INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA DE INTELIGÊNCIA ARTIFICIAL




Tetrahedron-Tetrahedron Intersection and Volume Computation Using Neural Networks

ERENDIRO SANGUEVE NJUNJUVILI PEDRO
Julho de 2025

Tetrahedron-Tetrahedron Intersection and Volume Computation Using Neural Networks

Erendiro Sangueve Njunjuvili Pedro

Student No.: 1160555

A dissertation submitted in partial fulfilment of the requirements for the degree of Master of Science, Area of Artificial Intelligence Engineering

Supervisor:
Dr. Gabriel Zachmann, Full Professor, University of Bremen
Dr. Carlos Fernando da Silva Ramos, Full Professor, Institute of Engineering, Polytechnic of Porto

Technical Supervision:
Dr. Jorge M. Santos, Coordinating Professor, Institute of Engineering, Polytechnic of Porto
Dr. René Weller, Coordinating Professor, University of Bremen
Navid Mirzayousef Jadid, Researcher, University of Bremen
Thomas Hudcovic, Researcher, University of Bremen

Evaluation Committee:
President:
Dr. António Constantino Lopes Martins, Coordinating Professor at the School of Engineering, Polytechnic Institute of Porto

Members:
Dr. Luiz Felipe Rocha de Faria, Coordinating Professor at the School of Engineering, Polytechnic Institute of Porto
Dr. Gabriel Zachmann, Full Professor, University of Bremen

Porto, July 1, 2025

Dedicatory

To those looking for beautiful questions.

Abstract

This thesis introduces a framework for fast, learning-based analysis of tetrahedron-tetrahedron interactions, combining scalable dataset generation with an efficient neural model. At its core is TetrahedronPairDatasetV1, a curated collection of one million labeled tetrahedron pairs with ground truth intersection status and volumes, filling a longstanding gap in geometry learning.

Built on this dataset, we present TetrahedronPairNet, a neural architecture that adapts PointNet and DeepSets for processing tetrahedron pairs. The model simultaneously predicts intersection classification and intersection volume, achieving real-time performance: over 98% classification accuracy and a mean absolute error of ≈ 0.0012 in volume estimation ($R^2 = 0.68$). It processes over 30,000 samples per second with full preprocessing—orders of magnitude faster than classical algorithms.

Unlike traditional symbolic approaches, TetrahedronPairNet is robust to degenerate configurations and requires no handcrafted geometry logic. Its fully batched, differentiable design supports seamless integration into simulation pipelines, CAD tools, and learning-based physics engines.

This work reframes geometric intersection as a data-driven inference task, laying the foundation for scalable, real-time, and intelligent geometry processing across computational design, simulation, AR/VR, and scientific computing.

Keywords: Point Cloud Analysis, Multi-Layer Perceptron, 3D Object Interaction Modeling, Geometric Deep Learning, Narrow Phase Collision Detection, Predicate Powered Learning

Resumo

Esta dissertação propõe uma nova abordagem para a análise rápida e baseada em aprendizagem de interações entre pares de tetraedros, aliando geração escalável de dados a uma arquitetura neural eficiente. No centro deste trabalho encontra-se o TetrahedronPairDatasetV1, um conjunto de dados cuidadosamente construído com um milhão de pares de tetraedros rotulados, contendo informações de interseção e volumes. Este dataset vem colmatar uma lacuna histórica na area de processamento geométrico.

Com base neste dataset, desenvolvemos o TetrahedronPairNet, uma arquitetura neural que adapta conceitos do PointNet e do DeepSets para processar tetrahedros. O modelo prevê simultaneamente a existência de interseção e o volume correspondente, alcançando uma precisão superior a 98% na classificação e um erro absoluto médio de aproximadamente 0.0012 na estimativa de volume ($R^2 = 0.68$). É capaz de processar mais de 30,000 amostras por segundo, superando de forma significativa os métodos algorítmicos tradicionais.

Diferentemente das abordagens simbólicas clássicas, o TetrahedronPairNet é robusto a configurações degeneradas e não depende de lógica geométrica artesanal. A sua estrutura totalmente batched e diferenciável permite a integração direta em pipelines de simulação, ferramentas CAD e motores físicos baseados em aprendizagem.

Esta investigação reconceptualiza a interseção geométrica como uma tarefa de inferência baseada em dados, estabelecendo as bases para uma nova geração de algoritmos geométricos — escaláveis, em tempo real e dotados de inteligência adaptativa. Para além do modelo e do dataset, a dissertação apresenta contribuições metodológicas, incluindo um gerador parametrizável de dados (*TetrahedronPairGenerator*), uma pipeline modular de aprendizagem automática (*TetrahedronPairML*) e uma análise aprofundada do desempenho e da generalização do modelo.

Os resultados experimentais validam o modelo em múltiplos regimes de dados, métricas de classificação e regressão, bem como escalabilidade computacional. As aplicações abrangem desde simulações físicas interativas, sistemas CAD/CAM em tempo real e experiências imersivas em AR/VR, até à computação científica e robótica autónoma, onde a necessidade de raciocínio espacial em tempo real é crítica.

Ao substituir lógica simbólica por inferência aprendida, esta dissertação contribui para um novo paradigma em geometria computacional, onde algoritmos são treinados — não programados — e onde a inteligência geométrica é uma capacidade emergente, pronta para ser explorada em escala industrial e científica.

Palavras-chave: Point Cloud Analysis, Multi-Layer Perceptron, 3D Object Interaction Modeling, Geometric Deep Learning, Narrow Phase Collision Detection, Predicate Powered Learning

Acknowledgement

First and above all, I want to thank you—for the perseverance, dedication, and genuine passion that carried you through this journey. For wanting to learn, for learning how to learn, and for choosing to embrace the process, even when it meant facing failure and the feeling of inadequacy. Thank you.

To my friends and family, whose unwavering support was indispensable throughout this journey—Alberta, Olavio, Neele, Elmer, Alda, Altino, and Edmira—thank you for all your support and your care. You are a great source of strength to me.

In the academic realm, I want to thank, first and foremost, Professor Gabriel Zachmann—for everything, truly. For introducing me to a topic that turned out to be one of the most fascinating questions I've had the pleasure to study. For making me even more passionate about the field of computer graphics. For your patience and understanding with my struggles with deadlines, and for allowing me to take the time I needed to write this thesis properly. Your guidance opened doors to discoveries I never imagined possible.

To Navid—for your brilliant insights that constantly pushed my thinking in new and exciting directions. To Thomas—for generously sharing your precious time and expertise when I needed it most. To Professors Carlos and Jorge—for the conversations, perspectives, and guidance that shaped my thought process.

To the entire CGVR lab at Bremen University—thank you. I truly wouldn't have been able to learn and grow as much without your collaborative spirit and intellectual support. You created an environment where curiosity could flourish.

To my sister Edmira—for the amazing conversations about statistical learning that clarified my thinking and reminded me why I fell in love with mathematics in the first place.

And most of all, to Neele—my girlfriend and closest collaborator. Not only for your tireless reviews and deep, thoughtful conversations about this work, but for your unwavering support through it all. Your patience, encouragement, love, and belief—in this project and in me—made all the difference.

Gratitude is the feeling that overflows from me right now, and it belongs to each of you.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Context and Motivation

"I think the universe is pure geometry — basically, a beautiful shape twisting around and dancing over space-time." - Antony Garrett Lisi

For millennia, geometry, the study of shapes and spatial relations, has been a pivotal force in shaping how humanity conceptualizes and addresses challenges. From the Pyramids of Giza, where ancient architects applied geometric principles for remarkable precision [1], to the present-day advancements in medical imaging technology employing geometric algorithms [2],it is an understatement to declare geometry as less than omnipresent.

As time progressed, the study of geometry underwent transformative phases that significantly expanded its scope and applicability. In its early stages, geometric shapes were rudimentarily sketched in sand, providing insights into their inherent properties. However, the study was initially limited, focusing primarily on two fundamental measurements: angle and distance [1]. A pivotal shift occurred with the introduction of coordinates to represent points in space. This innovation enabled geometric shapes to be more easily described algebraically, marking a profound convergence of algebra and geometry, transforming the way mathematicians approached and understood geometric concepts [3]. In the mid-20th century, as computing power surged, computers were more deeply integrated into the realm of mathematics. A rising contingent of mathematicians began employing them to create examples, scrutinize theorems, and even construct proof [3]. It was during this period that the field of computational geometry emerged, focusing on developing algorithms and data structures for solving problems involving geometric shapes and structures [4].

Simultaneously, as these advancements unfolded, the field of machine learning (ML) grew [5]. ML focuses on developing computer systems that can learn and adapt by leveraging data, without the need for explicit instructions. [6]. Neural network-based machine learning, in particular, excelled at approximating functions in very high dimensions with remarkable efficiency and accuracy. The success of these methods led to an increasing trend in applying them to a broad spectrum of problems including computational mathematics [7]. Despite these achievements, computational geometry and algebra remain fields in which machine learning has not been integrated to its full potential [8]. For instance, calculating the volume of high-dimensional geometric shapes is computationally expensive, and randomized approaches are often used in practical applications [8]. Integrating machine learning into such problems could leverage data-driven insights to enhance efficiency and accuracy.

A notable example of such geometric problems is finding the intersection of polytopes—geometric figures defined by their vertices, edges, and facets. This problem is fundamental in addressing real-world issues across diverse fields. For example, in logistics and supply chain management, the intersection of polytopes can model feasible solutions for optimizing transportation routes. This includes considering important factors such as time constraints, the ability to reject customers,

and vehicle capacity [9]. In finance, particularly in portfolio optimization, polytope intersections can model critical strategies by capturing constraints involved in selecting asset combinations that maximize returns while minimizing risk. This ensures compliance with requirements on asset allocation, expected returns, and risk tolerance [10]. Conservation planning also benefits from polytope intersections, as they represent spatial configurations that adhere to ecological constraints. This facilitates the design of effective conservation strategies [11]. In game theory, polytope intersections are employed to model feasible strategies for players, considering constraints and their objectives. This is especially relevant in scenarios involving multi-agent systems and strategic decision-making because they provide a geometric framework to model and analyze the common feasible strategies. This allows for a concise representation of shared decision spaces, helping to understand and predict outcomes in complex strategic interactions [12]. In applications like virtual prototyping for automotive crashes, for instance, this enhances safety design, and substantially reduces the need for physical prototypes, which can be costly and resource-consuming to operate [13], ultimately expediting the development of safer vehicles and contributing to advancements in automotive engineering and safety standards. As a direct consequence, this could lead to a noticeable reduction in both the frequency and severity of automotive accidents.

As we can see, in a broader sense, when faced with a set of linear constraints [1] that must satisfy specific criteria, finding an efficient solution to this problem can be immensely valuable.

The specific problem of tetrahedron-tetrahedron intersection and volume computation is common in the context of representing and processing 3D geometric objects [14]. In the field of computer graphics, a mesh is a representation of a three-dimensional object or surface composed of vertices, edges, and faces. Meshes provide a way to represent and model complex 3D shapes [15]. Tetrahedral meshes consist of tetrahedra, or triangular pyramids, figures with four triangular faces, four vertices, and six edges each (see figure 1.1). They serve as efficient representations for arbitrarily complex volumetric objects in 3D space [16]. They are widely used in simulations, collision detection algorithms, and various other computational tasks [17]. In scenarios involving multiple intersecting tetrahedra, as seen in collision detection algorithms, it is essential to calculate precise intersections in order to accurately model and predict object behavior in simulated environments. Knowledge of the volume of the intersection enables simulation systems to apply appropriate forces or constraints to objects, ensuring realistic and physically plausible responses to collisions [18].

As ML algorithms reshape the way we tackle geometric problems [19], this work aims to go beyond mere model development, and aspires to propose a framework for tackling these problems through the lens of ML. In doing so, it seeks to expand the toolkit for solving such problems, uncovering new connections and approaches. The outcomes of this thesis could lead to advancements that directly or indirectly benefit industries relying on geometric computations, such as logistics, finance, game theory, simulations, and other computational tasks. Yet, beyond any utilitarian application, there is a deep personal satisfaction in uncovering the elegant intricacies within geometry.

[1] Linear constraints are mathematical expressions (typically inequalities like $a_1x_1 + \cdots + a_nx_n \leq b$ or equations) that define boundaries in space. The solution space satisfying all constraints forms a polyhedron (bounded polytope).

1.2 Problem statement

This thesis develops a neural network-based model to predict the intersection of two tetrahedra and estimate the intersection volume within a unit cube. The problem is divided into two tasks:

1. Intersection Prediction: A **binary classification problem**[2] where the model determines if two tetrahedra intersect (*yes/no*).
2. Intersection Volume Estimation: A **regression problem**[3] where the model estimates the volume of the intersection.

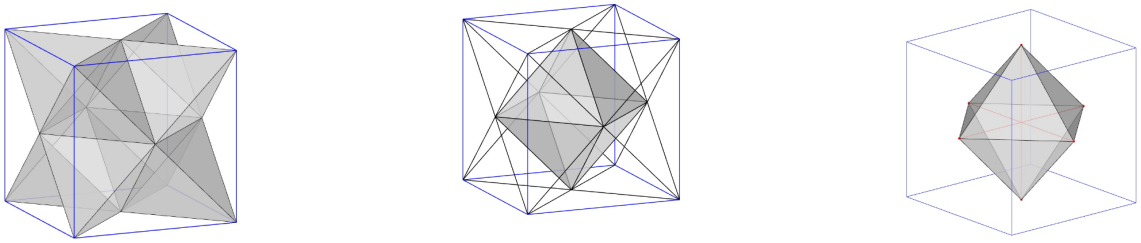

Figure 1.1: Intersection of tetrahedra. Stella octangula, a compound formed by the intersection of two equal tetrahedra, with the same centroid and opposite orientations. *Source*:[20].

[2]Binary Classification Problem: A problem where the output has only two possible values
[3]Regression Problem: A problem where the output is a continuous value.

1.3 Research Questions and Objectives

The following Research Questions (RQs) guide this work:

- **RQ1** - How can a dataset of tetrahedra pairs be generated to train a neural network for intersection prediction and volume estimation?
- **RQ2** - Which neural network models are most suitable for accurately determining if two tetrahedra intersect and the volume of the intersection?
- **RQ3** - How does the performance of neural networks compare to traditional methods for tetrahedron-tetrahedron intersection and volume computation?

The objectives (OB) of this thesis are defined as follows:

- **OB1** - Analyze current methods and algorithms for tetrahedron-tetrahedron intersection detection and volume computation.
- **OB2** - Create a dataset of tetrahedra pairs with labeled intersection status and intersection volume, ensuring diverse and representative configurations.
- **OB3** - Design and implement a neural network model for predicting whether two tetrahedra intersect and estimating the intersection volume of two tetrahedra.
- **OB4** - Compare the proposed models with state-of-the-art methods in terms of accuracy, efficiency, and scalability.

1.4 Scientific Contributions

The work presented in this thesis introduces several technical and scientific contributions to the intersection of computational geometry and machine learning:

- **Creation of a Diverse 1M-Scale Dataset:** A large-scale dataset comprising one million tetrahedron pairs was generated, covering a wide range of intersection types, spatial orientations, volumes, and other configurations (read 3.2.2, 5.12).
- **Parametrizable Dataset Generator:** A high-performance C++ command-line tool was developed to generate millions of samples with configurable distributions of intersection types, bounding volumes, and output formats. The generator has been used to produce over 20 million samples (read 4.1).
- **Configurable Machine Learning Pipeline:** An end-to-end, modular, and parametrizable machine learning pipeline was designed for large-scale geometric reasoning tasks. It includes support for logging, data filtering, normalization, and integration with custom loss functions and evaluation metrics (read 4.2).
- **MLP and TetrahedronPairNet Architectures:** Two distinct architectures were explored: a baseline multi-layer perceptron (MLP) trained without explicit invariance handling, and *TetrahedronPairNet*, a dedicated neural model for processing point clouds of tetrahedron pairs. Both performing joint classification (intersection prediction) and regression (volume estimation) with compact architecture and high throughput (read **??**).
- **Python and C++ Inference Wrappers:** Lightweight inference wrappers were implemented in both Python and C++ to facilitate deployment and integration into simulation environments. These interfaces handle data normalization, output decoding, and aim to replicate real-world performance conditions.

- **Scientific Availability:** All code, data, models, and pretrained weights are intended to be made publicly available to the scientific community, subject to licensing and publication constraints.

The purpose and function of each of these components will be further clarified and contextualized in the following sections.

1.5 Document Structure

This document is organized into seven chapters, each building upon the previous to form a coherent narrative addressing the research problem, methodology, and outcomes.

Chapter 2 – Ethical Considerations begins with a critical examination of the ethical implications surrounding mathematical and algorithmic research. It explores how theoretical contributions are influenced by—and impact—social, cultural, and historical contexts, emphasizing the need to recognize implicit biases and societal consequences.

