this thesis.

Bottom-Up Tree Construction [4] builds the inverse Collatz tree starting from $n = 1$ using the transformations $(n-1)/3$ and $2n$ under valid divisibility constraints. This reverse, layer-wise construction exposes recurring patterns in convergence paths and highlights the shared ancestry of large subsets of natural numbers. Although primarily theoretical, it provides valuable intuition for the inverse traversal strategy proposed in this study and conceptually supports the pruning of redundant even paths validated later through machine learning experiments.

Convergence Stairs Framework [6] models the Collatz process as a concurrent program and introduces verified algorithms that generate all numbers belonging to a given stair; sets of integers with identical stopping times. This work demonstrates that stopping times form clustered distributions rather than random ones, providing computational evidence for the structural regularities explored in this thesis. By linking formal specification with algorithmic generation, the study bridges theoretical modeling and empirical validation, reinforcing the clustering behavior that underpins the data-driven component of this research.

The Proposed Work introduces a structure-aware traversal algorithm that integrates bit-level operations with pruning of redundant even paths in the inverse Collatz tree. This design achieves an average 28% reduction in iteration count compared to standard traversal across one billion inputs. Furthermore, the approach extends beyond numeric verification by formulating a novel regularization term for machine learning models based on the Collatz stopping time of derived weight statistics, thereby unifying algorithmic efficiency with structural learning.

2.5.1.4 Comparative Summary of Related Algorithmic Works

The comparative summary of benchmark algorithms is presented in Table 4. These approaches were selected because they directly align with the objectives of the proposed structure-aware algorithm. Although some of the studies are relatively old, they remain the only available computational methods that explicitly compute Collatz stopping times for natural numbers.

Most recent works on the conjecture are primarily theoretical, focusing on structural or probabilistic properties rather than implementable computation. Accordingly, the selected benchmark algorithms represent the core empirical efforts addressing the stopping-time problem, while key theoretical studies are also included for comparison since they provide structural insights that informed the design of the proposed algorithm.

Table 4: Comparative summary of major Collatz verification and analysis algorithms.

Study	Redundancy Reduction	Computation Speed	Scalability	Structural Insight	ML-Guided
Codeword Encoding [2]	˜	˜	˜	˜	✗
Bitwise Approach [3]	✗	˜	˜	✗	✗
Bottom-Up Tree Construction [4]	✓	✗	✗	✓	✗
Convergence Stairs [6]	✓	˜	˜	✓	✗
Other Structural Analyses [14, 11]	˜	✗	✗	✓	✗
Proposed (This Work)	✓	✓	✓	✓	✓

Legend: ✓ = present, ✗ = absent, ˜ = partial.

2.5.2 Related regularization techniques

2.5.2.1 Norm-Based Penalties

Among the many existing regularization strategies, the methods most closely related to this study are the classical norm-based penalties: ℓ_1 [7], ℓ_2 [8], and Elastic Net [9]. These techniques remain the foundation of parameter-level regularization, where model weights themselves determine the penalty applied during training. They are therefore referred to here as *parameter-informed* penalties, as the degree of regularization is explicitly governed by the magnitudes of the learnable parameters.

The ℓ_1 penalty encourages sparsity in model weights by adding the sum of their absolute

values to the task loss:

$$\mathcal{L}_{\ell_1}(\mathbf{w}) = \mathcal{L}_{\text{task}}(\mathbf{w}) + \lambda \sum_i |w_i| \,,$$

where $\mathcal{L}_{\text{task}}$ denotes the primary loss (e.g., cross-entropy or mean-squared error), $\mathbf{w}$ represents the trainable parameters, and λ controls the regularization strength. This formulation drives some weights exactly to zero, producing sparse and interpretable models.

The ℓ_2 penalty, or weight decay, penalizes the squared magnitudes of the weights:

$$\mathcal{L}_{\ell_2}(\mathbf{w}) = \mathcal{L}_{\text{task}}(\mathbf{w}) + \lambda \sum_i w_i^2.$$

Unlike ℓ_1, it uniformly shrinks all parameters toward zero, reducing variance and improving numerical stability. The Elastic Net combines the two by introducing both ℓ_1 and ℓ_2 terms, balancing sparsity and smoothness and performing well when input features are correlated.

The Collatz-based regularizer proposed in this work belongs to the same general family of parameter-informed penalties but introduces a fundamentally different mathematical formulation. Rather than directly summing or squaring parameter magnitudes, it computes the mean absolute weight value (excluding biases), maps its significant digits to an integer, and evaluates the Collatz stopping time of that integer. After computing the Collatz stopping time, the result is normalized to ensure a bounded penalty, then scaled by the regularization strength and added to the task loss. This formulation yields a deterministic and numerically stable penalty that remains within a controlled range, preventing excessive shrinkage even when model weights are large. In contrast, conventional norm-based penalties grow unbounded with parameter magnitude, which can lead to over-penalization and degraded learning under strong regularization. By embedding discrete, number-theoretic dynamics within the optimization process, the Collatz regularizer provides a structured and self-limiting alternative that maintains stability across varying penalty strengths.

2.5.2.2 Comparative Summary of Regularization Methods

The comparative summary of benchmark regularization techniques is presented in Table 5. These methods were selected because they represent the core parameter-informed regular-

ization strategies against which the proposed Collatz-based approach can be meaningfully compared. The comparison emphasizes four key aspects: whether the method is parameter-informed, requires only a single hyperparameter, maintains stability under high regularization strengths, and enforces a bounded penalty. This perspective highlights the unique property of the Collatz-based regularizer, which combines deterministic, parameter-informed behavior with bounded and stable effects across different penalty regimes.

Table 5: Comparative summary of regularization methods (norm-based baselines vs. proposed).

Method	**Parameter-informed**	**Single Hyperparameter**	**Stability at High Penalty**	**Bounded Penalty**
ℓ_1 (Lasso) [7]	✓	✓	X	X
ℓ_2 (Ridge) [8]	✓	✓	˜	X
Elastic Net [9]	✓	X	X	X
Collatz-based (This Work)	✓	✓	✓	✓

Legend: ✓ = present, X = absent, ˜ = partial/conditional.

2.5.2.3 Summary of Identified Gaps

The review of prior studies reveals distinct gaps across theoretical, computational, and regularization domains relevant to this research. From the theoretical investigations summarized in Table 1, most works remain confined to probabilistic or structural analyses without achieving a complete theoretical resolution of the Collatz conjecture. While these studies provide valuable conceptual insights, they do not yield practical methods for efficiently computing stopping times or verifying large input ranges.

As shown in the computational comparison (Table 4), existing algorithms rely heavily on brute-force enumeration and direct iteration. Although some introduce binary or automata-based optimizations, their time complexity still scales linearly with the sequence length,

limiting their applicability to very large numbers. Furthermore, none of these algorithms leverage the underlying structural regularities of the Collatz tree or employ machine-learning-guided pattern analysis to inform algorithmic design. This absence of scalable, structure-aware computation forms the core motivation for the present study.

Finally, the regularization strategies reviewed in Table 5 highlight another gap in how penalties are applied within learning systems. Classical norm-based methods such as ℓ_1, ℓ_2, and Elastic Net impose unbounded penalties that may excessively constrain model parameters, especially under high regularization strengths. This motivates the development of a bounded, deterministic penalty formulation, which is realized here through the Collatz-based regularizer, maintaining stability while preserving parameter interpretability.

Table 6: Summary of key gaps identified from the reviewed literature and their relation to this study.

Domain	**Identified Gaps in Prior Works**	**Direction in This Study**
Theoretical Studies	• No complete analytical resolution of the Collatz conjecture • Structural models are mainly conceptual; limited computational translation	• Use inverse-tree and convergence insights to inform structure-aware traversal
Computational Approaches	• Heavy reliance on brute-force iteration with time scaling as $O(L_{\text{seq}})$ • No ML-guided stopping-time pattern analysis to inform algorithmic design • Limited redundancy elimination; structural regularities underused	• Use ML explorations to surface patterns that guide deterministic heuristics • Develop a scalable, structure-aware algorithm that prunes redundant even runs
Regularization Techniques	• Unbounded norm penalties can over-constrain models under high λ • Lack of deterministic, bounded penalties grounded in mathematical structure	• Introduce a normalized, bounded Collatz-based penalty to maintain stability while remaining parameter-informed

To clearly summarize these gaps and their direct connection to the objectives of this study, Table 6 provides a concise synthesis across the three domains.

Building on these identified gaps, Chapter 3 details the structure-aware algorithm design and the ML-guided exploratory analyses that together address scalability in stopping-time computation and provide a bounded, stable regularization framework.

Chapter 3 Methodology

This chapter presents the methodology used to design, analyze, and validate a novel algorithm for computing Collatz stopping times. It combines exploratory analysis, structural investigation, algorithmic development, and comparative evaluation to build a rigorous foundation for the research.

We begin with a series of preliminary experiments that examine patterns in Collatz stopping times using visual analytics and basic machine learning models. These initial insights reveal non-random, repetitive behavior in the sequence, motivating a deeper structural analysis.

We then investigate the hierarchical organization of the Collatz tree, identifying key branching patterns and structural regularities. This analysis informs the development of a structure-aware algorithm that minimizes redundant computation by leveraging these patterns.

Subsequent sections present the algorithm's design, step-by-step operation, and a theoretical analysis of its time and space complexity.

The experiments conducted in this work utilized two different hardware configurations. The preliminary experiments described in Section 3.1, including the exploratory data analysis and machine learning-based pattern discovery, were conducted on Google Colab using Python 3.11.11 and a cloud-based runtime environment. All subsequent experiments in Sections 3.4, 4.1, 4.2, and 4.3 were performed locally on a MacBook Air equipped with an Apple M2 chip, 8GB of unified memory, and running macOS Sequoia Version 15.2, using Python 3.12.4 and Julia 1.11.2. Additionally, the large-scale execution time evaluation described in Section 4.2.2 was conducted on Google Colab utilizing an NVIDIA T4 GPU to facilitate efficient computation.

Figure 1 presents an overarching view of the methodology adopted in this thesis. The process begins with exploratory data analysis to uncover statistical and visual patterns in Collatz stopping times, followed by preliminary machine learning experiments to assess the

predictability and structure of the sequence. These insights inform a structural investigation of the Collatz tree, which serves as the foundation for a novel structure-aware algorithm aimed at reducing redundant computations.

The algorithm is then rigorously analyzed both theoretically and empirically, with large-scale experiments validating its efficiency, correctness, and scalability compared to existing methods. In addition to these computational evaluations, the stopping time computation framework is later adapted in the experimentation phase as a novel regularization term for neural network training. This Collatz-based penalty is compared against standard norm-based baselines (ℓ_1, ℓ_2, and Elastic Net) to assess its effectiveness across architectures and tasks. Unless otherwise noted, deep learning experiments are trained for 20 epochs at two regularization strengths ($\lambda \in \{0.01, 0.10\}$) and averaged over 10 independent runs for statistical robustness (details in Section 4.4).

3.1 Preliminary Experiments: Pattern Discovery via Regression and Clustering

We conducted preliminary experiments to gain insight into the relationship between natural numbers and their stopping times (i.e., the lengths of their Collatz sequences). These experiments began with exploratory data analysis to visually assess any underlying patterns or regularities.

Visual Analysis: Figure 2 presents a set of visualizations that reveal structural characteristics in the stopping times of natural numbers. Subfigure 2a shows the stopping times for natural numbers from 1 to 5000, where two key observations stand out. First, the red diamond markers identify selected large values of n, highlighting that their stopping times increase slowly and appear to follow a logarithmic growth pattern. Second, the green points identify the most frequent stopping time observed in this range (stopping time = 39), which is shared by many input values. This illustrates that stopping times are not unique and that several numbers converge to the same value, reinforcing the idea of structured repetition.