Chapter 3 – Research introduces core concepts relevant to the thesis, starting with a brief primer on geometric learning and an analysis of desirable data characteristics. It then surveys the state of the art in 3D point cloud neural networks, highlights suitable architectural choices for this problem space, and presents a comparative analysis of existing datasets and evaluation metrics.

Chapter 4 – Development details the technical contributions of the thesis, including a high-performance, parametrizable data generator for producing tetrahedron pair samples (*TetrahedronPairGenerator*) and a scalable, modular machine learning pipeline designed for geometric reasoning on large datasets (*TetrahedronPairML*).

Chapter 5 – Experiments outlines the experimental design, training procedures, hardware configurations, and other essential settings for reproducibility. It provides a detailed analysis of model performance under various data regimes and architectural configurations, and also explores a newly created dataset for 3D point cloud understanding in tetrahedra pairs (*TetrahedronPairDataset*).

Chapter 6 – Evaluation discusses evaluation results using multiple metrics, focusing on overall accuracy, mean accuracy, and AUC for classification, as well as MAE, MAPE, kappa value, and bin accuracy for volume estimation, computational throughput, and generalization capacity.

Chapter 7 – Conclusion summarizes key contributions, discusses limitations, and proposes directions for future research. Broader implications and open questions are also addressed.

Chapter 2

Ethical Considerations and Personal Motivations

Mathematics is often perceived as neutral, detached from ethical considerations [21]. However, this view overlooks how deeply mathematical developments are intertwined with society. Ideas do not emerge from the ether; they are shaped by human thought, societal needs, cultural influences, and historical contexts [22]. Likewise, mathematical theories, algorithms, and models extend beyond academic journals, they impact technology, science, social policies, and economic strategies [23]. These influences carry biases, some obvious, others subtle and hard to detect. The effects of these biases can ripple through society, affecting individuals and communities in unpredictable ways [24–27]. Therefore, despite my research tackling a fundamentally theoretical problem, it is crucial to understand it within its broader context, considering both its potential consequences and the ethical responsibilities involved. In this chapter, I will discuss the context of this thesis, reflect on my own biases, and explore the potential applications of my work.

2.1 Personal Motivation and Research Context

This research stems from my interest in geometry and computer graphics and my desire to explore how machine learning can be integrated within these fields. Before my Erasmus exchange at Bremen University, I contacted the Computer Graphics and Virtual Reality (CGVR) Labs to discuss potential topics and co-supervision. They recommended focusing on "tetrahedron-tetrahedron intersection and volume computation using ML", a topic that aligns with both my interests and the lab's research agenda. This problem is a common challenge in the field, detecting tetrahedra intersections and calculating the volume of these intersections are crucial for ensuring accurate simulations.

The expertise of the CGVR lab was crucial in shaping this research. During my time at Bremen University, their specialists in machine learning and computer graphics provided essential support for refining and debugging my algorithms and offered computational resources that significantly influenced my approach. This thesis would not have been possible without their support and contributes to the lab's ongoing collision detection projects.

2.2 Potential Use Case: Collision Detection

As explored in opening section of this chapter, even the most abstract problems in mathematics can eventually find their way into the real world—sometimes with surprising consequences. This chapter takes one such path and follows it to a practical example: using machine learning to detect collisions between 3D shapes.

The Technical Setup

Imagine a digital scene made of tiny blocks—specifically, tetrahedra. In many fields like robotics, virtual reality, and physics simulations, these tetrahedra are used to build complex objects and environments [14, 17]. But just like in the physical world, we need to make sure that things don't crash into each other when they're not supposed to. That is where **collision detection** comes in.

In general, this problem is split into two phases [28]:

1. A **broad phase**, which quickly filters out pairs of shapes that are clearly far apart and can't intersect.
2. A **narrow phase**, which takes the remaining pairs and performs detailed checks to see if they actually touch or overlap.

Our model is designed for the second part—the narrow phase. It doesn't try to understand the whole scene. Instead, it focuses on a much simpler question: "Given two tetrahedra, do they intersect? And if so, by how much?" This makes the problem both easier to train and faster to compute.

Why Machine Learning?

Traditional collision detection algorithms are rule-based and exact (see more in 3.1,3.1.3 and 3.1.4). But they are also slow, especially when applied to millions of pairs. Machine learning offers a different approach: instead of following hard-coded rules, the model learns patterns from many examples. The trade-off is speed versus accuracy.

This works well most of the time. But not all the time.

When It Fails

There are two ways a machine learning model can make mistakes:

1. It gets bad input—maybe from noisy sensors, missing data, or even deliberately corrupted data.
2. It gets good input, but still makes the wrong call because of its training limitations.

Unlike a hand-written algorithm, an ML model doesn't "know" geometry in the classical sense. It has seen many examples and learned a kind of fuzzy intuition. That is powerful, but also fragile. And when we're talking about systems that operate in the real world, even a small mistake can have big consequences.

Real-World Consequences

Lets take a few examples:

- Autonomous Vehicles: A collision detection error could cause a robot or car to misjudge its environment. That might lead to a minor bump—or something much worse.
- Industrial Robots: If a robot arm moves into an area it thought was clear, it might crash into equipment or even a person. That's a safety risk and an operational failure.
- Virtual Reality: Here the stakes are lower, but still meaningful. Poor collision detection can make virtual objects pass through each other, which ruins immersion and realism. In training simulations, that can undermine the whole learning experience.

In all these situations, a failure by the model breaks something more than just geometry—it breaks trust.

Thinking Ethically

This is why ethics and safety are not just afterthoughts—they are built into the core of the design. When we choose machine learning over rule-based methods, we're not just optimizing for speed. We're also taking on a new kind of responsibility.

Because now, our system's behavior depends not only on math, but on the data we feed it, the decisions we make about model architecture, and how well we have tested it in difficult scenarios.

Mitigating the Risks

To reduce the chance of failure, several strategies are important:

- **Data Validation:** Catch mistakes before the model even sees them.
- **Robustness Testing:** Train and test the model on edge cases—rare shapes, unusual overlaps, strange orientations.
- **Explainability:** Use tools that help us understand why the model made a certain prediction.
- **Uncertainty Estimation:** Let the model tell us when it is unsure, so we can step in.
- **Fallback Systems:** If things get too uncertain, switch back to classical algorithms or ask a human to check.

These strategies are part of a larger philosophy: design systems that fail gracefully. If we accept that mistakes will happen, then our job is to make sure those mistakes are caught, understood, and corrected—before they become dangerous.

By building these safeguards into our models, we bridge the gap between clever code and responsible deployment. It is not just about making something that works. It is about making something we can trust.

2.3 Transparency

Transparency is foundational to trustworthy research. It supports accuracy, reproducibility, and credibility by allowing others to trace key decisions, understand methods, and critique conclusions. In this thesis, transparency has served both as a scientific principle and an ethical stance.

Throughout this work, I have aimed to make influences visible—how prior research shaped the approach, why specific models or methods were chosen, and where trade-offs were made. Transparent documentation also makes it easier to identify hidden biases, design flaws, or oversights—many of which are inevitable, especially under constraints like time, expertise, or computational resources.

Research Documentation

To clarify the role of each source, references have been grouped in the appendix into three categories:

- **Checked:** Briefly consulted to extract general ideas or support minor points.
- **Analyzed:** Closely read for key insights, directly informing this work's arguments and methods.

- **Foundational:** Studied in depth, with repeated engagement. These shaped the core framing of the research.

Additional metadata describes each source's relevance (e.g., *Intersection Prediction*, *Volume Estimation*) and thematic focus (e.g., *Machine Learning*, *Algorithmic*, or *Argumentative*). This structure helps readers quickly locate the conceptual building blocks of the thesis.

Source Access

Several key papers were paywalled. To access them, I used Sci-Hub. While I recognize the tension this creates around intellectual property, the broader ethical principle—open access to scientific knowledge—remains central. In many regions and institutions, the barrier isn't interest or ability, but economics. Science should be accessible, not exclusive.

Open Source

All source code developed in this research will be released under an open-source license. This enables others to verify results, test extensions, or repurpose the methods in different contexts.

Accessibility and Plurality

This thesis is written with a wide audience in mind: researchers, practitioners, and curious learners alike. Where possible, I have prioritized clarity—using plain language, adding footnotes for less familiar concepts, and including supplementary explanations in the appendix. To further support accessibility, I also asked a proofreader outside the field to review the text and provide feedback on its readability.

Still, accessibility is not universal. Cultural context, language, background knowledge, and personal interest all shape how research is understood. English remains the dominant language of academic science, and with that comes a form of exclusion. On a personal note, I am reminded that someone like my own mother will likely never read this thesis—not due to a lack of curiosity, but because the language itself forms a barrier.

True accessibility goes beyond simplification it requires inclusivity. That means actively seeking feedback from diverse readers and being mindful of the perspectives that may be missing. This work is an invitation to dialogue, and I welcome engagement from voices across backgrounds, disciplines, and experiences.

Environmental Considerations

One overlooked ethical dimension of this research is energy use. Model training and dataset generation can consume substantial resources—especially when conducted without a strategy. Machine learning currently lacks clear standards for environmental efficiency, and experimentation often leads to waste.

In one case, I trained a model for nearly two weeks, chasing marginal gains over a shorter 3-day baseline. Early loss metrics already suggested diminishing returns, but I ignored them. That was a mistake—wasteful and unnecessary.

Although this research is small-scale compared to industry models, the same principles apply. We can and should be more deliberate.

Suggested Guidelines for Sustainable Research

- **Review First:** Before running experiments, survey the field. Prioritize promising baselines over brute-force exploration.
- **Use Early Stopping:** Monitor learning dynamics and terminate underperforming runs.
- **Track Energy Use:** Tools like *CarbonTracker* or *PowerMeter* help quantify and reduce emissions.
- **Favor Lean Models:** Use smaller architectures, distillation, pruning, and quantization where possible.
- **Optimize Data Generation:** Avoid redundancy. Focus on diverse, representative samples and estimate the minimum data needed to reach stable performance.
- **Collaborate:** Share pretrained models, datasets, and infrastructure to reduce duplicated effort.

Small changes at the individual level, if adopted broadly, can shift the culture of ML research toward greater sustainability.

2.4 Ethical Assessment

Ethical Intentionality

This thesis is grounded in a simple belief: that research should aim to reduce harm and expand human well-being. Every equation, model, and design decision must be interpreted not only in technical terms, but also in ethical ones.

Ethical Vigilance

Intent matters, but it is not enough. Well-meaning technologies can still cause harm when misapplied. Several risks deserve explicit attention:

- **Dual Use:** Algorithms developed for benign geometric tasks could, in other contexts, be repurposed for surveillance, automated targeting, or other harmful applications. This isn't a hypothetical concern, it is an active risk. Researchers and policymakers must proactively define and restrict such uses. For example, initiatives like the *Stop Killer Robots* campaign (`https://www.stopkillerrobots.org/`) advocate for preemptive regulation of autonomous weapons and other dual-use systems.
- **Over-Reliance:** There's a growing temptation to treat model outputs as authoritative. Especially in safety-critical domains, this is dangerous. Machine learning models are tools—they should support, not replace, human judgment. Over-reliance erodes accountability and hides uncertainty.

Being ethically vigilant means anticipating misuse, not just reacting to it. It requires designing systems with failure in mind, encouraging critical interpretation of outputs, and promoting policies that align technical progress with societal responsibility.

Mitigation Through Process

This work includes several built-in safeguards:

- **Transparency:** Methods, limitations, and failure modes are documented. Users should understand where the model can go wrong.

- **Iterative Evaluation:** Ethical concerns evolve. Regular feedback and post-deployment review are essential.
- **Interdisciplinary Dialogue:** Ethics isn't a solo discipline. Input from domain experts, ethicists, and impacted communities improves foresight.

Responsible Innovation

The success of research should not be measured only by technical performance, but by its real-world implications. This project represents a small step toward more intelligent, adaptable geometric models—but the deeper goal is to build tools that serve people and protect values.

In the end, this work is not just about detecting shape intersections. It's about navigating the space where technology meets ethics—and doing so with clarity, humility, and intent.

Chapter 3

Research

3.1 Premier

This section serves as a conceptual entry point. It doesn't aim to replace mathematical rigor but offers an intuitive guideline for the geometric objects at the center of this work.

3.1.1 What is a Tetrahedron?

Take four triangles. Connect them edge to edge so that each face touches the others, and each corner joins exactly three faces. The result is a tetrahedron—the simplest possible 3D shape. It has four triangular faces, six edges, and four vertices (see Figure 3.1). Unlike pyramids with polygonal bases,[1] a tetrahedron is made entirely of triangles [29]. There is no "base"—all faces are equivalent. In its most symmetric form, a regular tetrahedron, all faces are equilateral, and all edges are the same length. The name comes from the Greek tetra- (four) and hedra (face or base),[30] literally: "four-faced."

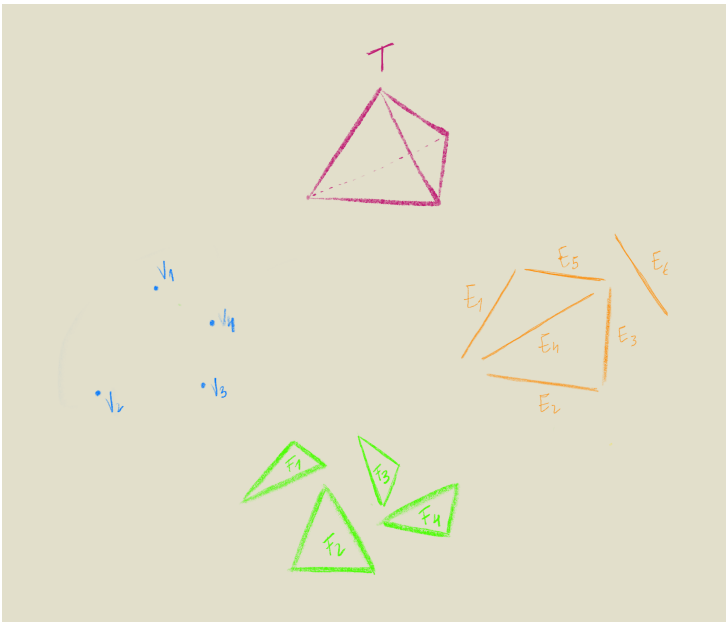

Figure 3.1: A tetrahedron showing its fundamental geometric properties: four triangular faces, six edges, and four vertices

Simple—but not trivial. The tetrahedron's minimalism is its strength. It appears in nature, physics, chemistry, and computation 1, precisely because its structure is both efficient and robust.

[1] A polygonal base refers to the flat bottom face of a 3D shape, typically a polygon like a square, pentagon, or hexagon. For example, a square pyramid has a square base and triangular sides.

3.1.2 Why Care About Tetrahedrons?

In geometry, each new dimension requires one additional point to define a shape that fully escapes the previous one's limitations.[31] These minimal sets of points, when connected, form what are known as *simplices*: the most elementary building blocks of that space (see Figure 3.2).

- **0D:** A single point—presence without extent
- **1D:** Two points—defining a line, but still confined
- **2D:** Three points not in the same line—defining a triangle and enclosing area
- **3D:** Four points not in the same plane—defining a tetrahedron and enclosing volume

Each simplex marks the minimum configuration required to span its dimension. In this sense, the tetrahedron is to 3D space what the triangle is to 2D: the smallest, most rigid, and most stable volumetric structure.

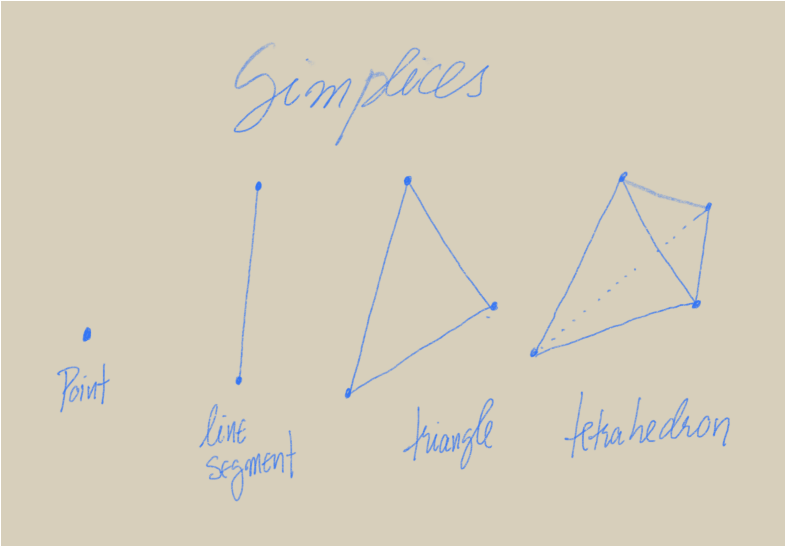

Figure 3.2: The progression of simplices across dimensions: from a single point (0D) to a line segment (1D), triangle (2D), and tetrahedron (3D). Each represents the minimal geometric structure that can fully span its respective dimensional space, with the tetrahedron being the fundamental building block of three-dimensional geometry.

This dimensional progression demonstrates the tetrahedron's fundamental role:

Dimension	Points	Simplex	Defines
0D	1	Point	Position
1D	2	Line	Distance
2D	3	Triangle	Area
3D	4	Tetrahedron	Volume

The tetrahedron thus emerges not by design, but by necessity. Its simplicity makes it universal. It is the canonical container of three-dimensional space.

3.1.3 Why Tetrahedron–Tetrahedron Intersection Status?

Tetrahedrons are the simplest shapes that enclose volume in three-dimensional space[29]. Because any complex 3D object can be decomposed into a collection of tetrahedrons, determining whether

two arbitrary solids intersect can be reduced to checking pairwise intersections between their constituent tetrahedrons[17].

This makes tetrahedron–tetrahedron intersection a foundational problem in computational geometry. If we can reliably determine whether two tetrahedrons intersect, we can scale that solution to possibly handle any polyhedral geometry.

Yet despite the tetrahedron's simplicity, the intersection problem is surprisingly intricate (3.2.2). Two tetrahedrons can interact in many ways: full containment, partial overlap, face-to-face alignment, edge contact, or touching at a single vertex[29, 32, 33]. A robust intersection algorithm must detect all these configurations accurately.

The primary challenge lies in numerical precision and algorithmic efficiency [34]. Floating-point arithmetic,[2] though fast, introduces rounding errors that can undermine geometric tests. These errors become especially problematic in degenerate or near-degenerate cases—when tetrahedrons are nearly coplanar, barely touching, or aligned along nearly parallel planes. Under such conditions, even minuscule numerical deviations can lead to large logical errors: overlaps may be missed, or false positives may occur[34].

Some modern algorithms mitigate these issues by postponing floating-point operations until the final stage. They evaluate geometric predicates—such as orientation and in-sphere tests—using exact arithmetic, and convert to approximate representations only when necessary.

A robust intersection test must therefore satisfy four critical criteria:

- **Edge-Case Handling:** Detect degenerate configurations like shared vertices or coplanar faces.
- **Numerical Stability:** Remain reliable despite rounding errors and representational noise.
- **Efficiency:** Scale to real-time or large-scale scenarios involving millions of checks.
- **Generality:** Support arbitrary positions, orientations, and aspect ratios.

The following table reviews some classical intersection algorithms, comparing their trade-offs across robustness, performance, and implementation complexity.

3.1.4 Why Compute the Volume of Intersection?

Determining whether two tetrahedrons intersect is useful but in many applications, a simple yes-or-no answer is not enough. What often matters is *how much* they intersect. This is where computing the volume of intersection becomes essential.

Whether in physics simulations, material science, or mesh-based modeling, the quantity of overlap influences outcomes such as force computations, structural analysis, and spatial occupancy [18].

Unlike regular shapes, whose volumes can be calculated with closed-form formulas, the intersection of two tetrahedrons forms an irregular convex polyhedron. Its structure depends entirely on how the tetrahedrons intersect. There is no general formula; the volume must either be explicitly constructed or estimated [38].

Two main strategies are commonly used:

[2] Floating-point arithmetic is a method of representing real numbers in computers using a finite number of bits. While efficient, it introduces rounding errors due to limited precision, which can lead to inconsistencies in geometric computations.

Method	Summary
GJK Algorithm	Iteratively builds simplices within the Minkowski difference [1]of two shapes. Early termination occurs if a simplex encloses the origin, indicating intersection. Efficient for convex polytopes but sensitive to numerical precision near boundaries [35].
SAT Algorithm	Tests potential separating axes derived from face normals and edge cross products. If no axis separates projected intervals, intersection occurs. [36] optimized this for tetrahedrons by reusing face-query results, avoiding exhaustive edge-pair checks [37].
McCoid-Gander Algorithm	Transforms one tetrahedron into a canonical reference frame, reducing 3D intersection checks to 2D projections. Tests vertex containment via barycentric coordinates and edge-face intersections, prioritizing robustness over speed [33].

[1] The Minkowski difference of sets A and B is $A - B = \{a - b \mid a \in A, b \in B\}$. If it contains the origin, A and B intersect.

Table 3.1: Classical methods for tetrahedron–tetrahedron intersection testing.

- **Exact geometric methods** compute the precise shape of the intersecting region. The resulting polyhedron is then decomposed into smaller tetrahedra or simpler components, and their volumes are summed to yield the total intersection volume.
- **Approximate methods**, such as Monte Carlo integration, avoid reconstructing the intersection geometry altogether. Instead, they estimate the volume probabilistically by randomly sampling points within a bounding region and testing for containment.

Both approaches come with challenges. Exact methods require robust geometric predicates, careful handling of degenerate cases, and efficient triangulation schemes. Monte Carlo approaches must balance sampling density, convergence rate, and computational cost to achieve acceptable accuracy. Computing intersection volume transforms geometric contact into a measurable quantity—bridging the gap between shape and interaction. It is a critical capability for turning geometry into simulation.

The table below summarizes classical and modern strategies used to approach this task.

Method	Principle and Tradeoffs
Nef Polyhedra	Represents tetrahedrons as boolean combinations of 3D halfspaces. Applies logical operators (AND) to compute intersections, then decomposes the result into sub-volumes. Rigorous but computationally intensive [39].
Monte Carlo	Estimates volume via random sampling: generates points within a bounding region and scales the fraction inside both tetrahedrons by the total volume. Efficient but trades precision for speed [38, 40].

Table 3.2: Methods for intersection volume computation. Nef polyhedra provide exact results; Monte Carlo emphasizes scalability.

3.1.5 The Problem

As stated previously in 1.2, given two arbitrary tetrahedrons within a unit cube, we aim to address two fundamental tasks: **(1)** determine whether they intersect, and **(2)** compute the volume of their intersection using a machine learning model.

Traditionally, both tasks are addressed through explicit geometric computation, specifically, by constructing the polyhedron formed by the intersection of the two tetrahedra (see Table 4.3). If such a polyhedron exists, the tetrahedra intersect, and its volume can be computed analytically [39]. Conversely, if no valid intersection shape is found, the volume is zero.

In other words, the geometric computation of intersection volume implicitly encodes both intersection status and overlap quantity. However, this process becomes computationally expensive at scale.[3]

Instead of relying on explicit geometric methods, We propose a data-driven approach: approximating this computation using a neural network trained on sampled pairs of tetrahedrons. The network is trained to map the 3D vertex coordinates of two tetrahedrons directly to:

- A binary label indicating intersection status (classification), and
- A continuous value representing the volume of intersection (regression).

This approach bypasses the need for polyhedron construction entirely. By learning the latent geometric patterns that govern intersection behavior, the model provides fast and scalable predictions. While this comes at the cost of analytical precision, it significantly accelerates inference, crucial for applications requiring real-time collision checks or high-throughput geometry processing.

What is Machine Learning?

Machine learning refers to the design of computational systems that autonomously solve tasks by transforming data into structured knowledge through experience [41, 42]. Crucially, "learning" in this context does not imply memorizing training data, but rather discovering generalizable patterns that apply to previously unseen instances [42]. The performance of a machine learning model is fundamentally determined by the quality and quantity of the data it learns from, which serves as ground truth [6, 41, 42].

In this work, exact geometric algorithms are used to compute ground-truth solutions for arbitrary pairs of tetrahedrons, enabling the creation of large, high-quality labeled datasets. This makes supervised learning, a paradigm that maps inputs to known outputs, a natural choice, with sufficient labeled data, supervised models can achieve high predictive accuracy [41, 43].

The primary class of models employed in this thesis are Neural Networks, a computational framework composed of layers of interconnected artificial neurons. Neural Networks are particularly effective at modeling complex, non-linear relationships in high-dimensional data [41, 44]. They offer advantages in scalability, expressive power, and training efficiency as we are going to explore in future sections.

Although the focus here is on Neural Networks, the techniques and analytical frameworks developed are model-agnostic and provide a solid foundation for future exploration using alternative machine learning methods.

[3] Imagine executing this loop-constructing intersection shapes—merely to determine intersection status and compute volume for millions of samples within seconds.

Supervised Learning Fundamentals

Supervised learning aims to find a mapping function that relates inputs to outputs using labeled data [41, 43–45]. The *input space* represents all features describing each data point—in this case, features like the coordinates of the vertices of two tetrahedra [42]. The *output space* contains the target values, such as whether the tetrahedra intersect (a binary outcome) or the volume of their intersection (a continuous value between 0 and $\frac{1}{6}$[4]). The goal is to identify a function that minimizes the difference between the model's predictions and the true labels [41, 42]. But how do we discover such a function? Here are the *fundamental ingredients*

Dataset
The dataset is the foundation for learning. It consists of input-output pairs that capture the relationship we want the model to learn. Its quality, size, and diversity are critical for generalization—ensuring the model performs well not just on training data but also on unseen examples [41].

Loss Function
To guide learning, we need a way to quantify prediction errors. A loss function measures how far predictions deviate from actual outcomes, providing feedback that the learning algorithm uses to improve model accuracy [41].

Hypothesis Class
The hypothesis class is the set of candidate functions considered to solve the problem. It encodes assumptions, or *inductive bias*, about the underlying relationship. Choosing an appropriate hypothesis class is crucial: if the true function lies outside this set, no amount of training will yield good results [41].

Learning Algorithm
The learning algorithm searches within the hypothesis class to minimize the loss function, effectively selecting the best function that represents the data's input-output mapping [41].

The learning process proceeds as follows:

1. Split the dataset into training and testing subsets.
2. The learning algorithm explores candidate hypotheses.
3. It evaluates predictions against actual outputs using the loss function.
4. Based on feedback, it iteratively refines its hypothesis until convergence.

Four key insights emerge from this framework:

- Data quality is paramount. Flawed or unrepresentative data undermines learning regardless of algorithmic sophistication.
- The hypothesis class must be well-chosen to contain good approximations of the true function.
- The loss function directs learning. Poorly designed losses can mislead optimization.

[4]The maximum volume of a tetrahedron contained within the unit cube $[0,1]^3$ is $\frac{1}{6}$. This is because the largest tetrahedron can be formed by choosing the three vertices along the edges from the origin to the points $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$, together with the origin $(0,0,0)$. The volume of a tetrahedron defined by points $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, and $\mathbf{d}$ is given by $\frac{1}{6}\left|\det\left(\mathbf{b}-\mathbf{a}, \mathbf{c}-\mathbf{a}, \mathbf{d}-\mathbf{a}\right)\right|$. Substituting these points, the determinant equals 1, so the volume is $\frac{1}{6}$. No larger tetrahedron can fit inside the unit cube because these points maximize the spatial extent along the cube's edges.

- The learning algorithm's efficiency and strategy determine how effectively it navigates the hypothesis space.

These components are tightly interconnected—weakness in any one can cripple the whole process, while strength in all enables powerful, accurate models.

Two types of errors characterize learning outcomes [41–43]:

- *Estimation Error:* The gap caused by the algorithm's failure to find the best hypothesis, often due to limited data or algorithmic constraints.
- *Approximation Error:* The inherent error when the hypothesis class cannot perfectly represent the true function, no matter how well trained.

The primary goal is to minimize both estimation and approximation errors, aiming not only for good performance on the training data but also for strong generalization to unseen data.

How can we evaluate the generalizability of a trained model? We use a **validation dataset**—a separate set of samples drawn from the same distribution as the training data. Performance on this validation set provides an estimate of how well the model will perform on new, unseen data. Strong validation results indicate good potential for generalization.

How can we be confident that strong validation performance implies generalization to other data? Probably Approximately Correct (PAC) learning theory [41, 46]offers a partial explanation. It states that for certain hypothesis classes, given sufficiently large training data, the model's error on new samples from the **same distribution** will likely be close to its training or validation error. This concept is quantified by *sample complexity*—the amount of data needed to guarantee low error with high confidence. Importantly, PAC guarantees apply only when new data follows the same distribution as the training data; they do not ensure success on fundamentally different (out-of-distribution) data.

While PAC bounds are often conservative compared to empirical requirements, they provide foundational theoretical guarantees. Recent work even demonstrates that complex neural networks can be PAC-learnable under certain conditions.

Statistical learning theory further shows that some infinite hypothesis classes are PAC-learnable. Successful learning depends on carefully choosing the hypothesis class and balancing *bias* versus *variance* to achieve *uniform convergence*—where minimizing loss on sufficiently large datasets leads to consistent performance on any data from that distribution [42].

This naturally raises deeper questions: *Why are some hypothesis classes PAC-learnable and others not? Which hypothesis classes are PAC-learnable? Is it possible to learn effectively without prior assumptions about the hypothesis class?*

These foundational questions lie beyond the current scope but are essential to a full understanding of machine learning.[5]

Next, we explore neural networks in more detail.

Neural Networks Fundamentals

Artificial Neural Networks, or simply Neural Networks, inspired by the brain's architecture, are computational models designed to recognize patterns and approximate complex functions [8, 41,

[5] In brief: hypothesis classes differ in richness—the number of meaningfully distinct solutions they can represent. Infinitely rich classes are not PAC-learnable. Only those with finite VC dimension are. And no, we cannot learn without assumptions—the No Free Lunch theorem guarantees this.

44, 47]. The core unit is the perceptron, which computes a weighted sum of inputs, adds a bias, and applies a non-linear activation to produce an output [44, 48].

Traditionally, perceptrons define linear decision boundaries[6] that partition the feature space by thresholding weighted inputs [49]. This allows classification by activating when inputs exceed the threshold.

Perceptrons also handle regression by replacing the threshold with continuous activations, outputting values across a range instead of binary classes. In simple linear regression, a perceptron's output is a weighted sum plus bias—essentially a linear function. Stacking perceptrons into multilayer networks extends this to model complex, highly non-linear relationships between inputs and continuous outputs [44].

Multilayer perceptrons (MLPs) with non-linear activations can approximate virtually any continuous function. Training minimizes a loss function—commonly Mean Squared Error (MSE)—to reduce the difference between predicted values and targets. This makes neural networks versatile for regression tasks, from predicting volumes or intersection areas to time-series forecasting.

Arranged in layers, MLPs transform inputs through successive nonlinear mappings, enabling them to approximate intricate decision surfaces. This power is key for challenging problems like geometric intersection detection [44].

More recently, neural networks are valued not just for decision boundaries but for representation learning—automatically extracting abstract features that capture complex, non-linear patterns [43]. For geometric tasks such as tetrahedron-tetrahedron intersection, this means learning spatial relationships between vertices and edges without manual feature engineering, improving both detection and volume estimation.

A network's parameters—weights and biases—define its hypothesis class, the set of functions it can represent. The core challenge is optimizing these parameters to find the best approximation within this vast function space.

The ability of neural networks to establish decision boundaries and perform representation learning hinges on their training process. Training involves adjusting the network's weights and biases to minimize the discrepancy between its predictions and the desired outcomes—a process known as optimization. Most optimization processes rely on automated differentiation techniques, with the most widely used method being *backpropagation*.

The optimization process works in three phases:

- **Forward Pass**: The network takes an input, works through its layers step by step, and produces an output. It then compares this output to the correct answer (the target) and calculates the loss.
- **Backward Pass**: The algorithm figures out how much each parameter contributed to the error. It uses a technique called the *chain rule* to trace the error backward through the network. The idea is that the final output is directly or indirectly intertwined with all the values of weights and biases of the network, much like a chain. The challenge is to quantify their impact and progressively adjust them to obtain better results. This is achieved by calculating the partial derivatives[7] of the weights or biases concerning the loss function.
- **Parameter Update**: After calculating all partial derivatives, we use an optimizer, an algorithm that adjusts the parameters to reduce the loss in the least amount of steps. Examples

[6] A decision boundary is a surface or curve separating different classes in the input space, marking where predicted labels switch.

[7] A derivative is a mathematical operation that quantifies the rate of change; a partial derivative tells us how much a change in a single parameter of a multi-parameter function affects the overall value of the function.

of popular optimizers include SGD, MomentumSGD, and Adam, with Adam being widely used due to its efficiency and adaptability. Although detailed discussions on optimizers are beyond the scope of this thesis, it is important to note that optimizers rely on hyperparameters such as the learning rate and batch size, which are chosen based on the specific problem through experimentation.

Through this iterative process, the network learns to minimize the loss function, improving its ability to fit the data and make accurate predictions [44]. This process, however, cannot guarantee finding the best possible function (global minimum); instead, it often converges to a local minimum. Most of the time, these local minima perform well in practice.

Why Neural Networks?