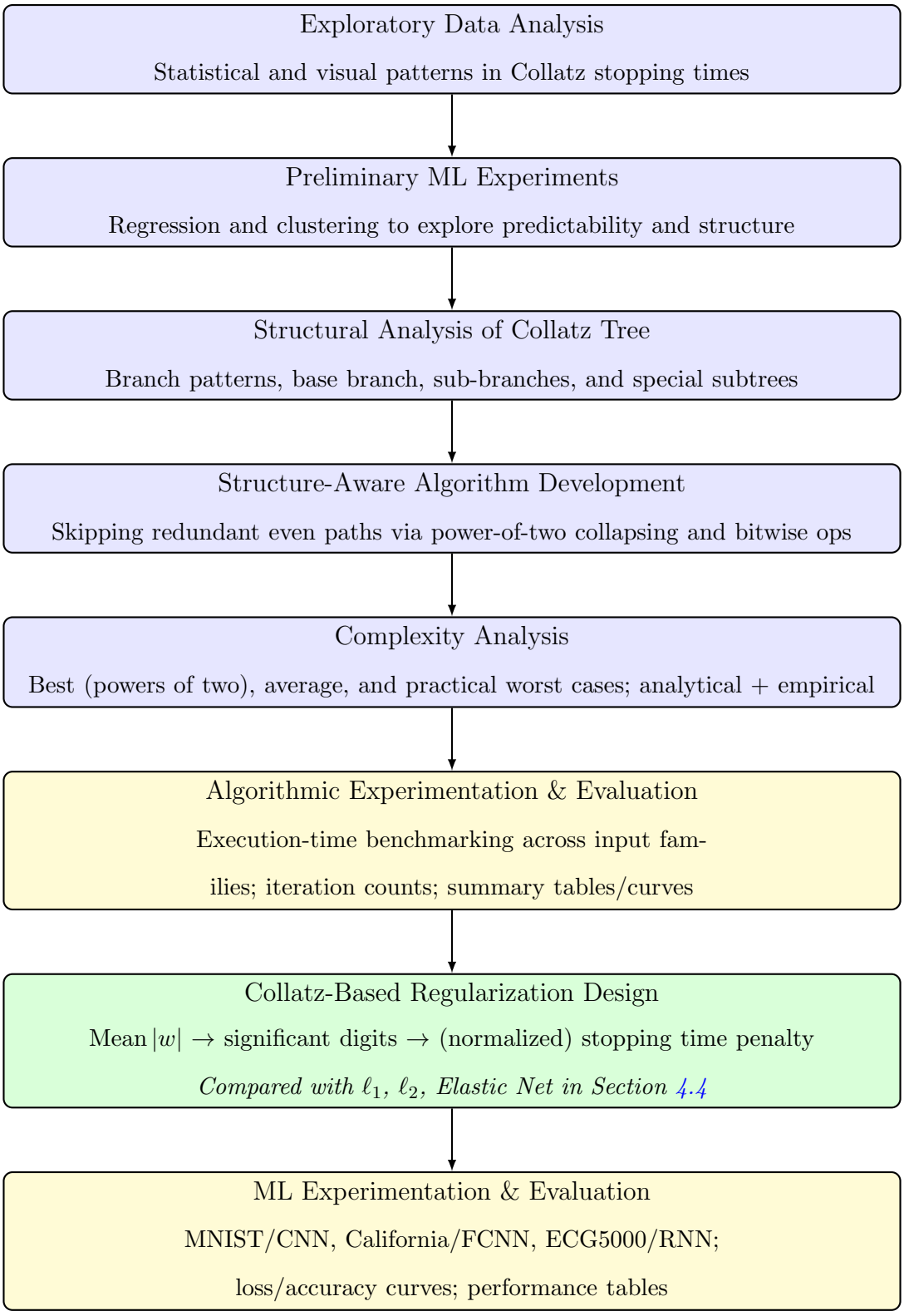
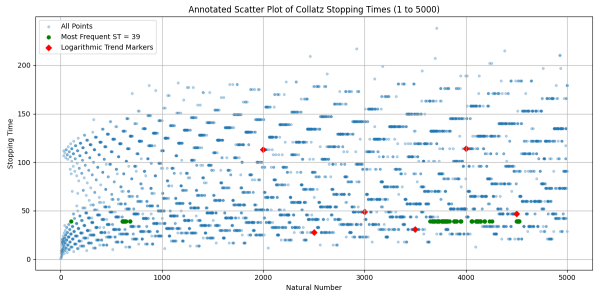
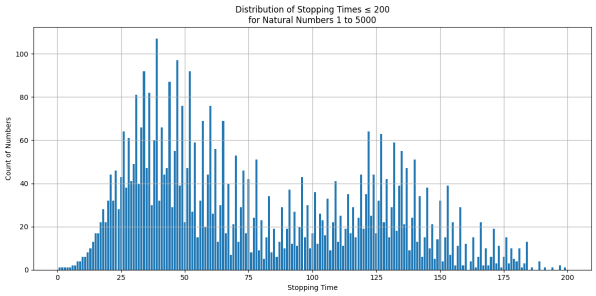


Figure 1: Methodology pipeline: from exploratory analysis and structural insights to algorithm design, complexity analysis, and two-stage experimentation (algorithmic and ML).

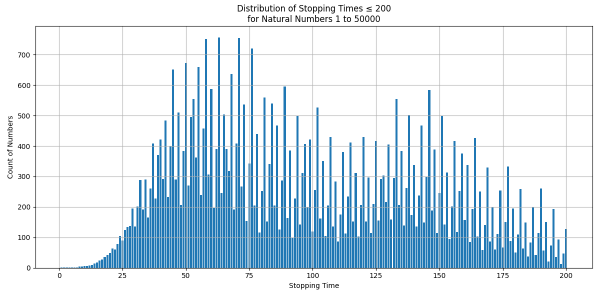
To further investigate how stopping time distributions behave over larger numeric ranges, we analyzed the frequency of stopping times less than or equal to 200, as shown in Subfigures 2b, 2c, and 2d. These subfigures correspond to the input ranges $n \in [1, 5{,}000]$, $[1, 50{,}000]$, and $[1, 500{,}000]$, respectively. As the input range increases, the distribution of stopping times not only spreads to include higher values but also becomes more densely populated and stable across that range. Nevertheless, certain stopping times remain disproportionately frequent, suggesting a non-uniform yet highly structured distribution.



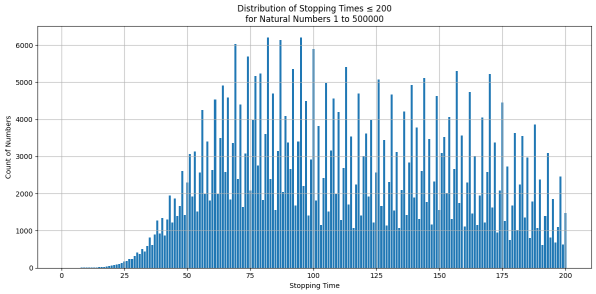
(a) Annotated scatter plot for $n \in [1, 5000]$.



(b) Distribution for $n \in [1, 5{,}000]$.



(c) Distribution for $n \in [1, 50{,}000]$.



(d) Distribution for $n \in [1, 500{,}000]$.

Figure 2: Exploratory analysis of Collatz stopping times: (a) structural repetition and logarithmic growth; (b–d) distributions across increasing input ranges (stopping times $\leq$ 200).

Regression and Clustering Experiments: Following the exploratory data analysis, the next step in our investigation involved applying machine learning models to gain deeper insights into the relationship between natural numbers and their stopping times. While

classification algorithms might seem natural due to the discrete nature of stopping times, the exploratory analysis revealed that there is no finite bound on the range of possible stopping times, even within relatively small input ranges. Consequently, modeling the problem as a standard classification task is infeasible. Instead, we focused on regression and clustering approaches.

The regression and clustering experiments were implemented using `scikit-learn` for traditional models and `PyTorch` for neural network training. For regression, three models were evaluated: a standard linear regressor from `scikit-learn`, a decision tree regressor with a fixed depth of 4, and a feedforward neural network (FCNN) implemented in PyTorch. The FCNN consisted of two hidden layers with 64 units each, used ReLU activations, and was trained for 100 epochs using the Adam optimizer.

For clustering, DBSCAN was tested with $\epsilon = 0.6$ and `min_samples` $= 7$, while agglomerative clustering was applied with a distance threshold of 1.5 and no fixed number of clusters. All clustering inputs were standardized using `StandardScaler` to ensure effective distance-based separation. The regression models were evaluated using root mean square error (RMSE), while clustering performance was measured using silhouette scores.

The regression models aimed to predict the stopping time directly from the input number, while the clustering algorithms sought to identify underlying structural groupings in the number–stopping time space. Table 7 summarizes the performance of the models applied. Linear regression achieved an RMSE of 44.38, closely followed by a decision tree regressor with a maximum depth of 4 (RMSE = 44.48). A feedforward neural network (FCNN) performed slightly worse, with an RMSE of 53.36, suggesting that greater model complexity does not necessarily yield better predictive performance for this task.

On the clustering side, DBSCAN failed to find meaningful groupings under the tested parameters, producing a silhouette score of -1.0, indicative of a single cluster or poor separation. In contrast, agglomerative clustering achieved a silhouette score of 0.50, confirming the earlier

observation that stopping times are organized in a structured, hierarchical manner. These preliminary machine learning experiments reinforce the idea that stopping times are not randomly distributed but follow patterns that simple models and hierarchical clustering can partially capture.

Table 7: Regression and clustering analysis of Collatz stopping times evaluated using RMSE and silhouette score.

Algorithm	**Type**	**Metric**	**Value**
Linear Regression	Regression	RMSE	44.38
FCNN (Regression)	Regression	RMSE	53.36
Decision Tree Regressor (depth=4)	Regression	RMSE	44.48
DBSCAN (eps=0.6, min_samples=7)	Clustering	Silhouette Score	-1.00
Agglomerative Clustering	Clustering	Silhouette Score	0.50

The results of our preliminary experiments, combined with observations from previous studies such as [4], strongly suggest that the stopping times of natural numbers are governed by hierarchical and structured patterns rather than purely random behavior. In particular, the successful application of simple regression models, the identification of repeated stopping times, and the effective clustering achieved through agglomerative methods reinforce the view that the Collatz sequence evolution exhibits an inherent tree-like organization.

Motivated by these findings, we adopt a bottom-up approach that constructs the Collatz sequences in reverse, starting from $n = 1$ and applying inverse transformations to map integers back to the origin. This perspective not only aligns with the structural insights observed experimentally but also leverages the convergence properties highlighted in prior theoretical analyses. The next section presents a detailed examination of the Collatz tree,

outlining its key branches, branching rules, and special substructures, which together form the foundation for the algorithmic strategies developed later in this work.

3.2 The Collatz Tree and Observed Patterns

The Collatz tree is a hierarchical structure constructed by applying inverse transformations to the Collatz function defined in Equation (1.1), starting from the base node $n = 1$. The tree is formed by recursively identifying all valid predecessors of a number under the Collatz function. There are two such inverse operations: the inverse of the division by 2 step is multiplication by 2, which is valid for all n; the inverse of the $3n + 1$ step is given by $(n-1)/3$, which is valid when n is even and $n-1$ is divisible by 3. These inverse operations are formally expressed as:

$$C^{-1}(n) = \begin{cases} 2n, & \text{for all } n, \\ \dfrac{n-1}{3}, & \text{if } n \text{ is even and } 3 \mid (n-1). \end{cases} \tag{3.1}$$

By iteratively applying these inverse rules, the tree grows upward from 1, systematically exploring all possible predecessors that could lead to a given value under the Collatz sequence. As shown in Fig. 3, the resulting structure reveals key branching patterns and recurrent substructures. These properties serve as the foundation for designing efficient algorithms that reduce redundant computation by selectively navigating and pruning the tree.

Base Branch: The base branch starts at 1 and extends to infinity, following the pattern of powers of 2:

$$1 \to 2 \to 4 \to 8 \to 16 \to 32 \to 64 \to \ldots \to 2^n \ldots$$

Sub-Branches: All branches other than the base branch (sub-branches) also start with an odd number and extend to infinity as multiples of powers of 2.

$$2m + 1 \to 2(2m + 1) \to \ldots \to (2m + 1) \cdot 2^n \to \ldots$$

where $m \geq 1, n \geq 0$.

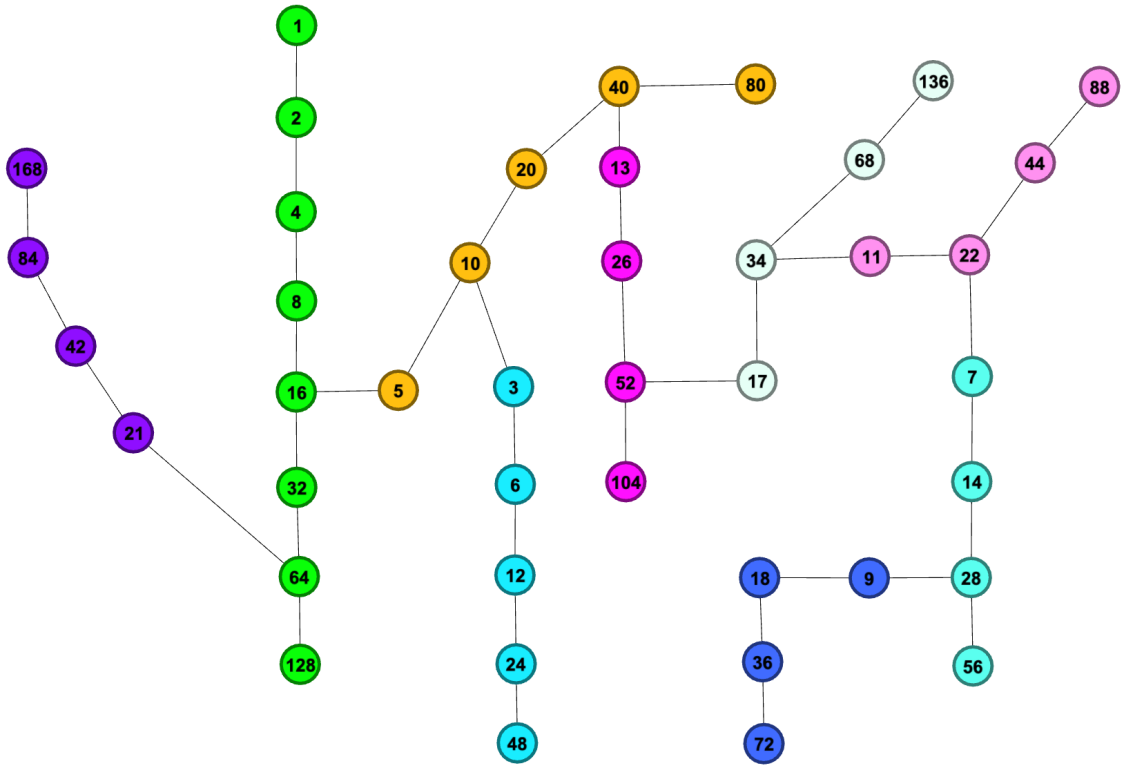


Figure 3: The Collatz tree with the base path (2^n) and sub-branches starting at $6k + 10$. Each sub-branch follows the pattern $m \cdot 2^n$, where m is an odd number.