Neural networks are celebrated for their flexibility and representational power. In fact, they can approximate any arbitrary continuous function. The Universal Approximation Theorem formalizes this by stating that a feedforward neural network with at least one hidden layer can approximate continuous functions on a compact subset of $\mathbb{R}^n$, given sufficient neurons and suitable activation functions [41, 44]. This theorem underscores the potential of neural networks to model highly complex patterns and relationships in data, making them invaluable in a wide array of machine learning tasks.

Predicate-Powered Learning

In theory, with enough high-quality data and computational power, even a simple two-layer MLP could approximate any function [41]. Yet, despite growing datasets and increasing computational resources, practitioners consistently seek ways to embed prior knowledge into the learning process to improve both optimization efficiency and model generalization [41, 50].

Recently, a paradigm shift in statistical learning encourages moving away from brute-force searches across vast function spaces. Instead, focusing on a domain-informed search approach [46]. This emerging methodology, known as *predicate-powered learning*, introduces the use of predicates, fundamental truths or core concepts that are highly relevant to the problem at hand. Predicates serve as guiding principles within the learning process thus enhancing model performance.

By defining relevant predicates, we effectively constrain the search space, allowing the model to converge faster and more accurately toward meaningful solutions [51]. This predicate-based approach aligns with the broader goals of statistical learning: not only to approximate functions but also to do so in a way that reflects the essential structure and nuances of the problem domain.

[51] shows three main mechanisms through which predicates can influence learning:

1. **Input Transformation:** Predicates can transform the input data in a way that filters out irrelevant information, ensuring that only pertinent features remain. This allows the model to focus on essential aspects of the data.
2. **Hypothesis Class:** Predicates can be used to construct a learnable mapping function that inherently disregards irrelevant information.
3. **Algorithmic Guidance:** Predicates can guide the tuning of the mapping function through constraints imposed during the training process. A strategy further discussed in [46].

In all these mechanisms, predicates help steer the learning algorithm towards more efficient and relevant solutions, significantly reducing computational costs while improving model performance.

Predicates of the problem

The following invariances and constraints characterize the fundamental properties any intersection test or volume calculation between two tetrahedra must respect:

- **Rotational Invariance:** The intersection status and volume between two tetrahedra remain unchanged under any simultaneous rotation of both tetrahedra.
- **Translational Invariance:** The intersection status and volume between two tetrahedra are unaffected by any joint translation. Shifting both tetrahedra by the same vector in space does not alter the intersection outcome.
- **Scale Invariance:** If both tetrahedra are uniformly scaled, the intersection status remains unchanged, and the volume of intersection scales in proportion to the determinant of transformation.
- **Reflection Invariance:** The outcome is the same if both tetrahedra are reflected across a plane.
- **Permutation Invariance:** The vertex labeling of a tetrahedron is arbitrary, so permuting the vertices should not alter the outcome, also intersecting T_1 with T_2 is the same as intersecting T_2 with T_1.
- **Volume Constraints:** The volume of intersection between the two tetrahedra must fall within the range of 0 to $\frac{1}{6}$.

3.2 Data

3.2.1 Representation

The way a tetrahedron is represented fundamentally shapes both the computational efficiency and the algorithmic strategies applicable to geometric tasks. Different representations offer trade-offs between geometric fidelity, expressiveness, and computational complexity. Table 3.2.1 summarizes the most common formats and their associated characteristics.

Representation	**Description and Applications**
Cartesian Coordinates	Each vertex defined as (x, y, z). Intuitive and efficient for basic computations and transformations.
Spherical Coordinates	Encodes vertices using radius r and angles (θ, ϕ). Useful for rotational tasks and spherical projections.
Plücker Coordinates	Edges represented as 6D vectors capturing direction and moment. Key in projective geometry and line-space reasoning.
Half-Space Intersection	Defined by intersecting four face-bounding planes. Efficient for collision detection and containment checks.
Mesh Representation	Explicit vertices, edges, and faces with topological structure. Detailed but redundant for simple polyhedra.
Graph Representation	Complete graph K_4 structure over vertices. Suitable for combinatorial and topological analysis.
Nef Polyhedra	Boolean operations over half-spaces enable exact and robust modeling for constructive geometry.

Table 3.3: Alternative representations of a tetrahedron and their characteristics

While simple formats like Cartesian coordinates are computationally light and easy to use, they encode minimal structure, often requiring models to infer geometric relationships implicitly. In contrast, richer formats—such as meshes or graphs—embed geometric and topological structure directly, aiding tasks that benefit from explicit connectivity but introducing additional computational overhead [41].

I would argue that for most tetrahedron-focused applications, vertex-based representations strike the best balance between expressiveness and efficiency. A tetrahedron is uniquely defined by its four vertices; since the connectivity forms a complete graph (K_4)[8], edges and faces can be derived algorithmically, making additional storage unnecessary. This property also makes point-based representations ideal for use in both classical geometric algorithms and modern deep learning models designed for point clouds.

Point Clouds as a Minimal Representation

3D point clouds, a set of unordered points in three-dimensional space, provide a minimal representation that avoids storing explicit connectivity [15]. For a tetrahedra pair, this corresponds to precisely eight points in $\mathbb{R}^3$, fully defining the shape without redundancy.

[8] A complete graph K_4 means that every vertex is connected to every other vertex by an edge. So, with 4 vertices, there are edges between each pair, making 6 edges in total. This fully connects the shape without ambiguity.

Despite these advantages, point clouds also come with some hurdles. The lack of inherent structure means that relationships such as adjacency or orientation are not explicitly encoded. This places a greater burden on the learning algorithm to infer spatial relationships purely from coordinate data [41]. Additionally, because the points are unordered, any learning algorithm must treat permutations of the same tetrahedron identically. Capturing this permutation invariance in model design is non-trivial and requires deliberate architectural choices [52, 53].

Nonetheless, for simple 3D shapes like tetrahedra, point clouds represent a balance between simplicity and expressiveness. They provide just enough information to capture geometry while minimizing redundancy, making them an effective foundation for learning-based methods in tasks involving spatial reasoning, such as intersection detection and volume estimation.[9]

3.2.2 Diversity

Intersection Types and Geometric Likelihoods

The spatial relationship between two tetrahedra can fall into five canonical cases, adapted from Li and Chen [32] and Zinani [54]. These intersection types range from complete separation to volumetric overlap, as illustrated in Figure 3.3.

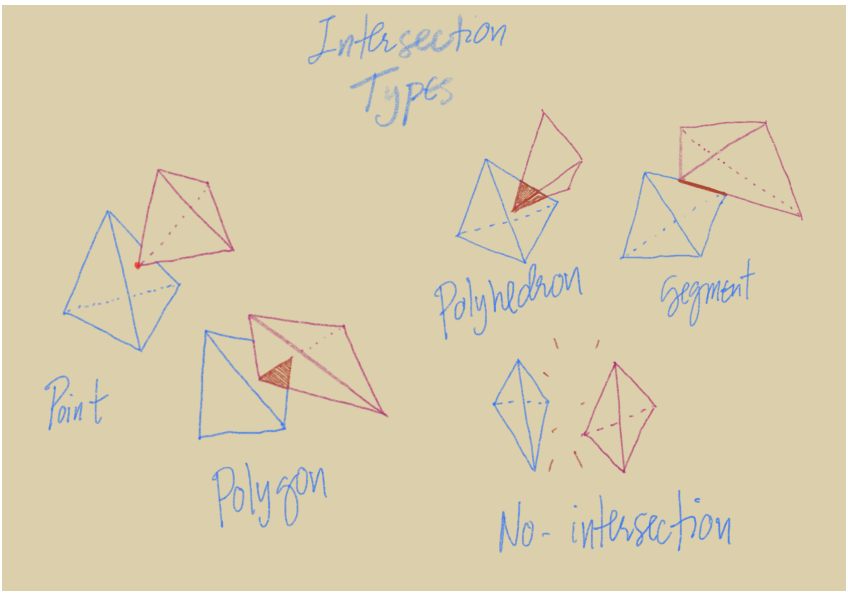

Figure 3.3: Five canonical intersection types between two tetrahedra: (0) No intersection—completely disjoint tetrahedra; (1) Point intersection—contact at a single vertex; (2) Segment intersection—shared edge between tetrahedra; (3) Polygon intersection—shared triangular face; (4) Polyhedron intersection—overlapping volumes creating a shared polyhedral region.

Table 3.4 summarizes these intersection cases and their relative probabilities under different sampling conditions.

In uniform random sampling (e.g., selecting all vertex coordinates from $[0, 1]$), most tetrahedra are small (expected volume ≈ 0.01), and the probability of intersection is low. Even in the rare case where both tetrahedra are maximally large (e.g., unitary tetrahedra), overlap remains geometrically improbable.

[9] If you were to solve the intersection problem analytically, you would only need the vertices of the tetrahedra because the edges and faces can be precisely calculated from these points. This means vertices contain all the essential information to understand the shape's geometry.

Table 3.4: Intersection Types and Likelihood Estimates

Case	Description	Likelihood Estimate
None (0)	Disjoint tetrahedra	High (especially in random sampling)
Point (1)	Intersection at a single vertex	Low in random sampling; high in simulations
Segment (2)	Common edge shared	Lower than Point; moderate in contact cases
Polygon (3)	Shared triangle face	Unlikely unless face-to-face contact occurs
Polyhedron (4)	Shared volume	Less likely than Case 0 in uniform sampling; common in simulations

However, in simulated environments (e.g., collision detection or physics engines), one tetrahedron is often static while the other is moved or projected toward it. In such settings, the most probable first contact is a vertex-to-face interaction (Case 1: Point), followed by shallow overlaps (Case 4: Polyhedron). Deep face-to-face intersections (Case 3: Polygon) are least likely in dynamic simulation due to strict spatial alignment requirements.

Volume Distribution Bias in Random Sampling

A naive method for generating intersecting tetrahedra is to sample all vertex coordinates uniformly in $[0, 1]$. However, this introduces a statistical bias: smaller intersection volumes occur far more frequently than larger ones. As shown in Zinani [54], the average volume of a randomly sampled tetrahedron in the unit cube is approximately ≈ 0.0138, and volumes exceeding 0.1 are extremely rare.

This imbalance affects model learning: frequent small volumes dominate the training set, leading to reduced accuracy on rare, high-volume cases. While generating a massive dataset increases coverage, it is computationally inefficient. A better solution is to deliberately oversample underrepresented cases during data generation—achieving diversity through controlled, distribution-aware sampling.

In summary, the classification function space defined by two tetrahedra is exponentially large due to the high input dimensionality and coordinate precision, making broad and representative sampling essential for effective generalization. Furthermore, the likelihood of different intersection types is not uniform—it strongly depends on the sampling regime. In purely random sampling within the unit cube, most pairs are disjoint, and even when they intersect, smaller overlaps are far more probable than large ones. By contrast, simulation-based sampling, where one tetrahedron moves toward another, biases the distribution toward more meaningful intersection events like point or volumetric contact. Finally, the distribution of intersection volumes is heavily skewed toward near-zero values, which can distort model learning if not corrected. Ensuring dataset diversity—across both intersection categories and volume scales—is therefore not only desirable but necessary for building a model capable of robust geometric reasoning.

3.2.3 Augmentation

To further enhance model robustness, augmentation techniques can embed the geometric invariances defined in Section 3.1. Two key strategies are tetrahedronwise and pointwise permutations, which expose the model to semantically equivalent but differently ordered configurations. These transformations increase data diversity without altering geometry. Alternatively, sorting strategies can enforce input consistency across samples:

- **Order-Based Sorting :** Sorting vertices or entire tetrahedra by spatial coordinates (e.g., x-axis) or using space-filling curves[10] introduces structured coherence that can benefit generalization.
- **Size-Based Ordering:** Standardizing the input by presenting the larger tetrahedron first reduces asymmetry and may stabilize predictions.
- **Difficulty-Based Curriculum:** Organizing training samples by geometric complexity or model uncertainty supports curriculum learning, accelerating convergence and improving final performance.

Because the problem is permutation-invariant—within each tetrahedron and across the pair—such augmentations can help learning robust and generalizable representations without requiring explicit architectural enforcement.

3.2.4 Transformations

Beyond data augmentation, geometric transformations can reframe the input space into a more structured form. One such method is the *Unitary Tetrahedron Transformation*, where one tetrahedron is normalized to a fixed reference configuration—a unitary tetrahedron—and the second tetrahedron is represented in this transformed space[11]. In practice, since the first tetrahedron serves as the reference, only the second tetrahedron needs to be fed into the model.

Similarly, affine or rigid-body transformations—such as *Principal Component Analysis* (PCA) alignment—can reduce geometric variance and standardize spatial context.

These transformations are evaluated in detail in Section 5.10.1, where we assess their impact on performance and convergence. In general, mapping inputs into a canonical form simplifies the learning task and promotes better generalization. However, even with normalization, the input, in the form of point clouds, remains unstructured and unordered. This poses unique challenges for neural networks, which must be architected to handle the discrete and irregular nature of 3D spatial data.

We now turn to the development of such architectures.

3.3 Neural Networks for 3D Point Clouds

3.3.1 Overview

Challenges in Processing Point Clouds

[10] Space-filling curves (e.g., Hilbert or Z-order) are continuous mappings from 1D to multi-dimensional space that preserve locality. They provide a deterministic way to impose spatial order on unordered points.

[11] The transformation space refers to the geometric embedding in which the reference tetrahedron is mapped to a canonical form. This typically involves an affine linear transformation, including translation, rotation, scaling, and shearing.

Point clouds are not evenly distributed, some areas are dense with details, while others are sparse, leaving gaps. Point clouds are unstructured, with no fixed spacing between points. They are also unordered, meaning the order in which points are stored does not change what they represent. These irregular, unstructured, and unordered properties make processing point clouds complex and require specialized methods to analyze them effectively [15, 48].

Early Approaches to Point Cloud Processing

In the past, these challenges were addressed by crafting domain-specific features. These hand-crafted features were used to augment the dataset and train traditional machine learning models [15, 48]. However, this approach faced significant limitations:

- **Poor Adaptability:** Feature extraction was often specific to certain scenarios and lacked generalization across diverse datasets.
- **Dependence on Expert Knowledge:** Designing effective features required deep expertise in the domain, making the process time-consuming and inaccessible to non-experts.
- **Susceptibility to Noise:** Real-world point clouds often include noise, outliers, and missing data, reducing the robustness of traditional approaches.

Advances in computing power and data processing have paved the way for deep learning transforming point cloud analysis. Deep learning methods, unlike traditional ones, can automatically learn features directly from data, making them more adaptable and robust [41, 44].

Transition to Deep Learning: Converting Point Clouds to Structured Representations

Some approaches opt to adapt point cloud data to well-established architectures such as Convolutional Neural Networks (CNNs)[12] and Graph Neural Networks (GNNs)[13] by converting point clouds into structured formats such as voxels, 2D projections, or graphs.

Voxelization maps objects onto a 3D volumetric grid[14], enabling the use of 3D CNNs. However, this introduces quantization errors[15] and incurs high memory overhead.

Multi-view projections convert point clouds into 2D images from multiple viewpoints[16], allowing processing with conventional CNNs. While this simplifies the task, it risks loss of spatial information and introduces a dependency on optimal view selection.

Graph-based methods represent point clouds as graphs, where points are nodes connected by edges based on spatial proximity or other criteria. This structure enables the use of GNNs, albeit at the cost of increased computational complexity[17] [48].

While these approaches leverage the strengths of well-established architectures, their reliance on data conversion adds overhead and can limit performance in applications requiring high inference

[12] CNNs are deep learning models particularly effective for processing data with a grid-like topology, such as images.

[13] GNNs operate on graph structures, enabling the modeling of relationships between entities through nodes and edges.

[14] A voxel (volumetric pixel) is the 3D equivalent of a pixel, representing a value on a regular grid in three-dimensional space.

[15] Quantization error refers to the loss of precision due to discretizing continuous space into finite voxel units.

[16] Each projection corresponds to a rendered view of the 3D structure from a specific camera angle.

[17] Graph-based learning often involves operations like message passing and neighborhood aggregation, which scale poorly with graph size.

speed and scalability. These limitations have driven the development of methods that directly process raw point cloud data, as discussed in the followings.

3.3.2 DeepSets (2017)

In 2017, *Manzil Zaheer, Satwik Kottur, Siamak Ravanbhakhsh et al.* introduced DeepSets in [53], a novel deep learning framework designed to handle machine learning tasks on sets. Unlike traditional neural networks that operate on fixed-dimensional vectors, DeepSets addresses problems where the input data is a set and objective functions must be invariant to the permutation of elements within the set.