Branch Points: Branches occur throughout the Collatz tree at specific positions defined by $6k + 10$ $(k \geq 0)$.

Special subtrees: Odd multiples of 3 $(3, 9, 15, 21, \ldots)$ form special subtrees which do not branch further.

$$3(2m + 1) \to 6(2m + 1) \to \ldots \to 3(2m + 1) \cdot 2^n \to \ldots$$

where $m, n \geq 0$.

The structural observations derived from the Collatz tree provide critical insights for algorithm design. By recognizing patterns such as the base branch formed by powers of two, the organization of sub-branches originating from specific odd numbers, and the predictable behavior of branch points and special subtrees, we can significantly optimize the calculation

of stopping times. These hierarchical and repetitive structures reduce the need for redundant computations and enable more efficient traversal strategies. Building on these insights, the next section presents the development of an optimized algorithm that leverages the Collatz tree's properties to compute stopping times more effectively.

3.3 Algorithm Development for Calculating Stopping Time

The proposed algorithm leverages the structural properties of the Collatz tree and observed patterns in its branches to minimize redundant computations.

Handling Odd Numbers: If the current number is odd, it is the root of its branch. To move to the "previous branch" in the Collatz tree, compute the previous number using the equation:

$$\text{previous Number} = 3 \times \text{current Number} + 1 \tag{3.2}$$

This computation gives the even number from which the current branch originated.

Handling Even Numbers: If the current number is even, it belongs to a branch rooted at an odd number. To find the root of the current branch, determine the largest power of 2 that divides the current number with a remainder of zero. This can be computed efficiently using the bitwise operation:

$$\text{largest power of 2} = \text{current number} \wedge \text{-current number} \tag{3.3}$$

where $\wedge$ is a bitwise AND operation.

Dividing the current even number by the largest power of 2 yields the root odd number:

$$\text{odd Root} = \frac{\text{current Number}}{\text{largest power of 2}} \tag{3.4}$$

If the odd root equals 1, the base branch is reached, and the stopping time calculation is complete.

Repeating the Process: Alternate between handling odd and even numbers, reducing the current number until $n = 1$. This iterative process ensures efficient computation of the stopping time.

Algorithm Overview: Algorithm 3 outlines the proposed method for calculating the stopping time of a given number. The algorithm applies the standard Collatz transformations, with an optimization when the number is even by identifying and dividing by the largest possible power of two in a single step. This strategy reduces the number of iterations needed, thereby improving computational efficiency without altering the correctness of the stopping time calculation.

Algorithm 3: CalculateStoppingTime

Input: n: Integer for which the stopping time is calculated

Output: *stopping_time*: Total steps to reach 1

```
stopping_time ← 0;
while n ≠ 1 do
    if n is odd then
        n ← 3 · n + 1;
        stopping_time ← stopping_time + 1;
    else
        largest_power ← n ∧ −n;
        exponent ← log2(largest_power);
        stopping_time ← stopping_time + exponent;
        n ← n/largest_power;
return stopping_time + 1;
```

Example Walkthrough: We will compute the stopping time for $n = 20{,}480$ using algorithm 3 step-by-step.

Initial Setup:

$$\begin{aligned} &\text{Input:} && n = 20{,}480 \\ &\text{Initialize:} && \text{stopping\_time} = 0 \end{aligned}$$

Iteration 1: $n = 20{,}480$ (even):

$$\begin{aligned} &\text{largest\_power} = 20{,}480 \wedge -20{,}480 = 4{,}096 \\ &\text{exponent} = \log_2(4{,}096) = 12 \\ &\text{stopping\_time} = 0 + 12 = 12 \\ &n = \tfrac{20{,}480}{4{,}096} = 5 \end{aligned}$$

Iteration 2: $n = 5$ (odd):

$$\begin{aligned} &n = 3 \times 5 + 1 = 16 \\ &\text{stopping\_time} = 12 + 1 = 13 \end{aligned}$$

Iteration 3: $n = 16$ (even):

$$\begin{aligned} &\text{largest\_power} = 16 \wedge -16 = 16 \\ &\text{exponent} = \log_2(16) = 4 \\ &\text{stopping\_time} = 13 + 4 = 17 \\ &n = \tfrac{16}{16} = 1 \end{aligned}$$

Termination: $n = 1$

$$\text{stopping\_time} = 17 + 1 = 18$$

Hence, the stopping time for $n = 20{,}480$ is 18. While the state-of-the-art algorithms compute this result in 18 iterations, the proposed algorithm achieves the same result using only 3 iterations by directly leveraging structural patterns in the Collatz tree. This optimization significantly reduces computational redundancy and highlights the algorithm's efficiency.

Beyond its application to numeric verification, the stopping time computation implemented in this algorithm also serves as the core component of the Collatz-based regularization term described later in this thesis. In the deep learning setting, this procedure is applied to model weights during training to generate a deterministic, number-theoretic penalty that augments the primary loss function.

3.4 Analysis of Proposed Algorithm

To analyze the total iterations performed by the algorithm, consider the number $n = 48$ as shown in Fig. 3. The traversal involves two sub-branches (starting with 3 and 5) and the base branch.

For each sub-branch:

- One iteration reduces the number to the root odd number that starts the sub-branch.
- One iteration moves up the tree to the parent sub-branch.

In this case:

- One iteration is used to reduce 48 to 3
- One iteration moves 3 to the parent sub-branch starting with 5 $(3 \times 3 + 1 = 10)$
- One iteration is used to reduce 10 to 5
- One iteration moves 5 to the base branch $(5 \times 3 + 1 = 16)$
- One iteration is used to reduce 16 to 1

which gives a total of 5 iterations.

In general for k sub-branches:

- k iterations are required to reduce the numbers in each sub-branch to their root odd numbers.

- k iterations are required to traverse up the tree to the parent sub-branches.
- One iteration is required at the base branch to reduce the number to 1.

Therefore, the total number of iterations is:

$$\text{Total Iterations} = 2k + 1 \tag{3.5}$$

where k is the number of sub-branches.

Key Observations:

- Equation 3.5 encapsulates the traversal of both even and odd numbers in the Collatz sequence.
- The value of k depends on the structure of the Collatz tree and the placement of n within it.
- Numbers in the base branch ($k = 0$) represent the best case, while numbers requiring traversal of many sub-branches correspond to the worst case.

Best-Case Time Complexity: The best case occurs when the input number n is a power of 2 ($n = 2^k$). Numbers on the base branch of the Collatz tree (powers of 2) require no branching and are reduced directly to 1 in a single iteration (as verified experimentally in Section 4.2.2 and discussed in the complexity analysis results).

Steps:

1. Compute $\log_2(n)$ to determine the number of trailing zeros, which is performed in $O(1)$ using optimized hardware instructions.
2. The largest power of 2 dividing n is n itself, and $n/n = 1$ terminates the loop.

Stopping Time: The stopping time is:

$$\text{stopping\_time} = \log_2(n) + 1,$$

where $\log_2(n)$ accounts for trailing zeros and $+1$ for the final step.

Confirmation with $2k + 1$: In the best case ($k = 0$), the total iterations is $2k + 1 = 2(0) + 1 = 1$, and this confirms that any number on the base branch reduces to 1 in $O(1)$, without requiring any traversal through sub-branches.

Time Complexity: Each operation ($\log_2(n)$, bitwise AND, division) is performed in $O(1)$, leading to an overall best-case complexity of $\mathbf{O(1)}$.

Worst-Case and Average-Case Scenarios: To derive the worst- and average-case complexities, both the theoretical structure of the Collatz tree and empirical execution results were analyzed in parallel. The hypothetical worst case would occur if the sequence exhibited an equal proportion of even and odd numbers, thereby maximizing the number of branching points (k). However, this configuration does not occur in practice, as confirmed by both analytical reasoning and experimental observation. The following consistent structural patterns were identified:

- Even numbers dominate odd numbers in every branch due to rapid reductions by successive divisions by 2.
- Branching occurs only at specific numbers of the form $6k + 10,\ k \geq 0$.
- Odd multiples of 3 do not initiate new branches.
- Each branch typically undergoes several even reductions before reaching the next sub-branch, indicating that branching is infrequent relative to total sequence length.

Consequently, the number of sub-branches k remains much smaller than the total count of even transitions in any given sequence. Because the exact distribution of odd and even transitions cannot be derived analytically from tree patterns alone, a computational approach was used to estimate the effective growth rate and verify these relationships empirically (see Section 4.2.2). This joint analytical–empirical framework provides the basis for defining the observed average- and worst-case time complexities.

Computational Approach: The behavior of iterations and stopping times was analyzed using the proposed algorithm. The steps undertaken were as follows:

1. The proposed algorithm was used to compute the stopping times and the number of iterations $(2k + 1)$ for a range of inputs.

2. Stopping times and iterations were analyzed over exponentially increasing ranges, such as 10,000, 20,000, up to 5,120,000.

3. For each range, the best, average, and worst iterations were recorded to observe scaling behavior.

Table 8: Best, average, and worst iteration counts for computing Collatz stopping times using the proposed algorithm.

Number of Inputs	**Best Iterations**	**Average Iterations**	**Worst Iterations**
10,000	1	56.90	192
20,000	1	61.36	204
40,000	1	65.68	238
80,000	1	70.27	258
160,000	1	75.00	282
320,000	1	79.78	328
640,000	1	84.59	378
1,280,000	1	89.37	392
2,560,000	1	94.15	416
5,120,000	1	98.93	444

The results shown in Table 8 offer valuable empirical evidence for the efficiency and scalability of the proposed approach, particularly in terms of how stopping times evolve across large

input ranges. The consistent growth patterns observed across best, average, and worst iterations underscore the structured nature of the Collatz process and further justify the algorithmic strategies developed in this section.

Visualization: Figure 4, generated with Matplotlib, provides a visual representation of the iterations across different input sizes. The scatter plots demonstrate how iterations vary with input size, revealing the logarithmic growth of iterations with respect to the input numbers. Each subplot corresponds to a selected input range, offering a detailed view of the scaling behavior.

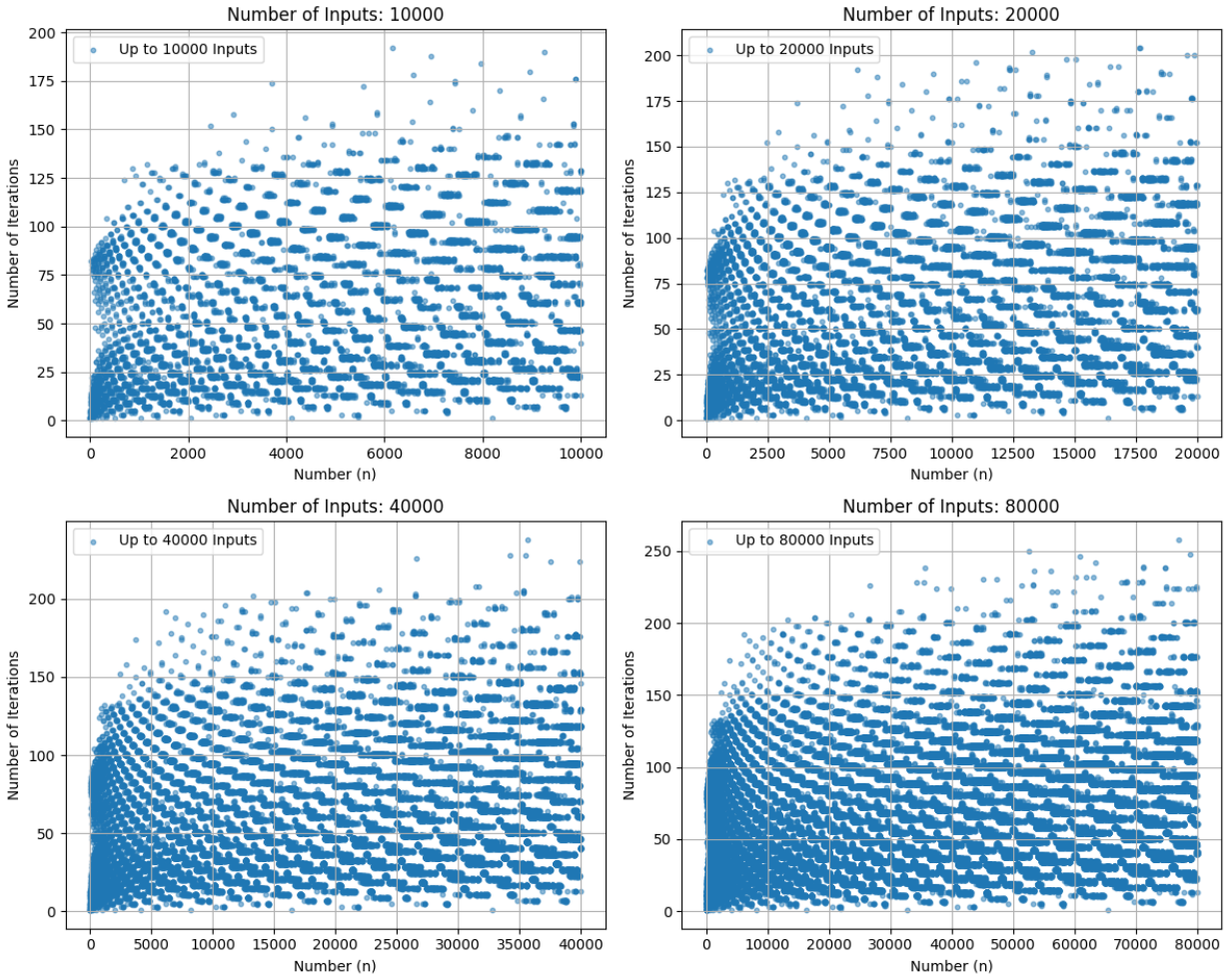


Figure 4: Scatter plots of iteration counts across increasing input ranges.