The core theoretical contribution of DeepSets is a theorem demonstrating that any permutation-invariant function can be decomposed into a specific form:

$$\rho\left(\sum_{x\in X}\phi(x)\right),$$

where ϕ and ρ are suitable transformations. This structural insight allows for the design of deep network architectures that can process sets of varying sizes. The model works by transforming each element x_m in the set into a representation $\phi(x_m)$, summing these representations, and then processing the aggregated result through a network ρ.

DeepSets also defines conditions for permutation equivariance, enabling the creation of deep layers where permuting input elements leads to a corresponding permutation of output labels. This is achieved through specific parameter-sharing schemes, such as using max-pooling operations that are commutative. The versatility of DeepSets has been demonstrated across various applications, including population statistic estimation, point cloud classification, set expansion, and outlier detection.

3.3.3 PointNet (2017)

In 2017, *Charles R. Qi, Hao Su et al.* introduced PointNet in [55], a groundbreaking neural network for direct point cloud processing. PointNet processes point clouds by applying learnable rigid transformations using the T-Net module, extracting point-wise features through shared fully connected layers for each point, and aggregating global features using a maximum pooling function. While revolutionary, PointNet has limitations, such as its inability to capture local relationships between points, which reduces its effectiveness for certain datasets.

3.3.4 Pointwise MLP Methods

Since then, the field has advanced significantly, giving rise to many new architectures. These methods are generally based on convolution operations, graphs, attention, pointwise MLP processing or other techniques. For a detailed explanation of these methods, please refer to [15]. In this study, we focus on pointwise MLP methods for the following reasons:

- **Relevance to Our Dataset:** Most other methods emphasize local feature extraction to address PointNet's weaknesses. However, for our dataset, pairs of tetrahedrons, the simplest 3D shapes, fine-grained local feature extraction is unnecessary.
- **Efficiency:** Many advanced architectures require substantial computational resources and memory. Simpler methods are lightweight and more practical for efficient processing. In fact most real world applications tend to use the PointNet-based architectures.

- **Dataset Simplicity:** State-of-the-art methods are typically evaluated on complex datasets, achieving high accuracy. Our dataset is much simpler, and most methods are expected to perform well without added complexity.
- **Simplified Design:** these methods prioritize simplicity to boost inference speed without sacrificing performance. These streamlined architectures deliver results comparable to more complex approaches, making them ideal for our needs.
- **Accessibility and Usability:** Simpler architectures are easier to experiment with and understand. By choosing a straightforward approach, we make it easier for others to engage with this research and contribute to future development.

These methods employ shared MLPs to transform points into higher-dimensional feature representations. Since point clouds are inherently unordered, these architectures utilize symmetric aggregation functions—such as sum, max pooling or average pooling—to generate a global feature vector that remains invariant to permutations of the input points.

PointNet++, 2017 [52]

Building on the success of PointNet, this work addresses one of its major limitations: the lack of awareness of local relationships between points. Inspired by the success of CNNs and their ability to capture local patterns, PointNet++ adopts a hierarchical approach. Just as CNNs begin by focusing on small image details and progressively abstract larger patterns, PointNet++ processes points in smaller groups first and gradually captures more global features.

The architecture operates by selecting small neighborhoods of points, extracting features from these neighborhoods, grouping them into larger sets, and then extracting higher-level features. This hierarchical process continues until the entire point set is processed.

The Three Fundamental Stages:

1. Sampling: The point set is partitioned into smaller subsets using the *Furthest Point Sampling (FPS)* method. This approach selects centroids such that each centroid is as far as possible from all previously selected points. These centroids represent the groups to be processed.
2. Grouping: For each centroid, the *K nearest points* are selected to form a group. The value of K can vary across different groups.
3. Feature Extraction: The features of each group are extracted using the original *PointNet* architecture. This process produces a compact representation of each group.

This hierarchical process of sampling, grouping, and feature extraction aims to ensure that both local and global geometric features are effectively captured across the point set.

PointMLP, 2022 [56]

PointMLP addresses the computational cost of elaborate local feature extractors in point cloud processing by leveraging only residual MLPs, focusing on both efficiency and accuracy. Inspired by PointNet++, it introduces two key innovations, a geometric affine transformation module to learn a canonical geometric representation of points. Residual MLPs with batch normalization and max pooling for efficient, permutation-invariant feature extraction.

Stages of PointMLP

The architecture follows the same stages as PointNet++:

1. Grouping: Performed via Farthest Point Sampling (FPS).
2. Sampling: Groups points using *K nearest neighbors* (KNN).
3. Feature Extraction: Introduces the primary innovations:
    - Geometric Affine Transformation Module: Replaces PointNet's T-Net by normalizing and transforming local features with learnable affine linear transformation.
    - Residual Point Blocks: Processes points through a sequence of residual MLP layers. Each layer consists of batch normalization, an activation function, and residual connections. These blocks capture complex feature interactions, followed by a max pooling operation to aggregate features.

PointNeXt, 2022 [57]

PointNeXt revisits PointNet++ and modernizes it with updated training, data augmentation techniques, scaling strategies and minor architectural changes.

Key Enhancements

- Data Augmentation: Includes whole point sampling, jittering, random scaling, color dropping, and other techniques.
- Optimization: Employs label smoothing, uses *AdamW* instead of *Adam*, and adopts cosine decay for learning rate scheduling instead of step decay.
- Scaling: Incorporates receptive field scaling by adopting a larger radius for neighborhood queries and model scaling by adding more set abstraction blocks.

The impact of each technique is documented in the paper.

Architectural Changes Compared to the original PointNet++, PointNeXt introduces the following improvements:

1. An initial MLP layer to preprocess input features.
2. Feature extraction via MLPs and max pooling, replacing PointNet and removing the T-Net module.
3. Addition of inverted residual MLP modules to mitigate vanishing gradients and improve network efficiency.

These seemingly simple changes result in a lightweight, high-performance model.

3.4 Publicly Available Point Cloud Datasets

There are many publicly available datasets used to evaluate the efficiency of point cloud understanding models annotated with ground-truth labels. Below, we describe some of the most widely used datasets [15]:

Dataset	Year	Samples	Classes	Data Type and Source
ModelNet40	2015	12,311	40	Mesh-derived synthetic objects
ModelNet40-C	2015	185,000	15	Point cloud synthetic objects
ModelNet10	2015	4,899	10	Mesh-derived synthetic objects
Sydney Urban Objects	2015	588	14	Real-world point cloud scans
ShapeNet	2015	51,190	55	Mesh-based synthetic shapes
ScanNet	2017	12,283	17	RGB-D real-world scenes
ScanObjectNN	2019	2,902	15	Real-world object point clouds

Table 3.5: Overview of widely-used point cloud classification datasets

Datasets representing objects as meshes typically sample points uniformly from their surfaces, then center and scale objects to fit within a unit sphere. Commonly, over 1,000 points per object are sampled to capture fine detail.

As Table 3.5 shows, these datasets often include more classes and larger point counts than ours, posing more complex challenges. Strong performance on these benchmarks generally suggests good generalization potential to our dataset.

To date, no publicly available datasets focus specifically on tetrahedron-tetrahedron interactions.

3.5 Evaluation Metrics

This section presents a comprehensive evaluation framework to assess model performance from both technical and practical standpoints. For classification tasks, we focus on predictive correctness, fairness across classes, and discriminative ability. For regression, our emphasis is on absolute deviation and consistency with the target distribution. Based on these goals, we adopt the following set of metrics.

3.5.1 Classification Metrics

Loss Function

We explore **Binary Cross Entropy with Logits** as the primary loss for classification. This formulation combines the sigmoid activation and cross-entropy loss into a single, numerically stable operation, making it especially suitable for binary classification tasks with raw logit outputs.[18]

$$\mathcal{L}_{\text{BCE}} = -\left(y \cdot \log(\sigma(\hat{y})) + (1-y) \cdot \log(1-\sigma(\hat{y}))\right)$$

Where:

- $y \in \{0, 1\}$ is the true label,
- $\hat{y} \in \mathbb{R}$ is the raw model output (logit),
- $\sigma(\hat{y}) = \frac{1}{1+e^{-\hat{y}}}$ is the sigmoid activation.

Evaluation Metrics Table 6.5 summarizes classification metrics that assess correctness and discriminative power:

Metric	Formula	Interpretation
Overall Accuracy (OA)	$\text{OA} = \frac{TP+TN}{TP+TN+FP+FN}$	Fraction of total correct predictions.
Mean Class Accuracy (mAcc)	$\text{mAcc} = \frac{1}{C}\sum_{i=1}^{C} \frac{TP_i}{TP_i+FN_i}$	Average per-class recall, addressing class imbalance.
Area Under ROC Curve (AUC)	$\text{AUC} = \int_0^1 \text{TPR}(\text{FPR})\, d(\text{FPR})$	Overall ability to distinguish classes across thresholds.

Table 3.6: Classification evaluation metrics and interpretation

Notation for confusion matrix components is in Table 3.5.1:

Symbol	Term	Definition
TP	True Positives	Correct positive predictions
TN	True Negatives	Correct negative predictions
FP	False Positives	Incorrect positive predictions
FN	False Negatives	Incorrect negative predictions
TPR (Sensitivity)	True Positive Rate	$\frac{TP}{TP+FN}$: Proportion of positives correctly identified
FPR	False Positive Rate	$\frac{FP}{FP+TN}$: Proportion of negatives misclassified

Table 3.7: Confusion matrix notation and definitions

[18] Unlike separate sigmoid + BCE implementations, `BCEWithLogitsLoss` prevents floating-point underflow/overflow for extreme logit values (e.g., very large positive or negative predictions). This ensures more stable gradient flows during backpropagation, which improves convergence in practice. Additionally, its design aligns directly with binary decision tasks, as it maps unbounded inputs to $[0, 1]$ probabilities while penalizing misclassifications based on their confidence.

3.5.2 Regression Metrics

Loss Function

We use **Root Mean Log Squared Error (RMLSE)** as the regression loss, prioritizing relative error over absolute. This suits intersection volume predictions spanning several magnitudes.

$$\mathcal{L}_{\text{RMLSE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(1+y_i) - \log(1+\hat{y}_i))^2}$$

Where:

- y_i, $\hat{y}_i$ are true and predicted values,
- $\log(1+\cdot)$ prevents instability near zero.

Absolute Error Metrics

Absolute errors quantify prediction deviations in original volume units, crucial for practical impact assessments. They complement relative errors by reflecting true scale differences.

Metric	Formula	Interpretation
Mean Absolute Error (MAE)	$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}\lvert y_i - \hat{y}_i\rvert$	Average absolute deviation in original units.

Table 3.8: Absolute regression error metrics

Distributional Correctness

To evaluate how well the model predicts intersection volumes across different value ranges, we introduce *Mean Bin Accuracy*. This metric complements standard regression error by assessing the model's ability to assign predictions to the correct discretized volume bin, thereby capturing categorical correctness beyond raw numerical proximity.

We discretize the continuous volume range into N bins (e.g., logarithmic or uniform), then compute accuracy per bin based on whether predicted volumes fall into the same bin as the ground truth. Formally:

$$\text{Bin Accuracy}_i = \frac{\#\{\text{correct bin matches in } i\}}{\#\{\text{ground truth samples in bin } i\}}, \quad \text{Mean Bin Accuracy} = \frac{1}{N}\sum_{i=1}^{N}\text{Bin Accuracy}_i$$

This metric helps reveal distributional biases—e.g., models performing well on high-volume intersections but poorly on sparse or low-overlap cases.

Categorical Agreement

Cohen's Kappa quantifies agreement between predicted and true volume categories while adjusting for chance agreement. It reflects the reliability of volume classification beyond accuracy alone. This serves as a measure of robustness.

Component	Formula	Description
Cohen's Kappa (κ)	$\kappa = \frac{p_o - p_e}{1 - p_e}$	Agreement beyond chance expectation.
Observed Agreement (p_o)	$p_o = \frac{1}{N} \sum_{i=1}^{C} O_{ii}$	Fraction of exact class matches.
Expected Agreement (p_e)	$p_e = \sum_{i=1}^{C} \left(\frac{\sum_{j=1}^{C} O_{ij}}{N} \cdot \frac{\sum_{j=1}^{C} O_{ji}}{N} \right)$	Chance-level agreement from class marginals.

Table 3.9: Cohen's Kappa components and formula

Kappa Value	Interpretation
1.00	Perfect agreement
0.80–0.99	Almost perfect
0.60–0.79	Substantial
0.40–0.59	Moderate
0.20–0.39	Fair
0.00–0.19	Slight
0.00	No agreement beyond chance
<0.00	Less than random agreement

Table 3.10: Interpretation scale for Cohen's Kappa

3.5.3 Efficiency Metrics

In addition to accuracy, computational performance is essential, particularly in resource-constrained or real-time environments.

Metric	Units	Description and Importance
Latency	Milliseconds (ms)	End-to-end time per sample. Key for real-time applications.
Throughput	Samples per second (SPS)	Processing speed for batched workloads.
Model Size	Megabytes (MB)	Storage footprint, affecting deployability.

Table 3.11: Efficiency metrics for inference performance

Collectively, these metrics form a holistic framework for evaluating models in terms of accuracy, fairness, robustness, and deployment feasibility—especially within geometric modeling contexts.

3.6 Comparative Analysis

Table 3.6 summarizes the performance of several influential point cloud classification models evaluated on two widely-used benchmarks: ModelNet40 and ScanObjectNN. These datasets represent

synthetic and real-world challenges, respectively, providing a balanced view of model effectiveness.

Accuracy vs. Throughput Trade-offs

PointNet [55], the pioneering method for direct point cloud processing, achieves a baseline overall accuracy (OA) of 89.2% on ModelNet40 and 68.2% on ScanObjectNN, with exceptionally high throughput (4212 instances/sec). This highlights its efficiency and simplicity but also its limitations in capturing fine local structures, especially in noisy real-world data like ScanObjectNN.

PointNet++ [52] improves accuracy substantially by incorporating local neighborhood information through hierarchical grouping, reaching 91.9% OA on ModelNet40 and 77.9% on ScanObjectNN. However, this comes with a significant throughput drop (1872 instances/sec), reflecting the computational cost of local feature extraction.

More recent pointwise MLP-based architectures, such as PointMLP [56] and PointNeXt [57], push accuracy further, especially on the challenging ScanObjectNN dataset, achieving up to 88.4% OA. PointNeXt balances improved accuracy (87.7%) and throughput (2040 instances/sec), whereas PointMLP prioritizes accuracy at the cost of much lower throughput (191 instances/sec).

It is important to note that throughput heavily depends on the hardware used for testing. Since reported results come from independent studies, machine-independent throughput comparisons cannot be reliably provided.

Insights for Model Selection

The choice of model depends on application requirements:

- For scenarios demanding high-speed inference on relatively clean or synthetic data, PointNet remains a competitive choice.
- When local geometric relationships are critical, PointNet++ and PointNeXt offer a strong accuracy boost with moderate computational overhead.
- For highest accuracy in real-world, noisy environments, recent MLP-based models like PointMLP are preferred despite their higher computational cost.

In contexts involving simpler datasets or resource-constrained applications, these trade-offs should be carefully evaluated, balancing accuracy, inference speed, and hardware availability.

Model	**Year**	**ModelNet40 (OA %)**	**ScanObjectNN (OA %)**	**Throughput (instances/sec)**
PointNet [55]	2017	89.2	68.2	4212.0
PointNet++ [52]	2017	91.9	77.9	1872.0
PointMLP [56]	2022	94.1	85.4	191.0
PointNeXt [57]	2022	93.2	87.7	2040.0

Table 3.12: Comparison of point cloud classification models on ModelNet40 and ScanObjectNN datasets.

3.7 Related Work

3.7.1 A Machine Learning Framework for Volume Prediction (2019)

In 2019, *Umutcan Önal and Zafeirakis Zafeirakopoulos* investigated the use of machine learning to predict the volumes of polytopes, addressing both classification (volume comparison) and regression (volume estimation) tasks. Their approach involved representing polytopes via vertex coordinates and applying models such as random forests and neural networks. To accommodate varying numbers of vertices and dimensionality, they employed autoencoders for feature extraction, enabling both modular and end-to-end learning pipelines. The dataset comprised 19,000 polytopes.

For training, they used Binary Cross-Entropy loss, a learning rate of 0.001, the Adam optimizer, and trained over 30 epochs. Models trained on fixed-dimension inputs consistently outperformed those designed for variable dimensions. Interestingly, joint models that performed both classification and regression achieved better comparison accuracy but at the cost of slightly lower R^2 scores for regression. Another notable finding was that input normalization degraded performance—contrary to common machine learning practice.

Their best model achieved a volume comparison accuracy of 0.9631 and an R^2 score of 0.9980 for volume estimation. To the best of our knowledge, this is the earliest application of machine learning to polytope volume prediction.

3.7.2 Polytopes and Machine Learning (2021)

[58]

This study applies supervised learning models—including MLPs, CNNs, and Random Forests—to predict geometric properties of lattice polytopes using Plücker coordinates as input features. The Plücker-based representation outperformed vertex-based alternatives, achieving over 90% accuracy in predicting volumes within an acceptable range, and demonstrating moderate success in reflexivity classification.[19] The experiments encompassed both 2D and 3D polytopes.

However, the work omits several critical implementation details, such as coordinate precision, volume resolution, or inference latency. Furthermore, the structural and statistical properties of the problem space are not explored. Consequently, while the study offers a valuable proof of concept, it lacks the depth and transparency necessary for deployment in downstream or production-grade applications.

[19] The authors argue that Plücker coordinates better preserve volumetric information by encoding edge-level relationships.

3.8 Complementary Work: Tetrahedral kDet, Linear Time Collision Detection for Tetrahedral Meshes

From a theoretical standpoint, detecting collisions between two polygonal objects is straightforward: check every polygon of one object against every polygon of the other [17]. However, in practice, this brute-force approach is computationally infeasible for most applications. This complexity arises from the theoretical possibility of fitting an infinite number of 2D polygons into a confined 3D volume, because of that, in theory, there can be an infinite amount of collision checks necessary for an intersection test. However, such scenarios are largely artificial and uncommon in practical applications [14, 17]. A challenge in collision detection lies in determining the number of potentially colliding polygons within a 3D scene. Significant effort has therefore been directed toward developing acceleration data structures that minimize the number of potentially colliding polygon pairs to be tested [17].

In 2017, [17] introduced kDet, a parallel linear-time collision detection algorithm for polygonal meshes designed to run on GPUs. It focuses on scenarios where each polygon interacts only with a limited number of others in its environment, introducing the concept of k-freeness:

- A polygon p is k-free if fewer than k polygons meet the following criteria:
  1. Have a larger minimum enclosing sphere than p.
  2. Intersect the volume formed by sweeping a sphere of diameter $d/2$ around p.

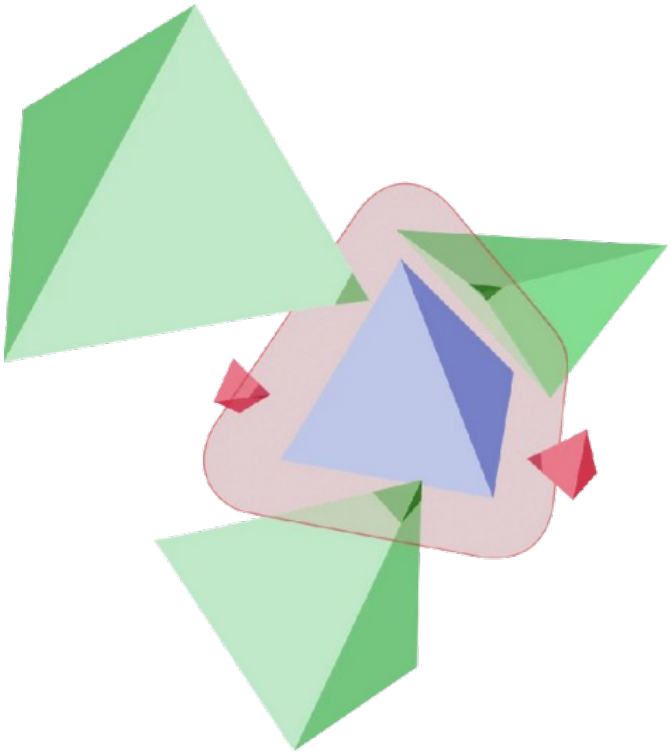

Figure 3.4: 4-free tetrahedron (blue); red area marks the Minkowski-sum[1] of said tetrahedron and a sphere half the diameter of its minimum enclosing sphere; a 4-free polyhedron intersects at most 3 "larger" polyhedra (determined by the minimum enclosing sphere) with said Minkowski-sum. *Source*:[14]

[20] Simplifying further, the polygons considered for k-freeness are those that are equal to or larger than the polygon in question. A set of polygons is considered k-free if every polygon $p_i \in A$ satisfies these conditions. The algorithm's key idea is to check collisions only against nearby polygons that are "larger."

Many real-world problems require managing internal structures rather than just surfaces, which polygons alone cannot adequately represent. To address this, the concept of k-freeness was expanded to tetrahedral meshes. In 2023, [14] extended kDet to volumetric 3D objects with tetrahedral meshes. They demonstrated that the algorithm retains its linear runtime property and proved that for a k-free set of polyhedra, the maximum number of intersections between a polyhedron p and larger polyhedra $p_j \in P$ is qk, where q is the minimum number of spheres of diameter $d/2$ needed to fully cover each polyhedron.

3.8.1 Algorithm Overview

The kDet algorithm for tetrahedral meshes proceeds as follows:

Algorithm 3.1 kDet Algorithm for Detecting Intersections in Tetrahedral Meshes

```
Insert all tetrahedra into a hierarchical grid
Traverse the grid to collect all potential collision pairs via k-freeness predicate
Remove duplicate tetrahedron pairs
for all unique tetrahedron pairs (T_i, T_j) do
    Perform intersection test on (T_i, T_j)
    if T_i intersects T_j then
        Add (T_i, T_j) to the intersection list
    end if
end for
return List of intersecting tetrahedron pairs
```

By leveraging the geometric properties of individual objects rather than their configurations (i.e., focusing on object size rather than position and orientation), kDet achieves near-linear runtime and close-to-constant time performance when executed in parallel [17]. This efficiency makes it a valuable tool for collision detection in 3D environments.

3.8.2 Potential Integration with Machine Learning

While kDet excels at efficiently identifying potentially colliding tetrahedron pairs, its final step relies on traditional intersection tests. This work proposes augmenting kDet by running intersection tests on the GPU alongside it. A trained neural network could potentially predict collision outcomes for all identified pairs simultaneously, taking advantage of parallel computation to improve both speed and scalability.

[20] The Minkowski sum of two sets is the collection of all points obtained by adding each point in one set to each point in the other.

3.9 Conclusions, Research Gaps, and Challenges

3.9.1 Overview

In computer graphics, geometric algorithm development for tasks such as tetrahedron-tetrahedron intersection and volume computation often relies on explicit queries tailored to solve specific problems. While baseline algorithms for these tasks exist, practitioners continually optimize them to enhance their applicability, robustness, resource efficiency, and performance. This optimization process demands a deep understanding of the problem, the limitations of existing methods, and occasionally, the invention of novel approaches—leveraging human expertise to improve algorithmic efficiency.

However, this approach is inherently challenging, time-intensive, and constrained by human capacity to devise increasingly efficient queries. In his essay [59], *The Bitter Lesson*, Rich Sutton highlights how the most significant advances in intelligent systems have resulted from adopting automated search methods over handcrafted solutions. Extending this principle to geometric algorithms offers the opportunity to transition from optimizing explicit queries to discovering them through data-driven learning processes.

Recent years have witnessed the rise of deep learning, with numerous architectures designed for 3D data. Despite this progress, in our research, limited research has applied these techniques to traditional geometric algorithms. Using neural networks for tasks such as tetrahedron-tetrahedron intersection and volume computation could enable a unified model to handle these heterogeneous computations simultaneously and at scale. As data availability and computational power grow, these methods are expected to improve, decoupling optimization from direct human expertise. Human insight would remain critical, but its focus should shift to constraining the search space for machine learning algorithms, following a predicate-powered learning approach.

This paradigm introduces trade-offs. Machine learning models are inherently data-driven and cannot guarantee finding the best predictor, potentially introducing errors compared to traditional geometric algorithms. Minimizing this error is essential. Furthermore, ensuring the robustness, generability, and reliability of these models requires extensive testing to understand their limitations.

3.9.2 Addressing Research Questions

RQ1 *How can a dataset of tetrahedra pairs be generated to effectively train a neural network for intersection prediction and volume estimation?*

A point cloud representation of tetrahedron vertices appears to be a suitable choice for this task due to it's simplicity, low storage requirement and sufficiency to represent tetrahedrons. To ensure model generalization, the dataset must exhibit diversity: various intersection types, tetrahedron sizes, vertex permutations, tetrahedron permutations, different orientations, and different volume distributions. Randomization techniques and data augmentation can help enrich the data.

RQ2 *Which neural network architectures are most suitable for accurately predicting whether two tetrahedra intersect and, if so, estimating the intersection volume?*

Designing an effective model for this task requires alignment with the underlying geometric and mathematical structure of the problem. The following constraints and design objectives inform architectural choices:

- **Permutation Invariance:** The model must be invariant to the ordering of vertices within each tetrahedron and to the ordering of the two tetrahedra themselves. This ensures that

predictions are consistent under arbitrary input permutations. Symmetric aggregation functions—such as max pooling or summation—are commonly used to enforce this, as seen in architectures like PointNet and DeepSets.

- **Transformation Invariance:** The model's output should remain stable under simultaneous transformations of both tetrahedra, including scaling, rotation, reflection, and translation. Shearing is also relevant, as affine transformations modify volume according to the determinant of the transformation matrix.[21] Geometric modules such as the Geometric Affine Module (GAM) from PointMLP or T-Net from PointNet can be incorporated to canonicalize inputs and enhance invariance.
- **Volume Constraint:** The predicted intersection volume must lie within a geometrically valid range. For tetrahedra within the unit cube, the theoretical upper bound is $\frac{1}{6}$. However, achieving this bound is rare in practical scenarios. To enforce this constraint, outputs can be bounded using a scaled sigmoid function or regularized via loss penalties for out-of-range predictions. Volume normalization via input rescaling may also aid generalization across varying data distributions.
- **Scalability and Efficiency:** The architecture must support efficient training and inference, particularly for large-scale datasets or real-time applications. Residual connections and hierarchical encoders can improve gradient flow and representation capacity. Starting from a minimal architecture and progressively adding complexity enables controlled ablation and performance tracking.

These design principles ensure that the model not only achieves high predictive performance but also respects the problem's intrinsic geometric structure and invariants.

RQ3 *How does the proposed model compare to existing state-of-the-art methods for tetrahedron-tetrahedron intersection detection and volume computation in terms of accuracy, speed, and scalability?*

This question is addressed in the following chapters through empirical benchmarks and comparative analysis.

[21] In 3D, an affine transformation can stretch, rotate, reflect, shear, or translate a shape. The determinant of its matrix quantifies how volume is scaled.

Chapter 4

Development

4.1 Data Generator

As discussed in Section 3.2.1, the raw minimal representation of our problem consists of a point cloud containing the vertices of two tetrahedra, along with two associated labels describing their spatial interaction. The dataset is structured as follows:

4.1.1 Dataset Structure

Component	Description	Details
Tetrahedrons	Each sample represents a pair of tetrahedra	Each tetrahedron is defined by 4 vertices, each with 3D coordinates
Features	24 total input features (12 per tetrahedron)	Column naming: $\mathtt{T_t\_V_v\_}\alpha$, where $t \in \{1, 2\}$, $v \in \{1, 2, 3, 4\}$, $\alpha \in \{x, y, z\}$
Coordinates	Normalized to $[0, 1]^3$	Values are generated independently using an i.i.d. uniform distribution
HasIntersection	Binary classification output	0 = no intersection, 1 = intersection (including touching)
IntersectionVolume	Regression target output	Non-negative scalar: 0 if touching, > 0 if overlap exists
Input Space	Feature space	$\mathbb{R}^{24}$, corresponding to 24 normalized 3D coordinates
Output Space	Label space	HasIntersection $\in \{0, 1\}$; IntersectionVolume $\in \mathbb{R}_{\geq 0}$

Table 4.1: Structure of each dataset sample, including input and output spaces.

The objective of the data generator is to explore the configuration space of tetrahedron pairs. It must independently cover both intersecting and non-intersecting cases across diverse geometric configurations.

To that end, the generator is designed around three core pillars as we discuss in the next sections:

- **Quality:** Ensure geometric validity through checks such as minimum volume and normalized coordinates.
- **Diversity:** Sample broadly across orientations, sizes, translations, and intersection types.
- **Efficiency:** Enable large-scale generation with minimal computational overhead.

4.1.2 Quality

A trustworthy dataset begins with high-fidelity geometric representation and mathematically precise labels. We address two primary sources of potential error: coordinate representation and label construction.

Coordinate Precision

All vertex positions are sampled uniformly from the unit cube $[0, 1]^3$, using C++'s native `double` type (64-bit floating-point with ∼16 decimal digits of precision). This provides approximately 10^{16} distinct values per coordinate within $[0, 1]$.

Precision Metric	Value
Decimal digits of precision	~ 16
Distinct values per coordinate	$\sim 10^{16}$
Total representable configurations	$\gg 3 \times 10^{360}$

Table 4.2: Coordinate Precision Metrics

[1]

This enormous configuration space supports dense sampling across spatial relationships while maintaining numerical stability in near-degenerate edge cases, enabling accurate capture of phenomena like grazing contacts and near-coplanar face interactions.

The Challenge of Floating-Point Arithmetic

Standard floating-point arithmetic introduces rounding errors and limited precision, critical in nearly degenerate geometric cases. These inaccuracies can cause incorrect geometric predicates, misclassifying intersections or producing erroneous volumes 3.1.3.

CGAL Solution

The Computational Geometry Algorithms Library (CGAL) provides high-precision geometric computation with robust algorithms for intersection detection and volume computation. Its key feature is *exact arithmetic*, guaranteeing correctness even for constructive geometry computations.

CGAL Component	Purpose	Characteristics
Kernels	Define geometric primitive representation	Double-based (fast) vs. Exact (EPEC - mathematically correct)
Predicates	Boolean geometric queries	Exact decisions on spatial relationships, minimizing floating-point errors
Primitives	Fundamental geometric shapes	Low-level building blocks (points, lines, tetrahedrons)

Table 4.3: CGAL Components Overview

Tetrahedron Construction

Each tetrahedron is constructed by independently sampling four points from the unit cube $[0, 1]^3$, using CGAL's `Tetrahedron_3` primitive. The following validation criteria are applied to ensure geometric soundness:

[1] $\sim 3 \times 10^{360}$ vastly exceeds the estimated 10^{80} particles in the observable universe.

Validation Step	Method	Purpose
Degeneracy Check	`is_degenerate()` predicate	Ensures that tetrahedra have non-zero volume (no coplanar or collinear vertices)
Efficiency Priority	No minimum volume enforced	Maximizes sample generation rate by not filtering small, valid tetrahedra in a raw phase generation.
Validity Threshold	$\text{Volume}(T) > \epsilon_{\text{machine}}$	Prevents floating-point collapse; $\epsilon \approx 10^{-15}$

Table 4.4: Tetrahedron Validation Criteria

Labels Generation

Each tetrahedron pair is labeled using a two-part exact-computation pipeline:

1. **Binary Intersection Flag:** Computed using `CGAL::do_intersect(T1, T2)`, which identifies any overlapping region (volume, face, edge, or vertex contact).
2. **Intersection Volume:** Computed using the following exact geometric pipeline:
   - Construct symbolic intersection via `Nef_polyhedron_3`.
   - Convert the result to a triangulated surface mesh.
   - Compute exact volume using `CGAL::Polygon_mesh_processing::volume`.
   - Cast the final result to `double` only at the final step for storage.

The ground truth labels are generated using exact predicates and robust CGAL primitives:

- **Vertex Coordinates:** All vertices are sampled independently from a uniform distribution without approximation.
- **Intersection Status:** Determined using exact geometry predicates, with no floating-point error propagation.
- **Intersection Volume:** Computed entirely with exact arithmetic, with the only loss of precision occurring when converting the final volume to `float`. This truncation introduces a maximum error of $\sim 10^{-15}$, well below thresholds relevant to simulation or graphics applications.

As a result, the dataset can be considered numerically exact, with negligible precision loss restricted to the final output representation.

Validation and Inspection

4.1.3 Diversity and Generation Algorithms

Naïve sampling of tetrahedra pairs uniformly in $[0, 1]^3$ often produces trivial or rare intersection configurations. To ensure diverse geometric relationships, we construct samples across five canonical interaction modes. Each mode leverages targeted heuristics for sample generation and is rigorously verified using CGAL predicates, summarized in Table 4.6.

Validation Type	Method	Purpose
Automated Filters	Logical consistency checks	Eliminate invalid samples (e.g., positive volume with zero intersection flag)
Manual Inspection	Visual review of 30-50 random samples	Verify geometric label correctness and absence of degenerate tetrahedra

Table 4.5: Validation and Inspection Methods

Interaction Mode	Generation Strategy	Verification Method
No Intersection	Generate random pairs until disjoint	`CGAL::do_intersect` returns false
Point Contact	Place T_2 vertex on T_1 face, sample others away	Verify single-point intersection via intersection size = 1 vertex
Edge Intersection	Align edges from T_1 and T_2, offset remainder	Confirm edge-only contact through degenerate volume and shared edge segment
Face Intersection	Place T_2 face coplanar with T_1 face	Validate planar face contact using orientation predicates
Volume Intersection	Stratified sampling via inverse binning: subdivide $[a, b]$ into n bins and generate N samples with target volumes uniformly distributed across bins	Use `Nef_polyhedron_3` for precise boolean intersection and verify resulting volume falls into the target bin. Most intersections yield small volumes without constraint

Table 4.6: Canonical Interaction Modes and Validation Methods

The following subsections describe the generation algorithms for each interaction mode, demonstrating how diversity is systematically enforced through geometric constraints and targeted sampling.