Average-Case Time Complexity: To derive the average-case complexity, we analyzed the number of iterations for exponentially increasing input sizes. The observed average iteration counts shown in Table 8 for inputs ranging from 10,000 to 5,120,000 exhibited slow, logarithmic growth, consistent with the empirical results reported in the execution time experiments (Section 4.2.2).

Specifically:

- The average number of iterations grew by approximately 5 units as the input size doubled.
- This suggested a logarithmic relationship between the number of inputs and the average number of iterations.

Assuming the average number of iterations follows the form:

$$f(n) = a \cdot \log_2(n) + b,$$

we used two data points to calculate a and b.

Given the data points:

$$f(10,000) = 56.90, \quad \log_2(10,000) \approx 13.2877,$$

$$f(5,120,000) = 98.93, \quad \log_2(5,120,000) \approx 22.3181$$

we can express the relationships as a system of equations:

$$56.90 = a \cdot 13.2877 + b,$$

$$98.93 = a \cdot 22.3181 + b$$

Solving this system of linear equations yields:

$$a \approx 4.65, \quad b \approx -4.91$$

Therefore, the average number of iterations as a function of n is given by:

$$f(n) = 4.65 \cdot \log_2(n) - 4.91 \tag{3.6}$$

Order of Growth Equation 3.6 confirms that the average number of iterations grows logarithmically with the input size. Consequently, the algorithm's average-case complexity is $\mathbf{O}(\log(\mathbf{n}))$, where n is the input size. This result aligns with the dominance of even reductions and the relatively shallow depth of sub-branches observed in the Collatz tree structure.

Worst-Case Time Complexity: The worst-case complexity was analyzed using the proposed algorithm, focusing on the maximum number of iterations $(2k+1)$ observed for exponentially increasing input sizes.

The results, summarized in Table 8, show that the worst-case number of iterations grow slowly with input size, consistent with logarithmic scaling.

For instance:

- For $n = 10{,}000$, the worst-case number of iterations were 192.
- For $n = 5{,}120{,}000$, the worst-case number of iterations were 444.

To derive the worst-case complexity, we fit the observed data into the logarithmic form:

$$f(n) = a \cdot \log_2(n) + b,$$

where a and b are constants.

Using the data points:

$$f(10{,}000) = 192, \quad \log_2(10{,}000) \approx 13.2877,$$
$$f(5{,}120{,}000) = 444, \quad \log_2(5{,}120{,}000) \approx 22.3181$$

we can express the relationships as a system of equations:

$$192 = a \cdot 13.2877 + b,$$
$$444 = a \cdot 22.3181 + b$$

solving this system of linear equations yields:

$$a \approx 28.00, \quad b \approx -180.17$$

The formula for the worst-case number of iterations as a function of n is therefore:

$$f(n) = 28 \cdot \log_2(n) - 180.17 \tag{3.7}$$

Order of Growth: Equation 3.7 confirms that the worst-case complexity scales logarithmically with the input size. Therefore, the algorithm's worst-case complexity is $\mathbf{O}(\log(\mathbf{n}))$, where n is the input size.

Space Complexity: The algorithm's space complexity is constant ($\mathbf{O}(\mathbf{1})$) across all cases due to its iterative nature. It uses a fixed number of scalar variables to store the current value of n, the stopping time counter, and temporary intermediate results (e.g., largest power of 2). No additional data structures are required, and this remains true regardless of the input size or the number of iterations.

The time complexity analysis confirms that both the average-case and worst-case number of iterations for the proposed algorithm grow logarithmically with the input size. This result is consistent with the patterns observed during the preliminary experiments in Section 3.1, where exploratory data analysis revealed a slow, approximately logarithmic increase in stopping times as the natural numbers grew larger. The alignment between experimental observations and theoretical complexity provides further validation for the algorithm's efficiency. Building on this foundation, the next section presents a detailed verification of the proposed algorithm against brute-force methods to ensure its correctness.

3.5 Integration into Neural Network Regularization

The stopping time computation described in this chapter is integrated into a novel regularization term for deep learning models. At each training step, all learnable weight tensors (excluding biases) are flattened and concatenated. The absolute values are taken to avoid sign cancellation, and their mean, denoted $\overline{|w|}$, is computed to capture the global magnitude of parameters.

From this non-negative mean value, the first three significant digits are extracted using $\mathrm{sig}(\overline{|w|})$ and interpreted as an integer n. The Collatz stopping time $C(n)$ of this integer is then calculated and added to the task loss with strength λ:

$$\mathcal{L}_{\text{Collatz}}(\mathbf{w}) \;=\; \mathcal{L}_{\text{task}}(\mathbf{w}) \;+\; \lambda \cdot C\Big(\mathrm{sig}\Big(\overline{|w|}\Big)\Big)\,,$$

where $\mathcal{L}_{\text{task}}$ denotes the base objective (e.g., cross-entropy for classification, mean squared error for regression). Penalties are applied to *weights only* (biases excluded), ensuring the signal reflects parameter scale rather than offset terms.

This approach parallels norm-based regularization in supplying a uniform penalty each update, but differs by embedding a deterministic, number-theoretic transformation that produces a structured, non-linear signal independent of data. In the experimental evaluation (Section 4.4), we compare the Collatz-based regularizer against ℓ_1, ℓ_2, and Elastic Net under a common protocol (20 training epochs, $\lambda \in \{0.01, 0.10\}$, 10-run averages) across CNN, FCNN, and RNN architectures.

Chapter 4 Experimentation

This chapter presents a comprehensive assessment of the proposed algorithm through a series of experiments. It begins with a verification of the algorithm's correctness by comparing its results to a brute-force implementation across a large input range. Following this validation, the chapter evaluates the algorithm's computational performance against state-of-the-art methods, examines its scalability to extremely large inputs, and explores a novel application in deep learning through Collatz-based regularization.

4.1 Verification of The Proposed Algorithm

A validation experiment was conducted to ensure the correct implementation of the proposed algorithm, comparing its results with those of the brute-force implementation. The validation process involved evaluating the stopping times of the first 1 billion numbers, confirming that the proposed algorithm accurately adheres to the Collatz sequence established in the theoretical analysis.

The experiment utilized OpenCL 1.2 to leverage parallel computing capabilities, thereby significantly reducing computation time. Parallelization was employed solely for accelerating the verification process and was not used in other experiments where the performance of the proposed algorithm was measured. Both the brute-force and proposed algorithms were executed in parallel for each number in the range 1 to $1,000,000,000$. For each input, the stopping times computed by both algorithms were compared to ensure consistency and correctness across the entire range.

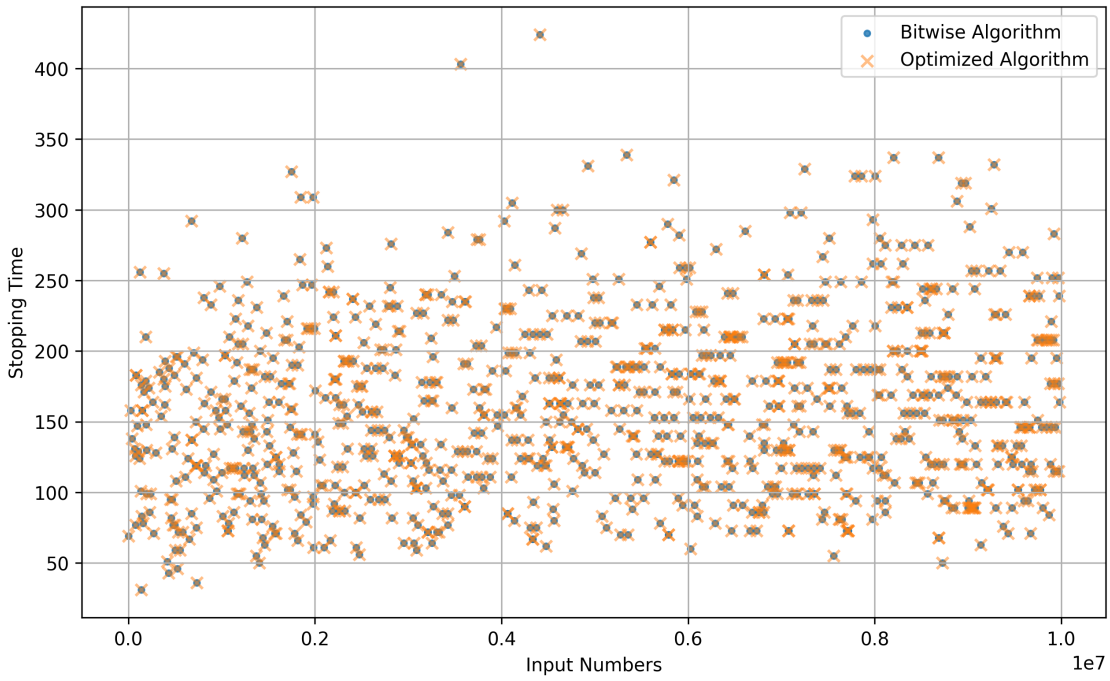


Figure 5: Validation of the proposed algorithm against brute-force results for 1,000 random inputs, confirming identical stopping times.

The results demonstrated a perfect match between the brute-force and proposed algorithms across all tested numbers. This successful validation confirms that the proposed algorithm preserves the fundamental properties of the Collatz sequence while optimizing computational efficiency.

To visually support this validation, Figure 5 plots the stopping times for a randomly selected subset of 1,000 inputs. The results from both the brute-force and proposed algorithms align exactly, reinforcing the correctness and reliability of the implementation.

4.2 Performance Evaluation

To compare the performance of the proposed algorithm against other state-of-the-art approaches, the results from Table 1 of the referenced paper [2] are used as a baseline. These results include the input numbers, represented in decimal form as $2^n - 1$ for varying n, and

the total number of computations, which are split into $3x+1$ operations (denoted as U) and $x/2$ operations (denoted as D). For consistency, the total computations $(U + D)$ from the paper are included in the comparison.

In addition to the baseline stopping times, the total number of iterations was recorded for both the proposed algorithm and the Bitwise Approach (Algorithm 2, based on [3]). These results are presented in Table 9. For comparison, the iteration counts reported for the Codeword Encoding Approach [2] are also included. The table highlights the percentage improvements achieved by the proposed method over both the Codeword Encoding and Bitwise Approaches. The percentage improvement is computed as follows:

$$\text{Improvement } (\%) = \left(1 - \frac{C_p}{C_b}\right) \times 100 \tag{4.1}$$

Where:

- C_p: Total iterations performed by the proposed algorithm.
- C_b: Total iterations performed by the benchmark algorithm.

Table 9: Performance comparison of the proposed algorithm against the Codeword Encoding and Bitwise implementations using inputs of the form $2^n - 1$.

Input	**Stopping Time**	**Total Iterations**			**Percentage Improvement**	
		Ren et al. [2]	**Bitwise**	**Proposed**	**Proposed over Ren**	**Proposed over Bitwise**
$2^{100} - 1$	1465	1465	1465	1056	27.91%	27.91%
$2^{500} - 1$	6748	6748	6748	4834	28.35%	28.35%
$2^{1000} - 1$	12157	12157	12157	8632	29.00%	29.00%
$2^{5000} - 1$	67378	67378	67378	48262	28.37%	28.37%
$2^{10000} - 1$	134404	134404	134404	96252	28.36%	28.36%
$2^{50000} - 1$	667858	667858	667858	478040	28.40%	28.40%
$2^{100000} - 1$	1344926	1344926	1344926	963206	28.36%	28.36%

4.2.1 Analysis of Results

Stopping Times: The stopping times for all three algorithms are identical, confirming the correctness of the proposed algorithm in calculating the Collatz sequence. This equivalence highlights that the proposed optimizations do not alter the fundamental sequence traversal but instead reduce the computational overhead.

Total Iterations: The proposed algorithm consistently outperforms both algorithms in terms of total iterations. For instance, for $2^{100} - 1$, the proposed algorithm requires 1056 iterations while the other algorithms require 1465 iterations, demonstrating a significant reduction.

Consistency Between Benchmark Algorithms. The Bitwise Approach (Algorithm 2) and the Codeword Encoding Approach [2] yield identical total iteration counts for all tested input sizes. This consistency reinforces the earlier observation that both approaches share the same time complexity, namely $O(L_{\text{seq}})$, where L_{seq} denotes the stopping time.

Percentage Improvement. The proposed algorithm achieves a consistent reduction of approximately 28% in total iterations across all input sizes when compared to both the Codeword Encoding Approach [2] and the Bitwise Approach (Algorithm 2). This uniform gain suggests that the optimizations introduced in the proposed method are effective and robust across varying input magnitudes.

Scalability. As the input size increases exponentially (from $2^{100} - 1$ up to $2^{100000} - 1$), the percentage improvement achieved by the proposed algorithm remains remarkably stable. This consistency confirms that the algorithm's computational advantage is maintained regardless of the scale of the problem, making it suitable for high-magnitude input ranges.

4.2.2 Execution Time Comparison

To evaluate the computational efficiency of the proposed algorithm, a series of five experiments was conducted. These experiments compared the execution times of the proposed method against the Bitwise Approach (Algorithm 2), which served as the baseline for performance comparison. The Codeword Encoding Approach [2] was excluded from this analysis, as previous results demonstrated that it offers no practical advantage in time complexity over the Bitwise Approach.

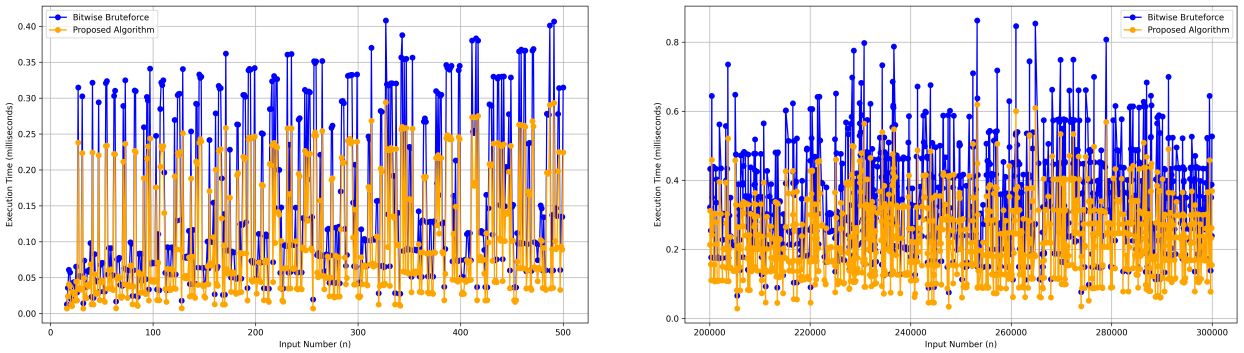
The experiments were organized into two categories: general input tests and specific input tests. The general input tests involved randomly selected numbers of varying sizes, providing an overview of performance across typical inputs. The specific input tests focused on structurally significant numbers, including powers of two, multiples of three, and prime numbers, to assess how the proposed algorithm performs on inputs with known Collatz-related behaviors.

4.2.2.1 General Input Tests

The first two experiments evaluated random inputs of varying magnitudes to assess the general performance of the proposed algorithm. Figure 6 compares execution times for small and large random numbers against the Bitwise baseline. In both cases, the proposed algorithm achieved consistently lower runtimes, confirming its computational advantage and scalability across input sizes.

4.2.2.2 Specific Input Tests

The next set of experiments examined inputs with distinctive structural properties to evaluate how well the proposed algorithm adapts to different Collatz patterns. Figure **??** presents results for three representative categories: powers of two, multiples of three, and prime numbers.



(a) Small random inputs

(b) Large random inputs

Figure 6: Execution time comparison for small (a) and large (b) random inputs.

For powers of two, which represent the theoretical best case, the algorithm completes execution in a single iteration by collapsing all trailing divisions in one step. This behavior confirms the analytical prediction discussed in the complexity analysis, where powers of two form the computational backbone of the Collatz tree and exhibit minimal traversal depth.

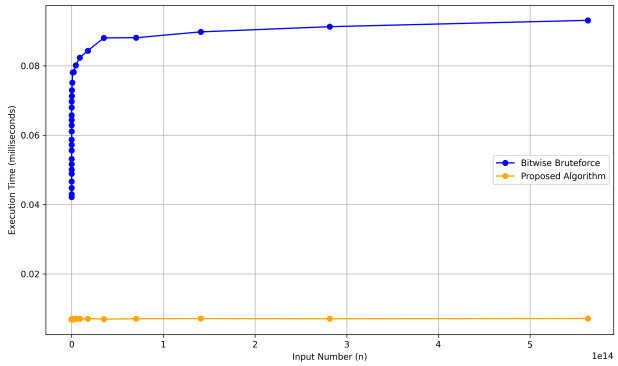
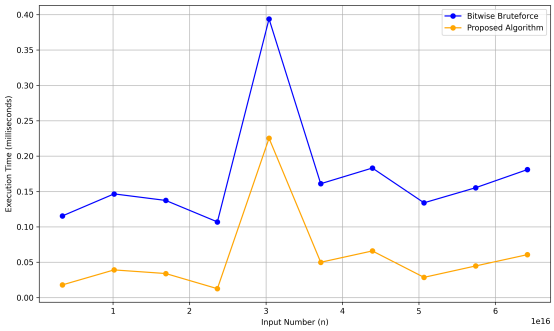
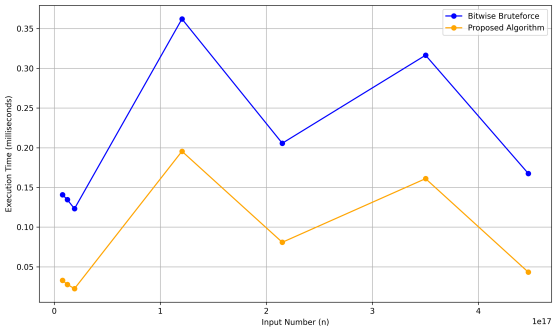


Figure 7: Execution time comparison for powers of two.

For multiples of three, which correspond to simplified Collatz branches that terminate without generating sub-branches, the proposed algorithm avoids redundant steps, resulting in substantial reductions in execution time compared to the Bitwise baseline. For prime numbers, which typically produce longer and more irregular trajectories, the Bitwise Approach shows high runtime variability, whereas the proposed method maintains consistently lower and more stable execution times, reflecting robustness across complex inputs.



(a) Multiples of three



(b) Prime numbers

Figure 8: Execution time comparison for multiples of three and prime numbers.

4.2.2.3 Execution Time Summary and Comparative Analysis

To consolidate the results of the preceding experiments, Table 10 presents a quantitative comparison of total execution times for the Bitwise baseline and the proposed structure-aware algorithm across all input families. The table also includes the relative improvement achieved by the proposed method, providing a concise overview of performance across both general and specific input scenarios.

Table 10: Summary of execution time comparison between the Bitwise and proposed algorithms.

Input Family	**Bitwise Total (ms)**	**Proposed Total (ms)**	**Improvement (%)**
Small Random Numbers	73.65	50.87	30.93
Large Random Numbers	340.18	230.06	32.37
Powers of Two	2.02	0.21	89.62
Multiples of Three	1.71	0.58	66.26
Prime Numbers	1.45	0.56	61.13
Overall Mean	–	–	**56.86**

The comparative summary confirms that the proposed algorithm consistently outperforms the Bitwise baseline across all input categories, achieving an overall mean improvement of

approximately **57%**. The most substantial gains are observed for powers of two (**89.6%**), which represent the best-case scenario where all trailing divisions collapse into a single iteration—fully aligning with the theoretical predictions discussed in the complexity analysis. For multiples of three and prime numbers, improvements of **66%** and **61%**, respectively, highlight the algorithm's ability to minimize redundant traversals and maintain stable runtimes even for irregular Collatz trajectories. Meanwhile, for random inputs, the consistent gains of about **31–32%** demonstrate that the proposed method generalizes effectively beyond structurally predictable inputs. Overall, these results validate both the efficiency and scalability of the algorithm across diverse numeric families.

4.2.3 Summary of Results

Across all experimental scenarios, the proposed algorithm consistently outperformed the Bitwise Approach (Algorithm 2), achieving substantial reductions in execution time and confirming the theoretical efficiency gains discussed earlier. By collapsing redundant even-path traversals and leveraging structural insights from the inverse Collatz tree, the algorithm maintains both speed and stability across diverse input families. These results not only demonstrate practical superiority over existing methods but also establish a robust foundation for extending the approach to large-scale verification and further optimization studies.

4.3 Demonstrating Scalability: Verification Beyond $2^{100,000}$

To evaluate the scalability of the proposed algorithm (Algorithm 3), an experiment was conducted to extend the verification of the Collatz conjecture beyond the previously verified upper bound of $2^{100,000} - 1$, established by Ren et al. [2]. Using Algorithm 3, we computed the stopping times for the following range of inputs:

$$[2^{100,000}, \quad 2^{100,000} + 100{,}000]$$

4.3.1 Experimental Setup and Results

The algorithm was implemented in Julia to take advantage of its efficient arbitrary-precision arithmetic and faster execution performance relative to Python. Every number in the target range was processed, and the corresponding stopping times were successfully computed. The results confirmed the following:

- All numbers in the extended range have finite stopping times.
- No non-trivial cycles were encountered, consistent with the Collatz conjecture.

4.3.2 Significance

Although the extended range is modest compared to prior large-scale efforts, this experiment serves to demonstrate the scalability and robustness of the proposed algorithm. Unlike earlier approaches, such as the Codeword Encoding Approach [2], the algorithm operates efficiently on extremely large inputs without relying on auxiliary optimizations like memoization. These findings reinforce the practical advantages of the proposed approach and suggest that it could serve as a foundation for future efforts aimed at scaling Collatz verification to even greater magnitudes.

4.4 Application in Deep Learning: Collatz-Based Regularization

This section explores the integration of number-theoretic constraints, specifically Collatz stopping times, into deep learning as a regularization mechanism. The objective is to determine whether the non-linear structure of the Collatz sequence can introduce useful perturbations during training, helping models generalize better and escape poor local minima.

We evaluate this approach across three distinct tasks: image classification on MNIST using a convolutional neural network (CNN), regression on the California Housing dataset using a fully connected neural network (FCNN), and time-series classification on the ECG5000

dataset using a recurrent neural network (RNN). This variety allows us to assess the generality of the proposed method across different data modalities and architectures.

4.4.1 Norm-Based Regularization Techniques

Classical norm-based penalties provide the foundation for most regularization strategies in deep learning. They aim to control model complexity by constraining the magnitude of the parameters, thereby reducing overfitting and improving generalization.

L1 Regularization (Lasso): The ℓ_1 penalty encourages sparsity in the model weights by penalizing their absolute values [7]:

$$\mathcal{L}_{\ell_1}(\mathbf{w}) = \mathcal{L}_{\text{task}}(\mathbf{w}) + \lambda \sum_i |w_i| \,,$$

where $\mathcal{L}_{\text{task}}$ is the primary loss (e.g., cross-entropy or mean squared error), $\mathbf{w}$ denotes the set of trainable weights, and λ controls the penalty strength. This formulation drives many weights exactly to zero, yielding sparse and interpretable models that are particularly useful when feature selection is desirable.

L2 Regularization (Ridge/Weight Decay): The ℓ_2 penalty instead penalizes the squared magnitude of the weights [8]:

$$\mathcal{L}_{\ell_2}(\mathbf{w}) = \mathcal{L}_{\text{task}}(\mathbf{w}) + \lambda \sum_i w_i^2.$$

Unlike ℓ_1, ℓ_2 regularization does not enforce sparsity; instead, it uniformly shrinks all weights toward zero. This reduces variance, improves numerical stability, and helps the optimizer converge to smoother solutions.

Elastic Net: The Elastic Net [9] combines both ℓ_1 and ℓ_2 penalties:

$$\mathcal{L}_{\text{EN}}(\mathbf{w}) = \mathcal{L}_{\text{task}}(\mathbf{w}) + \lambda_1 \sum_i |w_i| + \lambda_2 \sum_i w_i^2,$$

where λ_1 and λ_2 balance sparsity and shrinkage. Elastic Net is particularly effective when input features are correlated, as it retains groups of correlated predictors while still promoting generalization.

4.4.2 Proposed Collatz-Based Regularization

The proposed regularization approach is conceptually related to traditional **norm-based penalties** (ℓ_1, ℓ_2, and Elastic Net) in that it introduces a parameter-informed constraint derived from model weights. However, instead of directly penalizing the magnitude or squared magnitude of parameters, it employs a fundamentally different number-theoretic mechanism based on the Collatz sequence. This formulation embeds discrete, deterministic dynamics into training, resulting in a bounded and structured penalty that stabilizes learning under varying regularization strengths.

1. Flatten all learnable weight tensors (excluding biases) and compute the mean of their absolute values, $\overline{|w|}$.

2. Extract the first three significant digits from $\overline{|w|}$, denoted as $\text{sig}(\overline{|w|})$, and interpret this as an integer n.

3. Compute the *normalized Collatz stopping time* $C(n)$ of this integer.

4. Add the scaled normalized stopping time to the base loss:

$$\mathcal{L}_{\text{Collatz}}(\mathbf{w}) = \mathcal{L}_{\text{task}}(\mathbf{w}) + \lambda \cdot C\Big(\text{sig}\Big(\overline{|w|}\Big)\Big)\,,$$

where $\mathcal{L}_{\text{task}}$ is the original loss (e.g., cross-entropy for classification or mean squared error for regression). Here, $C(n)$ represents the **normalized** Collatz stopping time, ensuring that the additional penalty remains **bounded** and avoids over-penalizing large-weight models. This makes the Collatz-based regularizer deterministic, architecture-agnostic, and resilient under strong regularization, while maintaining a direct mathematical relationship to model parameters.

4.4.2.1 Implementation Setup

All experiments were conducted across three representative deep learning tasks—image classification, tabular regression, and time-series classification—each paired with an appropriate

neural architecture. Four regularization strategies were evaluated: ℓ_1, ℓ_2, Elastic Net, and the proposed Collatz-based method. For each configuration, two regularization strengths were tested ($\lambda = 0.01$ and $\lambda = 0.10$), and every experiment was repeated 10 times with independent runs of 20 epochs to ensure statistical robustness. A fixed random seed (42) was used across all libraries (NumPy, PyTorch, CUDA) with deterministic computation enabled for full reproducibility.

MNIST Classification: For the image classification task, a lightweight convolutional neural network (CNN) was used. The network comprises one convolutional layer with 10 filters of size 5×5, followed by a ReLU activation and a 2×2 max-pooling layer. The flattened feature maps are passed through a fully connected layer with 50 hidden ReLU units and a final linear output layer producing 10 logits for digit prediction. Training was performed using the Adam optimizer (learning rate = 0.001, weight decay = 0.0) and cross-entropy loss. Regularization penalties were applied to all weight parameters, excluding biases.