No Intersection

Algorithm 4.1 Non-Intersecting Tetrahedra

1: **repeat**
2: Generate random tetrahedra T_1, T_2
3: Check disjointness with `CGAL::do_intersect`
4: **until** No overlap detected
5: **return** (T_1, T_2)

Sampling two tetrahedra uniformly in a unit cube typically yields disjoint pairs due to spatial sparsity, making this process efficient. Figure 4.1 shows examples.

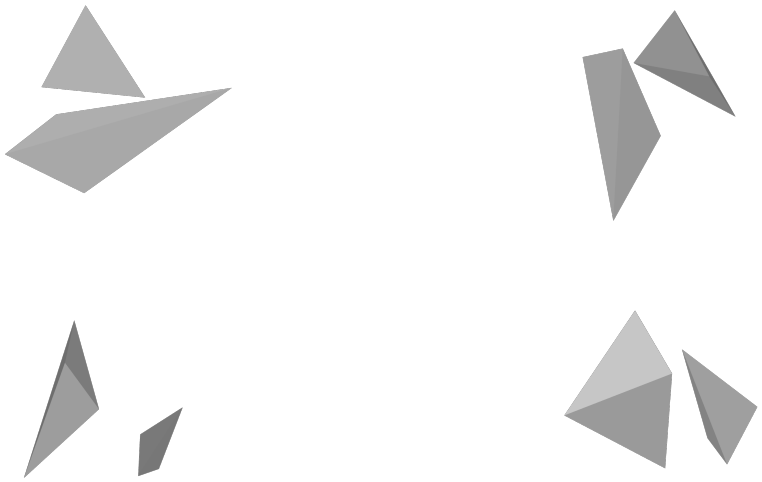

Figure 4.1: Examples of tetrahedra pairs with no intersection.

Point Contact

The point contact case represents the most geometrically constrained intersection type, where two tetrahedra touch at exactly one vertex.

Algorithm 4.2 Tetrahedra with Vertex Contact

1: Generate T_1 arbitrarily
2: Place one vertex of T_2 on a face of T_1
3: Compute face normal $\vec{n}$ and ensure outward orientation: if $\vec{n} \cdot (\text{apex} - \text{face\_vertex}) > 0$, then $\vec{n} \leftarrow -\vec{n}$
4: Construct local spherical coordinate system with $\vec{n}$ as z-axis
5: Sample remaining vertices using spherical coordinates $(\theta, \phi) \in [-\pi/2 + \epsilon, \pi/2 - \epsilon]$
6: For each vertex: calculate r_{max} for direction, sample $r \in [\epsilon, r_{max}]$
7: Transform to global coordinates and validate vertices lie outside T_1
8: Verify exactly one-point intersection

The key insight is using a local spherical coordinate system oriented by the face normal to ensure vertices are placed in the correct hemisphere, preventing volumetric overlap while maintaining contact. The algorithm employs timeout mechanisms and epsilon values for numerical stability and constraint satisfaction (Figure 4.2).

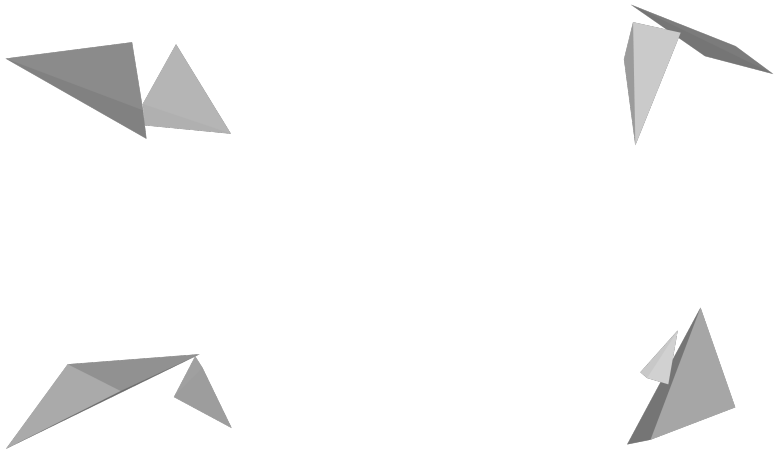

Figure 4.2: Examples of tetrahedra pairs with single-point contact intersections.

Segment Intersection

Algorithm 4.3 Tetrahedra with Edge Contact

1: Generate T_1 arbitrarily
2: Select a triangular face of T_1 and place first vertex of T_2 on this face
3: Generate second vertex of T_2 by projecting a random point onto the same face plane
4: Compute face normal $\vec{n}$ and ensure outward orientation
5: Construct local spherical coordinate system with $\vec{n}$ as z-axis
6: Sample remaining two vertices using spherical coordinates $(\theta, \phi) \in [-\pi/2 + \epsilon, \pi/2 - \epsilon]$
7: For each vertex: calculate r_{max} for direction, sample $r \in [\epsilon, r_{max}]$
8: Transform to global coordinates and validate vertices lie outside T_1
9: Verify intersection restricted to shared edge/segment

The edge intersection algorithm creates a shared line segment between two tetrahedra by placing two vertices of T_2 on the same face plane of T_1. This ensures the tetrahedra intersect along a common edge rather than at a single point. The remaining vertices are positioned using the same spherical coordinate approach as point contact, but with two vertices constrained to the face plane to guarantee linear intersection geometry.

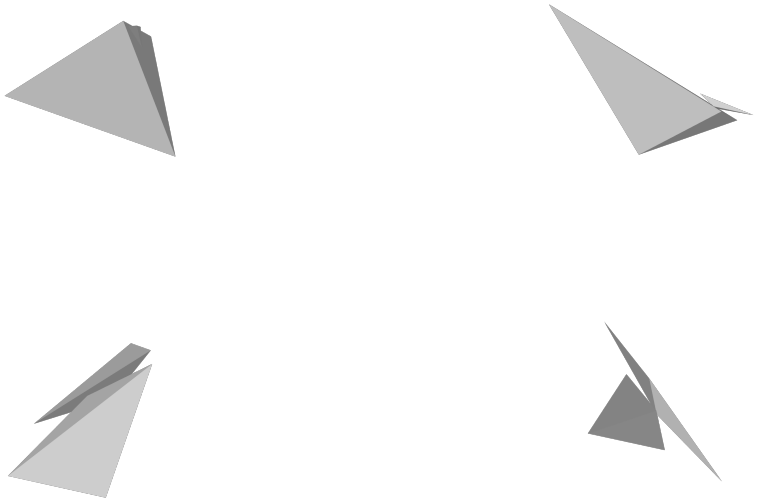

Figure 4.3: Examples of tetrahedra pairs intersecting along a shared edge, showing various orientations and contact configurations.

Polygon Intersection

Algorithm 4.4 Tetrahedra with Face Contact

1: Generate T_1 arbitrarily
2: Select a triangular face of T_1 and define its plane
3: Project three random points onto this face plane to form T_2's triangular face
4: Calculate centroid of the three projected points
5: Compute face normal $\vec{n}$ and ensure outward orientation
6: Construct local spherical coordinate system with $\vec{n}$ as z-axis
7: Sample fourth vertex using spherical coordinates $(\theta, \phi) \in [-\pi/2 + \epsilon, \pi/2 - \epsilon]$
8: Calculate r_{max} from centroid, sample $r \in [\epsilon, r_{max}]$
9: Transform to global coordinates and validate vertex lies outside T_1
10: Verify face contact

The polygon intersection algorithm creates a shared triangular face between two tetrahedra by placing three vertices of T_2 coplanar with a selected face of T_1. The fourth vertex is positioned using spherical coordinates centered at the face centroid, ensuring it lies in the hemisphere opposite to T_1 to prevent volumetric overlap while maintaining face contact.

Figure 4.4: Examples of tetrahedra intersecting precisely over a shared triangular face, demonstrating various orientations and contact configurations.

Polyhedron Intersection

Algorithm 4.5 Sampling tetrahedra with volume intersection via rejection sampling.

1: **repeat**
2: Generate random tetrahedra T_1, T_2
3: Check intersection using `CGAL::do_intersect`
4: **until** Volume overlap detected
5: **return** (T_1, T_2)

Due to the extremely low probability of non-volumetric intersections (e.g., face-face, edge-edge, or vertex-only contact) when uniformly sampling tetrahedra in $[0, 1]^3$, a simple rejection sampling approach suffices. The algorithm keeps generating pairs until an intersection is detected—almost always corresponding to true volumetric overlap, rather than degenerate contact.

This efficiency is primarily due to the convex nature of tetrahedra: any detected intersection between two randomly sampled tetrahedra nearly always involves overlapping volumes.

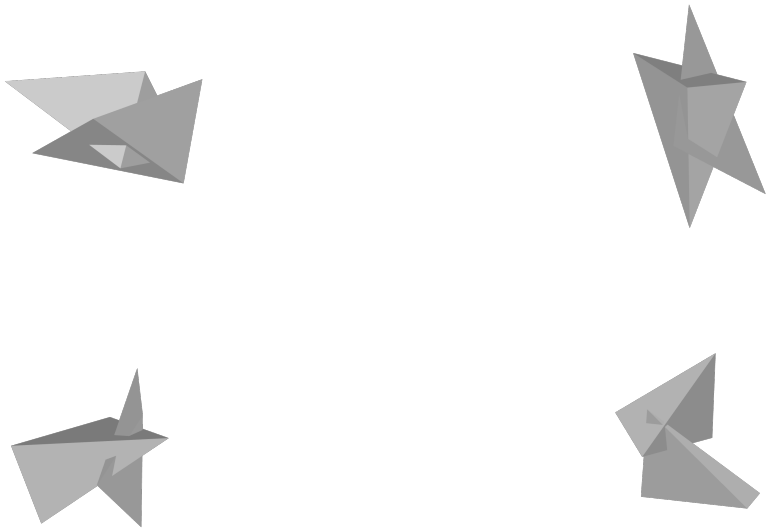

Figure 4.5: Examples of tetrahedra pairs exhibiting volumetric overlap.

This structured framework ensures systematic coverage of all fundamental geometric intersection types, balancing mathematical rigor and computational practicality.

4.1.4 Efficiency

To enable scalable generation of large geometric datasets, the system is optimized for computational performance:

- **C++ Core:** Implemented entirely in C++ for fast execution and efficient memory usage.[2]
- **Parallelism:** The modular pipeline supports multithreading via `std::thread`, enabling concurrent sampling, validation, and labeling with minimal synchronization overhead.
- **CMake Integration:** Structured as a CMake project for portability and straightforward integration with CGAL and other C++ libraries.

4.1.5 System Architecture

The dataset generation system follows the architecture illustrated in Figure A.1 (see Appendix A).

The *ApplicationRunner* manages execution flow and system initialization, while the `Configuration` module centralizes all generation parameters, including dataset size, volume bounds, and intersection distributions. The *DatasetGenerator* serves as the coordination engine, orchestrating sample generation and applying rejection sampling to maintain uniform intersection volume distributions.

Core geometric processing is handled by the *TetrahedronFactory*, which creates random tetrahedron pairs for specific intersection types, and *GeometryUtils*, which performs intersection detection and volume calculations. Output operations are managed by the *BaseWriter* for multi-format serialization and the *ProgressTracker* for runtime monitoring during large-scale generation runs.

[2] C++ enables low-overhead, high-performance computation ideal for geometry processing.

This architecture emphasizes component isolation and configurability, forming a robust foundation for generating large-scale, statistically controlled datasets used in the experimental validation described in the subsequent sections.

4.2 ML Pipeline

The machine learning pipeline transforms raw tetrahedron data into predictive models for geometric intersection analysis through a structured three-stage workflow: data processing, model training, and evaluation. This iterative process continues until achieving a properly tuned model, as illustrated in Figure 4.6.

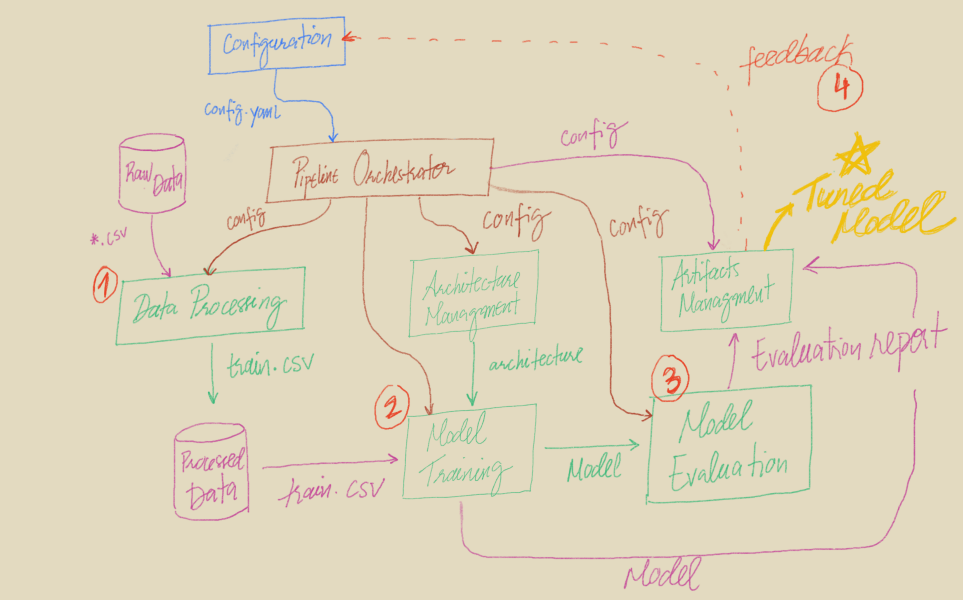

Figure 4.6: Complete machine learning pipeline workflow showing the progression from raw tetrahedron data generation through model training, evaluation, and deployment.

4.2.1 Pipeline Architecture

The pipeline consists of seven key components that handle the complete lifecycle from raw geometric data to deployed models:

CPipelineOrchestrator serves as the central controller, coordinating all pipeline stages through YAML configuration files. It manages data preprocessing, model construction, training, and evaluation with configurable stage skipping and supports both standard training and fine-tuning workflows.

CDataProcessor handles geometric data preprocessing including stratified sampling based on intersection volumes, data augmentation through point permutations and tetrahedron swapping, and geometric transformations for canonical alignment and normalization.

GeometryUtils provides core geometric operations for tetrahedron manipulation, implementing spatial sorting strategies (Morton codes, difficulty-based curriculum learning), coordinate transformations, and vectorized operations for efficient large-scale processing.

CArchitectureManager manages neural network construction, dynamically selecting between specialized architectures (TetrahedronPairNet, MLP, DeepSet, TPNet) and implementing permutation-invariant processing blocks designed for geometric learning tasks.

CModelTrainer executes training with geometric-aware loss functions, curriculum learning integration, and multi-task optimization strategies. It features GPU-accelerated training loops with real-time monitoring and automated visualization generation.

CEvaluator performs comprehensive validation measuring both geometric consistency (permutation invariance) and predictive performance (classification and regression metrics) across different intersection types and hardware platforms.

CArtifactManager manages experimental reproducibility through timestamped artifact storage, model optimization for deployment (TorchScript conversion, C++ export), and structured experiment tracking with configuration versioning.

4.2.2 Core Design Principles

This modular architecture, built around geometric awareness, reproducibility, and comprehensive evaluation, provides the foundation for the extensive experimental validation detailed in the following chapter. The pipeline's flexible design enables rapid iteration through different architectural approaches, data augmentation strategies, and training methodologies—essential capabilities for navigating the complex optimization landscape of geometric learning problems.

What follows is a curated set of the most instructive experiments. This isn't an exhaustive logbook, it's a highlight reel. Each section focuses on a specific dimension of model behavior that revealed useful patterns or helped us avoid dead ends. These insights helped shape both the final model and the thinking behind it.

Chapter 5

Experiments and Estimations

AI/ML engineering isn't about uncovering perfect answers—it's about building things that actually work. It's closer to bridge-building than to math proofs: messy, iterative, and grounded in real-world constraints. There are no universal rules or guaranteed recipes—just tools, trade-offs, and trial-and-error. As Ilya Sutskever aptly said, "The geometric mean of physics and biology is deep learning."[1] Throughout this project, we ran a large number of experiments. Most didn't lead to breakthroughs. Many weren't even formally recorded. But all of them, failures included, played a critical role in developing our design intuition, trimming the configuration space, and steadily improving model performance.