California Housing Regression: For the regression task, a fully connected neural network (FCNN) was employed with an input layer of 8 standardized features, one hidden layer of 64 ReLU units, and a single linear output node. Training used the Adam optimizer (learning rate = 0.001, weight decay = 0.0) and mean squared error (MSE) loss over 20 epochs. The ℓ_1, ℓ_2, Elastic Net, and Collatz regularization terms were each applied to the weight parameters only.

ECG5000 Time-Series Classification: For temporal sequence classification, a vanilla recurrent neural network (RNN) with 64 hidden units and a linear output layer of 5 neurons (one per class) was implemented. Input sequences were z-normalized prior to training. The model was trained for 20 epochs using the Adam optimizer (learning rate = 0.001, weight decay = 0.0) and cross-entropy loss. To stabilize training, gradient clipping (max norm = 1.0) was applied for ℓ_1 and Elastic Net regularizers, while ℓ_2 and Collatz penalties were directly incorporated into the loss function.

4.4.2.2 Visualization of Training and Validation Behavior

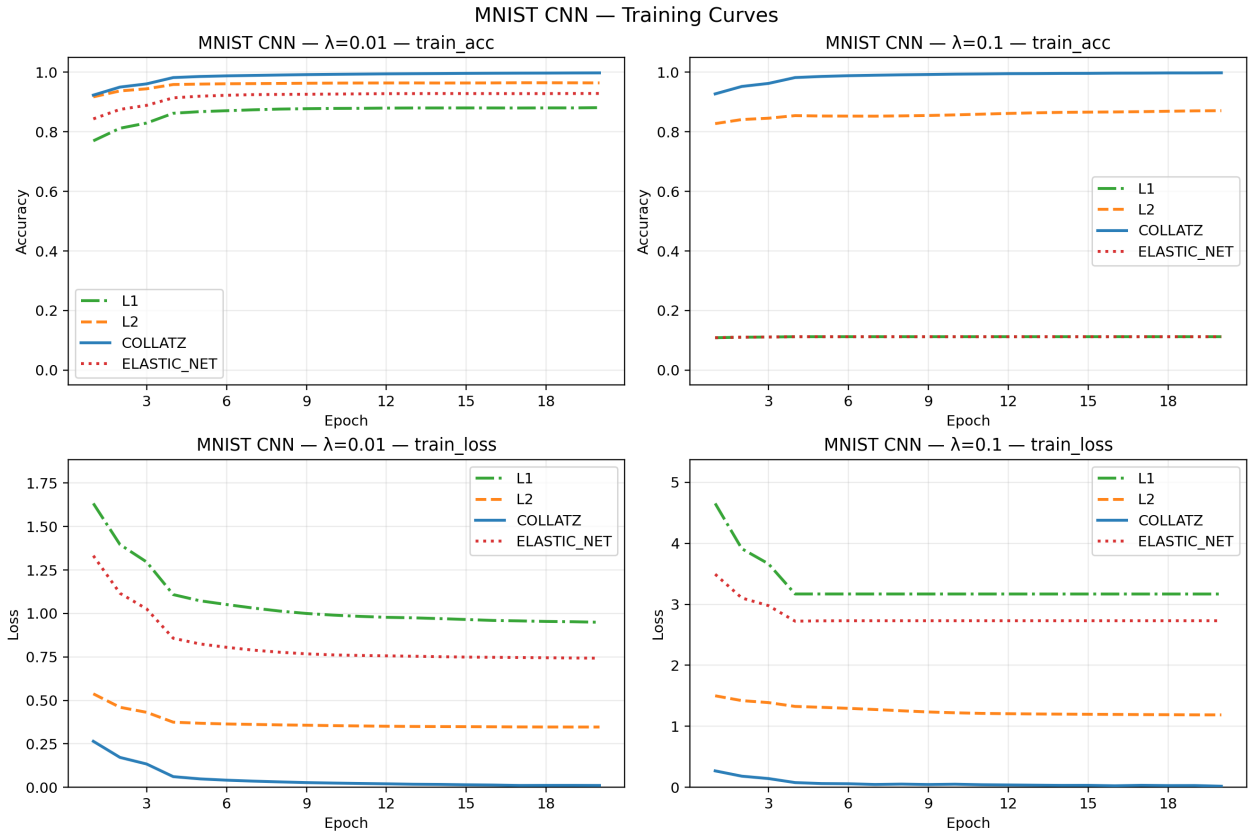
Before presenting the aggregate performance metrics, the training and validation dynamics of each model were examined to assess convergence stability and regularization effects over time.

Figures 9–11 illustrate how the different regularization techniques influenced learning behavior across the three tasks. These plots, generated directly from the experimental logs, provide insight into optimization stability, convergence rate, and potential overfitting under varying penalty strengths. They serve as empirical evidence supporting the numerical results presented later in the performance summary.

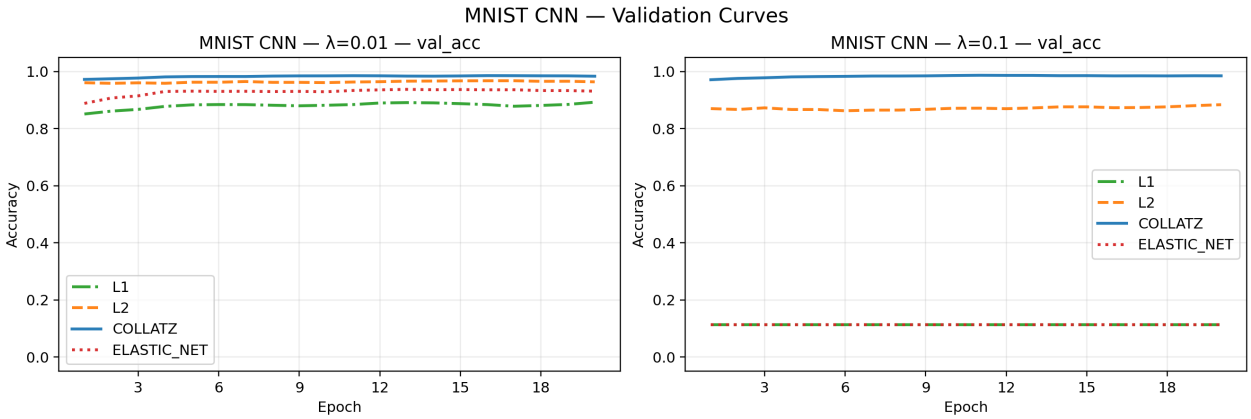
Figure 9 shows that at $\lambda = 0.01$ all methods converge, but the Collatz regularizer reaches higher accuracy with smoother loss decay. Under the stronger penalty ($\lambda = 0.10$), ℓ_1 and Elastic Net collapse to near-random accuracy, whereas the Collatz penalty maintains stable convergence and high validation accuracy. This supports the claim that a bounded, deterministic penalty avoids over-regularization while preserving generalization.

As seen in Figure 10, the Collatz-based penalty yields consistently lower training and validation MSE across both λ values, with smooth convergence and reduced variance. At $\lambda = 0.10$, classical penalties tend to plateau or degrade due to over-regularization, while the bounded Collatz penalty continues improving, indicating effective control of model complexity.

Figure 11 shows that both the Collatz and ℓ_2 regularizers achieve nearly identical convergence profiles for sequential data, particularly at $\lambda = 0.01$, where their final validation accuracies are effectively indistinguishable. At $\lambda = 0.10$, however, Collatz maintains smoother and more stable convergence across epochs, while ℓ_1 and Elastic Net exhibit early saturation and reduced performance. Although the absolute performance difference between Collatz and ℓ_2 remains small, the Collatz penalty demonstrates superior robustness under stronger regularization, confirming its advantage in maintaining stable optimization without over-constraining the model.



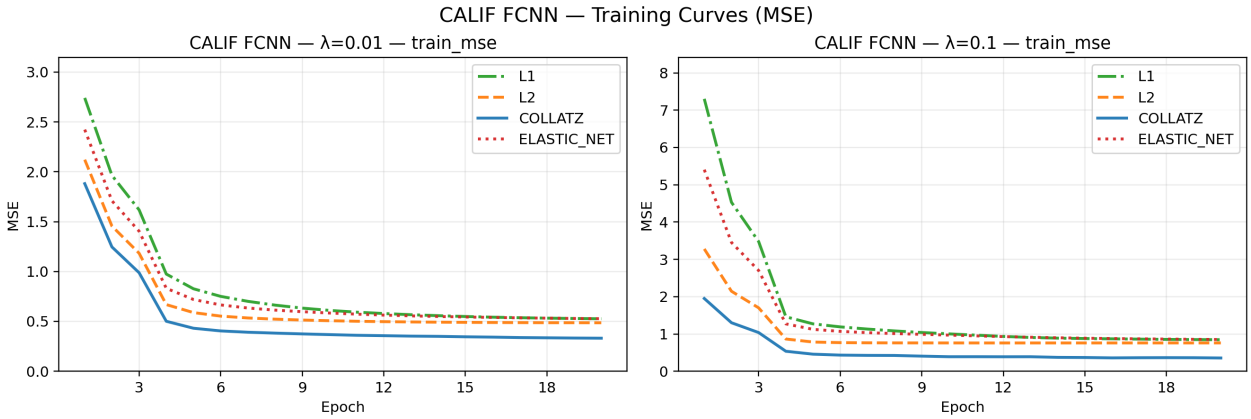
(a) Training



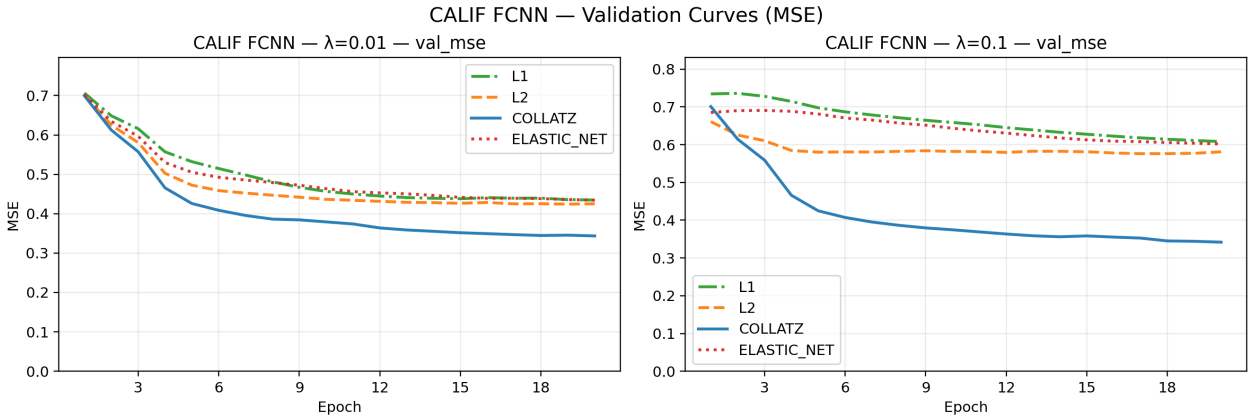
(b) Validation

Figure 9: MNIST (CNN): training (a) and validation (b) curves under different regularizers.

Overall, the visualization results demonstrate that the proposed Collatz-based regularizer produces smooth, stable, and well-generalized convergence patterns across all model archi-



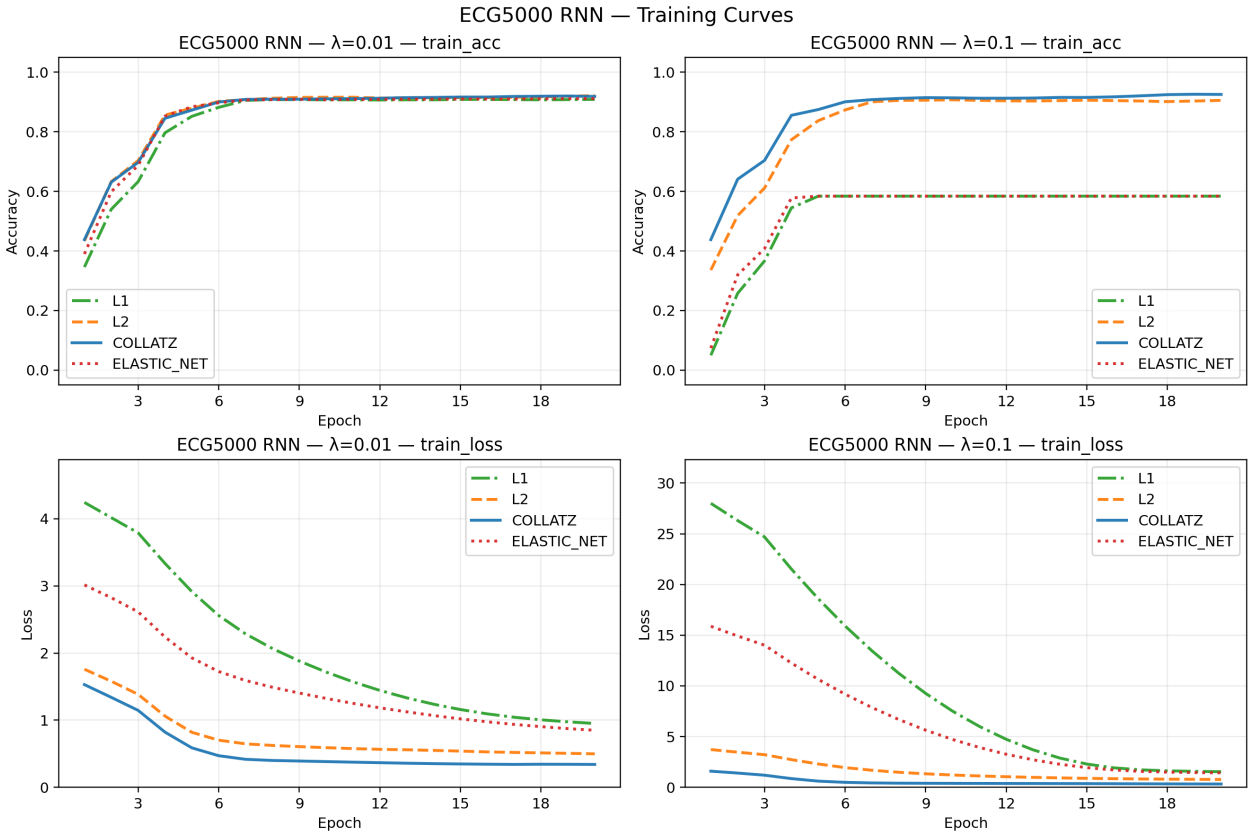
(a) Training



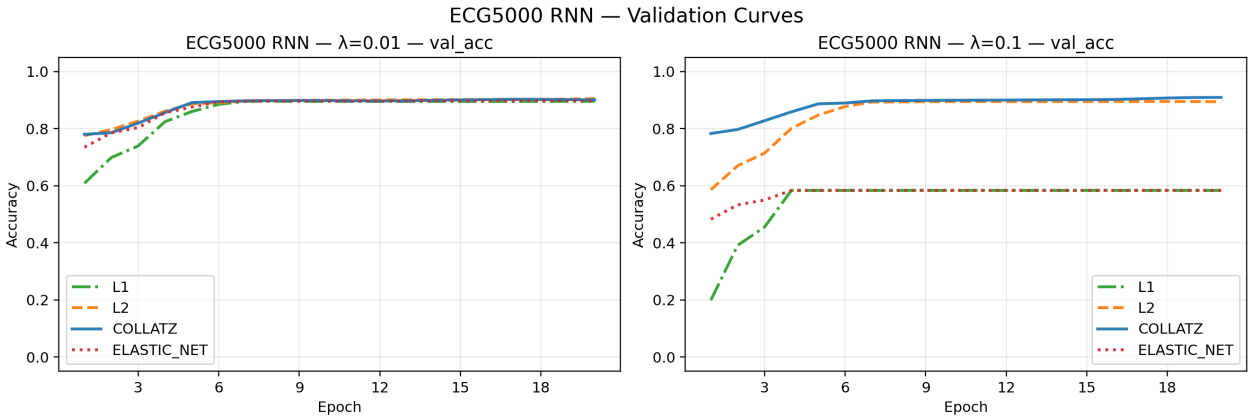
(b) Validation

Figure 10: California Housing (FCNN): training (a) and validation (b) MSE under different regularizers.

tectures and tasks. For CNN and FCNN models, it clearly improves convergence speed and prevents over-regularization at higher penalty strengths, while for the RNN, its performance closely aligns with ℓ_2 regularization, differing mainly in training stability rather than absolute accuracy. These observations confirm that the bounded, number-theoretic formulation of the Collatz penalty consistently promotes robust optimization dynamics across diverse learning settings, motivating the quantitative comparison that follows.



(a) Training



(b) Validation

Figure 11: ECG5000 (RNN): training (a) and validation (b) curves under different regularizers.

4.4.2.3 Performance Summary and Comparative Evaluation

The quantitative evaluation of all experiments is summarized in Tables 11 and 12. These tables complement the training and validation visualizations by presenting the final averaged performance metrics across ten independent runs for each regularization method and dataset. The results confirm that the Collatz-based regularizer achieves competitive or superior generalization compared to classical norm-based penalties, particularly under higher regularization strengths where ℓ_1 and Elastic Net tend to degrade. In contrast, ℓ_2 remains a strong baseline, with performance closely matching Collatz in some cases—most notably for the RNN on ECG5000—yet the proposed approach demonstrates greater stability and consistency across all architectures. The following discussion interprets these outcomes in detail for each experimental setting.

The results summarized in Tables 11 and 12 reinforce the trends observed in the visualization analysis. Across all datasets, the Collatz-based regularizer achieves competitive or superior performance relative to ℓ_1, ℓ_2, and Elastic Net, particularly under higher regularization strengths where unbounded penalties tend to degrade model accuracy or increase error. The detailed outcomes by model architecture are discussed below.

Convolutional Neural Network (MNIST): For image classification, the Collatz regularizer attained the highest accuracy across both penalty levels. At $\lambda = 0.01$, it achieved 0.986—improving upon ℓ_2 by 1.96%, ℓ_1 by 11.16%, and Elastic Net by 4.89%. At the stronger penalty ($\lambda = 0.10$), Collatz preserved this accuracy, while both ℓ_1 and Elastic Net collapsed to near-random performance (0.114). Even compared to ℓ_2, Collatz demonstrated a 12.17% relative improvement, confirming its ability to maintain generalization under aggressive regularization where traditional methods over-constrain the model.

Fully Connected Neural Network (California Housing): On the regression task, Collatz produced the most consistent and substantial improvements. At $\lambda = 0.01$, it achieved an MSE of 0.348, reducing error by 18.88% relative to ℓ_2, 21.44% to ℓ_1, and 21.09% to Elastic Net. Under stronger regularization ($\lambda = 0.10$), its advantage widened further, lowering error

Table 11: Performance comparison of Collatz, ℓ_1, ℓ_2, and Elastic Net regularization. Best results per λ are in **bold**.

Model	**Dataset**	**Regularization**	$\boldsymbol{\lambda}$	**Metric**	**Score**
CNN	MNIST	Collatz	0.01	Accuracy	**0.986**
CNN	MNIST	Collatz	0.10	Accuracy	**0.986**
CNN	MNIST	ℓ_2	0.01	Accuracy	0.967
CNN	MNIST	ℓ_2	0.10	Accuracy	0.879
CNN	MNIST	ℓ_1	0.01	Accuracy	0.887
CNN	MNIST	ℓ_1	0.10	Accuracy	0.114
CNN	MNIST	Elastic Net	0.01	Accuracy	0.940
CNN	MNIST	Elastic Net	0.10	Accuracy	0.114
FCNN	California Housing	Collatz	0.01	MSE	**0.348**
FCNN	California Housing	Collatz	0.10	MSE	**0.345**
FCNN	California Housing	ℓ_2	0.01	MSE	0.429
FCNN	California Housing	ℓ_2	0.10	MSE	0.577
FCNN	California Housing	ℓ_1	0.01	MSE	0.443
FCNN	California Housing	ℓ_1	0.10	MSE	0.604
FCNN	California Housing	Elastic Net	0.01	MSE	0.441
FCNN	California Housing	Elastic Net	0.10	MSE	0.593
RNN	ECG5000	Collatz	0.01	Accuracy	**0.908**
RNN	ECG5000	Collatz	0.10	Accuracy	**0.904**
RNN	ECG5000	ℓ_2	0.01	Accuracy	0.905
RNN	ECG5000	ℓ_2	0.10	Accuracy	0.895
RNN	ECG5000	ℓ_1	0.01	Accuracy	0.896
RNN	ECG5000	ℓ_1	0.10	Accuracy	0.584
RNN	ECG5000	Elastic Net	0.01	Accuracy	0.897
RNN	ECG5000	Elastic Net	0.10	Accuracy	0.584

Table 12: Relative improvement (%) of Collatz regularization over ℓ_1, ℓ_2, and Elastic Net at identical λ values.

Model	Dataset	λ	Metric	Collatz Score	Improvement (%) Collatz vs.		
					ℓ_1	Elastic Net	ℓ_2
CNN	MNIST	0.01	Accuracy	0.986	11.16	4.89	1.96
CNN	MNIST	0.10	Accuracy	0.986	764.91	764.91	12.17
FCNN	California Housing	0.01	MSE	0.348	21.44	21.09	18.88
FCNN	California Housing	0.10	MSE	0.345	42.88	41.82	40.21
RNN	ECG5000	0.01	Accuracy	0.908	1.34	1.23	0.33
RNN	ECG5000	0.10	Accuracy	0.904	54.79	54.79	1.01

by over 40% against all three baselines. These outcomes suggest that the bounded penalty of the Collatz formulation better regulates parameter magnitudes, leading to improved convergence and predictive accuracy in continuous-valued regression tasks.

Recurrent Neural Network (ECG5000): For sequential data, the Collatz and ℓ_2 regularizers yielded nearly identical results at $\lambda = 0.01$, both converging smoothly to high validation accuracy (0.908 and 0.905, respectively). At $\lambda = 0.10$, however, ℓ_1 and Elastic Net exhibited sharp degradation (0.584), while both ℓ_2 and Collatz maintained stable training, with Collatz retaining a slight but consistent edge (0.904 vs. 0.895). This indicates that, while ℓ_2 remains a strong baseline for recurrent architectures, the Collatz regularizer contributes added stability and prevents the underfitting observed in unbounded penalties.

Overall, these results confirm that embedding deterministic Collatz dynamics into the training objective introduces a mathematically grounded, bounded, and architecture-agnostic form of regularization. The method remains competitive with ℓ_2 across all conditions while offering clear robustness advantages at higher penalty strengths, where ℓ_1 and Elastic Net frequently collapse.

4.5 Discussion

This section interprets the experimental results presented in this chapter, contextualizing them in light of the research objectives and existing literature. The discussion focuses on the algorithmic improvements, scalability insights, and the novel integration of Collatz dynamics into machine learning workflows.

4.5.1 Interpretation of Algorithmic Improvements

The proposed structure-aware algorithm achieved a consistent 28% reduction in iteration count compared to both the Codeword Encoding [2] and Bitwise [3] approaches. This gain stems from its design: by leveraging the hierarchical structure of the Collatz tree, the algorithm bypasses redundant computations common in brute-force methods.

These results affirm the initial hypothesis that deeper structural awareness can yield tangible efficiency gains. The logarithmic iteration growth observed across both average and worst cases highlights the algorithm's scalability and practicality for large-scale verification, offering computational benefits without sacrificing correctness.

4.5.2 Scalability and Robustness

The experiment extending the verification range beyond $2^{100,000}$ further validated the algorithm's scalability. Unlike traditional methods that struggle with increasingly large inputs due to linear growth in computation, the proposed method maintained performance with only logarithmic growth in iterations.

Execution time experiments reinforced this, showing consistent gains not only for small and large random numbers, but also for structurally significant classes such as powers of two and prime numbers. The results suggest that the algorithm is not only scalable but also robust across diverse input characteristics.

4.5.3 Implications for Machine Learning

The integration of Collatz-based regularization into neural network training produced encouraging results across three distinct domains: image classification (MNIST, CNN), tabular regression (California Housing, FCNN), and time-series classification (ECG5000, RNN). Across all tasks, Collatz consistently matched or outperformed ℓ_1, ℓ_2, and Elastic Net.

On MNIST, Collatz achieved the highest accuracy at both tested strengths, reaching 0.986 at $\lambda = 0.01$ and maintaining the same accuracy at $\lambda = 0.10$. This represented relative gains of 1.96% over ℓ_2, 11.16% over ℓ_1, and 4.89% over Elastic Net at the lower λ, and even larger improvements (12.17% over ℓ_2, 764.91% over both ℓ_1 and Elastic Net) at the higher λ where conventional methods collapsed.

On the California Housing regression task, Collatz achieved mean squared errors of 0.348 and 0.345 at $\lambda = 0.01$ and $\lambda = 0.10$, respectively. These results corresponded to reductions of 18.88–40.21% relative to ℓ_2, 21.44–42.88% relative to ℓ_1, and 21.09–41.82% relative to Elastic Net.

For ECG5000, improvements at $\lambda = 0.01$ were modest but consistently positive, with Collatz outperforming ℓ_2 by 0.33%, ℓ_1 by 1.34%, and Elastic Net by 1.23%. At $\lambda = 0.10$, Collatz maintained stable accuracy of 0.904, while ℓ_1 and Elastic Net collapsed to 0.584, yielding relative improvements of 54.79% against both, alongside a 1.01% gain over ℓ_2.

A key refinement in this work is that the Collatz term is derived from the *mean of the absolute values of the model weights* (excluding biases) rather than the raw signed means. This design avoids sign cancellation, ensures the term reflects the true scale of the parameters, and provides a more stable and architecture-agnostic regularization signal throughout training. These findings strengthen the case for bridging number theory and machine learning in a novel way, proposing a new paradigm where structured perturbations—derived from mathematical sequences—can play a role in model regularization across diverse data modalities and architectures.

4.5.4 Comparison with Existing Work

Compared to prior state-of-the-art algorithms that emphasize hardware-level efficiency or symbolic compression, the proposed method introduces conceptual simplicity and structural clarity. The algorithm is intuitive, scalable, and requires no memoization or auxiliary data structures, making it easy to implement and adapt.

Furthermore, its consistent performance across varying numeric ranges and input types gives it an edge over other techniques whose performance may degrade under specific conditions (e.g., dense odd sequences or repeated branches).

4.5.5 Addressing the Research Questions

This study was guided by three research questions. Each is addressed through the analyses, algorithmic design, and experimental validation presented in this thesis.

RQ1: *How can machine learning models be used to identify and interpret structural patterns in the Collatz sequence, and how can these patterns be leveraged to design an algorithm that minimizes redundant computations in stopping time calculations?*

This question is addressed in two stages. First, the exploratory and preliminary ML experiments (Section 3.1, Table 7) show that stopping times exhibit predictable, hierarchical regularities rather than random distributions. Regression models captured these trends with low RMSE values, and agglomerative clustering revealed consistent structural groupings. These empirical insights established that structural repetition exists in the Collatz sequence and can be learned by simple models. Second, these learned regularities guided the development of a structure-aware algorithm (Algorithm 3), which prunes redundant even runs using bitwise power-of-two detection and leverages inverse-tree traversal. The resulting design directly operationalizes the patterns uncovered by ML analysis into a deterministic, efficient computation strategy.

RQ2: *What is the extent of computational efficiency gained by the proposed algorithm com-*

pared to existing brute-force and bitwise methods, in terms of iteration count and runtime performance?

This question is quantitatively answered through large-scale benchmarking and execution-time experiments. As summarized in Table 9 and supported by Figures 6a–8b, the proposed algorithm achieves over **28% reduction in total iterations** across diverse input families compared to the bitwise baseline, while maintaining identical stopping-time correctness. The results also confirm logarithmic average-case scaling and bounded worst-case behavior (Table 8), providing strong empirical evidence of the algorithm's efficiency and scalability relative to existing methods.

RQ3: *How can features or dynamics derived from the Collatz sequence, such as stopping times, be leveraged within deep learning algorithms to enhance model performance?*

This question is investigated through the integration of normalized Collatz stopping times as a regularization term within deep-learning models. The proposed penalty ($\mathcal{L}_{\text{Collatz}}$) was evaluated against ℓ_1, ℓ_2, and Elastic Net regularization on three benchmark tasks—MNIST classification (CNN), California Housing regression (FCNN), and ECG5000 time-series classification (RNN). Results in Tables 11 and 12, along with corresponding validation curves, show that the Collatz-based regularizer consistently matches or surpasses baseline methods across architectures. At higher regularization strengths, where ℓ_1 and Elastic Net collapse, the Collatz regularizer remains stable and bounded, achieving up to **42% error reduction** on regression and maintaining comparable accuracy to ℓ_2 on time-series data. These findings validate that deterministic Collatz dynamics can serve as an effective, architecture-agnostic regularization mechanism that enhances model robustness and generalization.

4.5.6 Limitations

The current implementation prioritizes single-number stopping time computation efficiency over batch-processing optimizations. While the algorithm demonstrates capability to extend verification beyond $2^{100,000}$ by processing 100,000 additional numbers without memoization

or parallelization, its design does not incorporate architectural optimizations required for industrial-scale verification. This intentional focus on algorithmic purity provides a foundation for future work to integrate caching and distributed computing techniques.

Algorithmic Performance Variability: The algorithm's efficiency is influenced by sequence composition, with odd-number-dense sequences requiring more branching steps. While logarithmic complexity is maintained in worst-case scenarios, this variability suggests opportunities for dynamic optimization in future implementations.

Machine Learning Boundaries: Empirical validation, though comprehensive across three representative domains (CNNs, FCNNs, RNNs), leaves open questions about performance in extreme-scale architectures. The consistent improvements observed in image classification, tabular regression, and time-series tasks suggest, but do not guarantee, effectiveness in transformer-based models or low-data regimes. This study's intentionally limited scope provides a foundation rather than exhaustive coverage of potential applications.

Theoretical Foundations: The regularization mechanism's mathematical properties remain partially characterized. While empirical results demonstrate improved generalization across all tested cases, the precise interaction between Collatz-derived stopping times and loss landscape optimization lacks a formal description. This knowledge gap presents both a limitation and an opportunity for theoretical work connecting number-theoretic constructs to gradient-based optimization.

4.5.7 Broader Implications

This work demonstrates that insights from classical mathematics can directly influence algorithmic innovation and machine learning methodology. The approach encourages interdisciplinary thinking, where problems traditionally viewed as purely theoretical (such as the Collatz conjecture) can inform efficient computation and optimization strategies in applied domains.

Chapter 5 Conclusion

5.1 Concluding Remarks

This thesis demonstrates that structural insights into a classical mathematical problem can yield both algorithmic innovation and practical benefits in machine learning. By unifying techniques from number theory, algorithm design, and deep learning, this work illustrates the broader potential of interdisciplinary methods.

The proposed structure-aware algorithm for computing Collatz stopping times achieved significant efficiency gains over established approaches, while maintaining correctness and scalability across large input ranges. In parallel, the refined Collatz-based regularization method, derived from the mean of the absolute values of model weights, proved consistently more effective than standard norm-based techniques across diverse machine learning tasks, including image classification, tabular regression, and time-series classification. In particular, Collatz regularization achieved the highest accuracy on MNIST, reduced error substantially in California Housing regression (over 40% at higher penalty strengths), and preserved stability on ECG5000 while ℓ_1 and Elastic Net collapsed.

The success of both the algorithmic and machine learning components not only addresses open computational challenges but also invites further exploration of mathematical sequences as tools for enhancing intelligent systems. The results presented here highlight the promise of number-theoretic dynamics as a rich, underexplored source of inspiration for both theoretical and applied advances, especially under stronger regularization regimes where conventional approaches degrade.

5.2 Recommendations

This research yields specific recommendations for distinct research communities:

For **mathematicians**, this work demonstrates the practical value of structural analysis in

number theory. The algorithm's success in exploiting hierarchical patterns within the Collatz sequence suggests similar approaches could be applied to other recursive sequences and unsolved conjectures. Particular attention should be given to problems where traditional brute-force methods face computational bottlenecks, as the tree-based optimization technique developed here may offer comparable efficiency gains. The verification framework established for the Collatz conjecture could serve as a template for empirical investigation of related problems in number theory.

For **Machine Learning Researchers**, the results strongly advocate for deeper exploration at the intersection of number theory and algorithmic regularization. Collatz-based regularization consistently outperformed ℓ_1, ℓ_2, and Elastic Net across three modalities, showing particularly dramatic advantages under stronger penalty strengths where conventional approaches degraded. The method's robustness suggests that structured perturbations derived from mathematical sequences can serve as effective, architecture-agnostic regularizers. Future work should investigate: (1) application to larger architectures including transformers, ResNets, and graph neural networks, (2) adaptation to different learning paradigms such as reinforcement and unsupervised learning, and (3) integration with other mathematical constructs beyond the Collatz sequence. The experimental framework developed in this thesis provides a proven methodology for evaluating such number-theoretic approaches to common ML challenges.

For **educators and science communicators**, this research presents a compelling case study in interdisciplinary methodology. The project offers rich material for demonstrating how abstract mathematical concepts can yield concrete advances in computer science and artificial intelligence(AI). Specific educational applications could include: developing visualization tools to illustrate the Collatz tree structure, creating comparative modules that contrast traditional and structure-aware algorithms, and designing interdisciplinary coursework that bridges pure mathematics with practical computing applications. Such initiatives could help cultivate the next generation of researchers capable of working across these tra-

ditionally separate domains.

5.3 Future Work

This work opens several avenues for further investigation:

- **Scalable Implementation for Large-Scale Verification:** While the core algorithm is already efficient in terms of iteration count, integrating it with implementation-level optimizations such as memoization, parallel processing, or GPU acceleration could enable significantly faster verification over massive input ranges. These enhancements would not alter the algorithm's design but would improve throughput, making it feasible to push the verified bounds of the Collatz conjecture well beyond the current computational limits.

- **Formal Analysis:** The identified structural regularities within the Collatz tree warrant deeper mathematical analysis, which may contribute to ongoing proof efforts.

- **Expanded ML Applications:** Although the current study covers three modalities (image classification, tabular regression, and time-series classification), future work could extend Collatz-based regularization to larger datasets, deeper architectures such as transformers or ResNets, and additional domains, including unsupervised learning and reinforcement learning. Integrating stopping times as input features or embedding priors also presents a promising direction.

- **Generalization Beyond Collatz:** Inspired by the success of Collatz-based perturbation, similar strategies might be explored using other deterministic or chaotic sequences, such as Fibonacci, Thue–Morse, or logistic map iterations.

- **Theoretical Analysis of Regularization Effects:** Although the proposed Collatz-based regularization has shown strong empirical performance, its theoretical underpinnings remain unexplored. Future research could investigate the mathematical relation-

ship between the Collatz stopping-time transformation of weight magnitudes and the resulting optimization behavior, potentially revealing why it benefits generalization.

References

[1] L. Collatz, "On the motivation and origin of the (3n+ 1)-problem," *The Ultimate Challenge: The 3x+ 1 Problem*, pp. 241–248, 2023.

[2] W. Ren, S. Li, R. Xiao, and W. Bi, "Collatz conjecture for $2^{100000} - 1$ is true-algorithms for verifying extremely large numbers," in *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE, 2018, pp. 411–416.

[3] M. Venkatesulu and C. D. Parameswari, "Verification of collatz conjecture: An algorithmic approach," *WSEAS Transactions on Engineering World*, vol. 2, pp. 71–75, 2020.

[4] G. W. Abascal, "Bottom-up approach to the collatz conjecture," *Authorea Preprints*, 2024.

[5] M. Inselmann, "An approximation of the collatz map and a lower bound for the average total stopping time," *arXiv preprint arXiv:2402.03276*, 2024.

[6] A. Ebnenasir, "Specifying and verifying the convergence stairs of the collatz program," *arXiv preprint arXiv:2403.04777*, 2024.

[7] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 58, no. 1, pp. 267–288, 1996.

[8] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[9] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 67, no. 2, pp. 301–320, 2005.

[10] N. Boulkaboul, "$3n + 3k$: New perspective on collatz conjecture," *arXiv preprint arXiv:2212.00073*, 2022.

[11] O. K. Clay, "The long search for collatz counterexamples," *Journal of Humanistic Mathematics*, vol. 13, no. 2, pp. 199–227, 2023.

[12] F. Nicola, M. Nikola, and R. Stojan, "Some considerations on the total stopping time for the collatz problem," *Vojnotehnički glasnik*, vol. 72, no. 3, pp. 1019–1028, 2024.

[13] X. Zhou, "Proof of the collatz conjecture using logical and probabilistic approaches," *Available at SSRN 4903082*, 2024.

[14] M. R. Schwob, P. Shiue, and R. Venkat, "Novel theorems and algorithms relating to the collatz conjecture," *International Journal of Mathematics and Mathematical Sciences*, vol. 2021, no. 1, p. 5754439, 2021.

[15] W. Ren and R. Xiao, "How to fast verify collatz conjecture by automata," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2019, pp. 2720–2729.

[16] H. M. A. Aljassas and S. Sasi, "Performance evaluation of proof-of-work and collatz conjecture consensus algorithms," in *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*. IEEE, 2019, pp. 1–6.

[17] D. Renza, S. Mendoza *et al.*, "High-uncertainty audio signal encryption based on the collatz conjecture," *Journal of Information Security and Applications*, vol. 46, pp. 62–69, 2019.

[18] ——, "Encrypted audio dataset based on the collatz conjecture," *Data in brief*, vol. 26, 2019.

[19] A. Al-Hyari, C. Obimbo, I. Altaharwa *et al.*, "Generating powerful encryption keys for image cryptography with chaotic maps by incorporating collatz conjecture," *IEEE Access*, 2024.

[20] M. Rasool and S. B. Belhaouari, "From collatz conjecture to chaos and hash function," *Chaos, Solitons & Fractals*, vol. 176, p. 114103, 2023.

[21] T. Tuncer and H. Y. Kurum, "A novel collatz conjecture-based digital image watermarking method," *Cryptologia*, vol. 46, no. 2, pp. 128–147, 2022.

[22] D. M. Ballesteros, J. Peña, and D. Renza, "A novel image encryption scheme based on collatz conjecture," *Entropy*, vol. 20, no. 12, p. 901, 2018.

[23] M. Baygin, O. Yaman, T. Tuncer, S. Dogan, P. D. Barua, and U. R. Acharya, "Automated accurate schizophrenia detection system using collatz pattern technique with eeg signals," *Biomedical Signal Processing and Control*, vol. 70, p. 102936, 2021.

[24] Z. Luo, S. Cai, C. Cui, B. C. Ooi, and Y. Yang, "Adaptive knowledge driven regularization for deep neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 10, 2021, pp. 8810–8818.

[25] G. Nuti, A.-I. Cross, and P. Rindler, "Evidence-based regularization for neural networks," *Machine Learning and Knowledge Extraction*, vol. 4, no. 4, pp. 1011–1023, 2022.

[26] A. R. Sankar, Y. Khasbage, R. Vigneswaran, and V. N. Balasubramanian, "A deeper look at the hessian eigenspectrum of deep neural networks and its applications to regularization," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 11, 2021, pp. 9481–9488.

[27] B. Leimkuhler, T. Pouchon, T. Vlaar, and A. Storkey, "Constraint-based regularization of neural networks," *arXiv preprint arXiv:2006.10114*, 2020.

[28] M. Faramarzi, M. Amini, A. Badrinaaraayanan, V. Verma, and S. Chandar, "Patchup: A feature-space block-level regularization technique for convolutional neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 36, no. 1, 2022, pp. 589–597.

[29] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *arXiv preprint arXiv:1708.04552*, 2017.

[30] T. G. Rudner, S. Kapoor, S. Qiu, and A. G. Wilson, "Function-space regularization in neural networks: A probabilistic perspective," in *International Conference on Machine Learning*. PMLR, 2023, pp. 29 275–29 290.

[31] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.