What follows is a curated set of the most instructive experiments. This isn't an exhaustive logbook, it's a highlight reel. Each section focuses on a specific dimension of model behavior that revealed useful patterns or helped us avoid dead ends. These insights helped shape both the final model and the thinking behind it.

5.1 Capacity

Note: The experiments in this section were conducted outside the main ML pipeline described in Section 4.2, using a prototyping setup in Google Colab. The objective was rapid iteration, not production-grade performance. All experiments were implemented in TensorFlow within a single Colab notebook. Results were stored in Google Drive without version control, limiting traceability. The data distribution used here differs from the final pipeline's regime, but these early-stage tests were valuable for identifying general capacity trends and informing architectural design choices.

In neural network design, *capacity* refers to a model's ability to approximate complex functions. It is governed by the number of trainable parameters, architectural depth and width, and activation dynamics [44]. While higher capacity increases representational power, it also raises the risk of overfitting—especially when the training data fails to adequately represent the input space.

These experiments explored how varying model capacity affects learning and generalization. The training dataset contained 1 million examples, with an even 50/50 split between intersecting and non-intersecting tetrahedron pairs. Two disjoint datasets, each with 1 million samples, were used for validation and testing. Performance metrics reported below refer exclusively to the test set.

To reduce variance and improve generalization, the dataset was made large enough to approximate the true input distribution empirically. This leverages the Law of Large Numbers,[2] helping to reveal whether performance limitations stem from underfitting or architectural bottlenecks.

[1] Sutskever, co-founder of OpenAI, alludes here to deep learning being equal parts grounded theory and empirical messiness.

[2] The Law of Large Numbers ensures that as sample size increases, the empirical distribution of data converges to the underlying true distribution—reducing variance and improving generalization.

This section investigates how capacity influences binary classification performance in a Multi-Layer Perceptron (MLP), using Area Under the ROC Curve (AUC) as the performance metric. The primary goal is to identify a configuration that achieves a favorable trade-off between representational power and computational efficiency, and that can later serve as a foundation for multitask learning—particularly when extending to regression tasks such as intersection volume estimation.

Parameter	Value
Data distribution	50% no intersection, 50% polyhedron intersection
Training samples	1,000,000
Validation samples	1,000,000
Test samples	1,000,000
Epochs	20
Batch size	32
Learning rate	0.001
Optimizer	Adam
Loss function	Binary Cross-Entropy
Activation	ReLU

Table 5.1: Fixed experimental settings

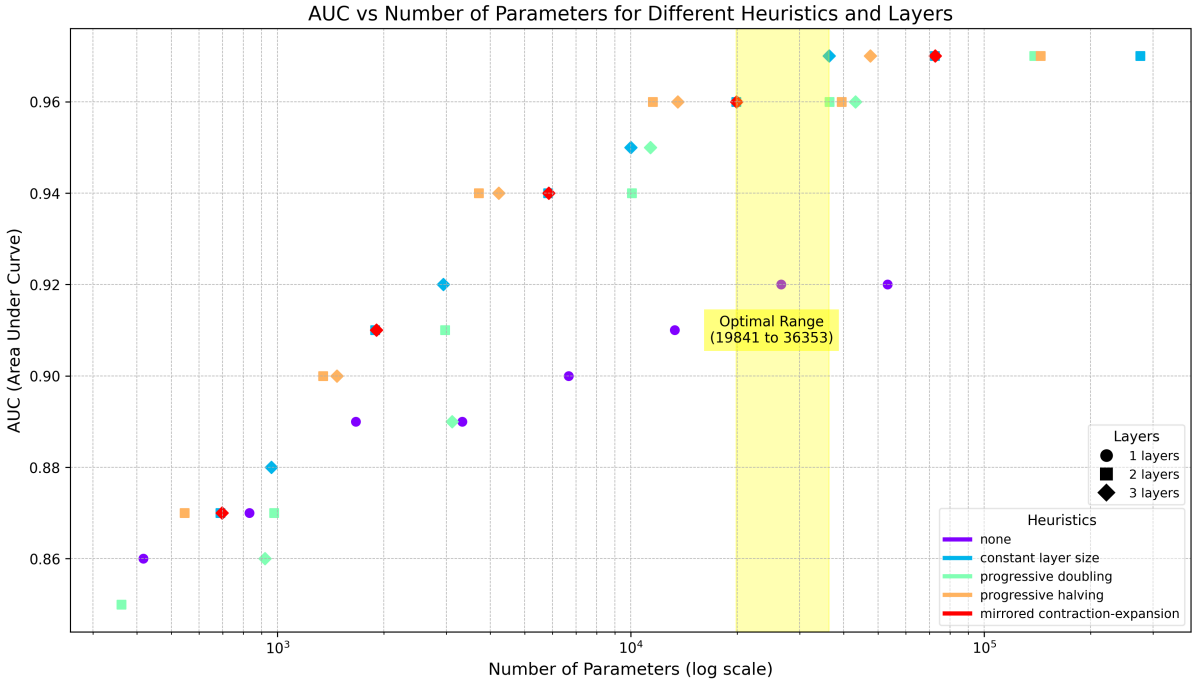

Figure 5.1: AUC vs. Number of Parameters

The architectural search space varied both in depth (number of hidden layers) and width (neurons per layer). Depth was limited to 1–3 hidden layers, and widths ranged from 8 to 2048 neurons. Several common structural heuristics were applied to define multilayer configurations.

Factor	Values
Network depth (hidden layers)	1, 2, 3
Layer width (neurons/layer)	8 to 2048
Heuristics	See Table **??**

Table 5.2: Architectural configurations explored

Heuristic	Example Configuration
Constant size	[64, 64], [128, 128, 128]
Progressive doubling	[32, 64], [64, 128]
Progressive halving	[64, 32], [128, 64, 32]
Mirrored contraction-expansion	[64, 32, 64], [128, 64, 128]

Table 5.3: Layer-sizing heuristics used in the study

5.1.1 Results

The summary of key findings is shown in Table **??**, while detailed trends are plotted in Figure 5.1.

Observation	Insights
Performance improves from 1 to 2 layers	A second hidden layer significantly boosts AUC by enabling more expressive representations.
Minimal gains from 2 to 3 layers	Adding a third layer yields only marginal improvement, suggesting diminishing returns.
Wider layers help more at low capacity	For shallow models, increasing width improves performance more than adding depth.
Progressive halving performs best	This layout consistently produces the best AUC across comparable sizes.
Progressive doubling performs worst	Compresses early features too quickly, likely harming early-stage expressiveness.
Constant and mirrored heuristics are comparable	Both show stable performance under size-matched conditions.
Optimal parameter range is 19k–36k	Balances accuracy, inference cost, and memory footprint effectively.

Table 5.4: Summary of findings on model capacity

These results suggest that architectures with around **19,000–36,000 parameters** and two hidden layers strike the best trade-off between performance and efficiency. While larger models (e.g., 100k+ parameters) may offer slightly higher AUCs, the marginal gains rarely justify the increased computational burden.

From a practical standpoint, despite the limited generability due to training and testing on just two dataset generation strategies—the results demonstrate that with 1 million training samples, it is possible to reach a discriminatory performance near 97% AUC. This indicates that even simple feedforward networks can achieve high discriminatory power over geometrically complex inputs, provided the input representation and data volume are adequate.

Note: Unless stated otherwise, from this point on all experiments were conducted on a Linux system (Ubuntu 20.04, kernel 5.15) with an Intel i7-12700H CPU (2.7 GHz), 8 logical cores, 7.5 GB RAM, and an NVIDIA GeForce MX330 GPU (2 GB VRAM, CUDA 12.1). The ML pipeline described in Section 4.2 was implemented in Python 3.8 using PyTorch 2.3.

Results are available at `https://github.com/ErendiroPedro/tetrahedron_pair_ML`*; each experiment is fully traceable with version-controlled configurations. Note that due to random weight initialization, identical pipeline configurations may yield slightly different results. However, repeated runs produce consistent average performance.*

5.2 Data Types Distributions

To understand how different data generation methods affect model performance and generalization, we ran a series of controlled experiments. Building on the capacity analysis in Section 5.1, which indicated an optimal model size around 20K–36K parameters, we selected a Multi-Layer Perceptron (MLP) with three hidden layers of 128 neurons each ([128, 128, 128]), totaling approximately 36K parameters.

The objective here is to evaluate how the composition of training data influences generalization, particularly across varying intersection types. Each experiment utilizes a training dataset with a distinct distribution of intersection categories (3.2.2), while keeping the training and validation sizes fixed at 100,000 samples each. This setup ensures that any performance differences stem from the nature of the data distribution itself rather than the dataset size. The experimental parameters remain consistent and are listed in Table **??**.

Parameter	**Value**
Model & Architecture	
Model Type	Multi-Layer Perceptron (MLP)
Topology	3 hidden layers, 128 neurons each
Total Parameters	∼36,000
Activation Function	ReLU
Data & Samples	
Training Samples	100,000 (composition varies per experiment)
Validation Samples	100,000
Test Samples	5 distinct sets of 100,000 samples each
Training Parameters	
Epochs	20
Batch Size	32
Optimizer	AdamW
Learning Rate	0.001
Loss Function	Binary Cross-Entropy With Logits

Table 5.5: Fixed experimental parameters for evaluating data generation strategies.

In the paper [57], similar hyperparameter settings were recommended.

5.2.1 Results

Each training distribution is represented as a five-element tuple, denoting the percentage of samples from: (No Intersection, Point Intersection, Segment Intersection, Polygon Intersection, Polyhedron Intersection). The trained models were then evaluated across five corresponding test sets. Performance results are summarized in Table **??**.

Training Distribution	No Int.	Point Int.	Segment Int.	Polygon Int.	Polyhedron Int.	Mean Accuracy
Initial Baseline Experiments						
(50,50,0,0,0)	0.9706	0.9789	0.9370	0.3542	0.0359	0.6553
(50,0,50,0,0)	0.9699	0.9739	0.9777	0.4219	0.0762	0.6839
(50,0,0,50,0)	0.9452	0.8975	0.8985	0.8730	0.3367	0.7902
(50,0,0,0,50)	0.9811	0.0760	0.1880	0.2728	0.8688	0.4773
Hypothesis 1: Balanced mix improves overall accuracy						
(50,10,15,20,5)	0.9571	0.9403	0.9437	0.7964	0.6591	0.8593
Result: +6.9% vs baseline; Polygon/Polyhedron remain weakest						
Hypothesis 2: Increase complex intersection data						
(50,5,7,22,16)	0.9326	0.9220	0.9298	0.8262	0.8494	0.8920
Result: +3.8% vs H1; Trade-off between simple and complex types						
Hypothesis 3: Further increase complex weighting						
(50,4,5,23,18)	0.9384	0.9198	0.9191	0.8314	0.7880	0.8793
Result: -1.3% vs H2; Diminishing returns observed						
Hypothesis 4: Optimized distribution						
(50,5,7,20,18)	0.8912	0.9591	0.9630	0.8907	0.9012	**0.9210**
Result: +2.9% vs H2; Achieved peak performance						

Table 5.6: Model accuracy on five test sets across training data distributions showing iterative hypothesis testing and refinement.

Intersection Type	Best Baseline	Hypothesis 2	Hypothesis 4
No Intersection	**0.9706**	0.9326	0.8912
Point Intersection	**0.9789**	0.9220	0.9591
Segment Intersection	**0.9777**	0.9298	0.9630
Polygon Intersection	0.8730	0.8262	**0.8907**
Polyhedron Intersection	0.8688	0.8494	**0.9012**
Mean Accuracy	0.7902	0.8920	**0.9210**

Table 5.7: Performance comparison across key experimental configurations.

This iterative search highlights a clear trajectory toward improved model generalization. The final distribution, (50,5,7,20,18), achieved the highest mean accuracy over the five datasets of **0.9210**, while maintaining a minimum per-category accuracy above 0.89. This indicates balanced learning across all intersection types.

Remarkably, the model trained on this mixed distribution often outperformed models trained exclusively on individual categories, achieving better polygon and polyhedron accuracy than those trained on 50% pure data from those types. This suggests the model learns more generalizable geometric features rather than overfitting to category-specific patterns.

Based on this evidence, the **(50,5,7,20,18)** training distribution will serve as the default in all subsequent experiments[3].

With the classification tasks addressed, we now turn our attention to the regression problem.

5.3 Volume Sampling Strategy

For geometric intuition, consider that the volume of a unitary tetrahedron—one whose vertices define a maximal-volume configuration inside the unit cube—is $\frac{1}{6} \approx 0.1667$, or roughly 17% of the cube's volume. This serves as a theoretical upper bound for intersection volume. Such configurations are extremely rare in practice, requiring near-perfect overlap of the two tetrahedra. As a geometric reference, a unit cube can accommodate exactly six non-overlapping unitary tetrahedra.

In contrast, the *expected* volume of a randomly sampled tetrahedron inside the unit cube is approximately 0.0138 [54], or about 1.4% of the cube's volume. When two such tetrahedra intersect, their intersection volume is typically much smaller—since both large volume and precise spatial alignment must occur simultaneously. Empirical results show that intersection volumes greater than $2 \cdot 10^{-2}$ are exceedingly rare and usually require deliberate sampling. On the other end, volumes smaller than $5 \cdot 10^{-5}$ are underrepresented due to sampling difficulties at fine spatial resolution.

This volume range has direct implications for the regression task. Very small intersections become indistinguishable to the model if not properly represented, effectively limiting the minimum volume the model can learn to resolve. Thus, this range sets the resolution floor for the regression task.

To ensure numerical stability and effective learning, we constrain all intersection volumes to lie within $[10^{-7}, 10^{-2}]$. The lower bound of 10^{-7} is selected to remain above the numerical precision threshold of `float32`. Any values outside this range are clipped or scaled.

This raises a natural question: *How can we ensure that the model generalizes well to larger volumes, despite their statistical rarity in the training data?*

The key lies in the geometric concept of *similarity*. Just as all triangles remain similar under uniform scaling,[4] tetrahedra also preserve their shape under isotropic transformations.[5] As long as shape and spatial relationships are preserved, the model can generalize across different volume scales—even for rarely seen large-volume cases—because their relative configurations resemble those seen during training.

To train the regression head, we need to sample intersection volumes across a wide range. However, a *linearly uniform* sampling strategy is problematic: it disproportionately favors high-volume intersections and severely under-samples the low-volume regime. This occurs because the interval $[10^{-7}, 10^{-2}]$ spans five orders of magnitude, and linear sampling distributes samples uniformly in absolute terms—not on a logarithmic scale. As a result, most sampled volumes cluster near the upper bound (around 10^{-2}), while the lower bound (around 10^{-7}) is sparsely populated.

This imbalance introduces two issues. First, the model is underexposed to small-volume cases, which are both more common in real-world data and more sensitive to small perturbations. Second, learning accurate regression targets in this low-volume region becomes numerically harder due to the limited dynamic range of `float32`, which reduces precision at smaller scales. Without adequate data coverage in this range, the model struggles to generalize and maintain stability near the lower limit.

[3] Alternative ballparked configuration, such as (43,3,5,22,27), also yielded strong results (minimum accuracy 0.83, overall 0.89) and merit future exploration.

[4] Two triangles are similar if their corresponding angles are equal and their sides are proportional—this holds even when one is a scaled version of the other.

[5] A tetrahedron remains similar to itself under any uniform scaling operation (e.g., doubling the distance between all vertices), meaning its relative geometry is unchanged.

To mitigate this, we adopt a **log-uniform**[6] sampling strategy. Instead of dividing the range into linearly spaced bins, we partition the *logarithmic* volume scale into equal-width intervals. This ensures each order of magnitude is uniformly sampled, balancing representation across small and large volumes. However, additional techniques—such as targeted augmentation—may still be required to ensure full coverage, especially near the extremes.

To implement log-uniform sampling, we discretize the $\log_{10}$-transformed volume range into 100 bins. Each bin spans equal width in log space, ensuring balanced exposure across all orders of magnitude. The choice of 100 bins provides a high-resolution approximation of the continuous log-uniform distribution while keeping computational overhead manageable; this value can be treated as a tunable hyperparameter depending on the desired granularity. In instances where the raw data is too sparse to populate all bins according to this scheme, we apply targeted geometric augmentations to generate new, valid samples, thereby enforcing the desired log-uniform distribution across the entire volume range.

5.3.1 Algorithmic Implementation

To formalize our sampling approach, we define two distinct strategies. The first is a baseline Linear Uniform sampler, which serves to highlight the data imbalance issue. The second is our proposed Log-Uniform sampler, which incorporates a multi-tiered augmentation system to guarantee full data coverage across all orders of magnitude.

Linear Uniform Sampling Strategy. Algorithm 5.1 describes the process for sampling uniformly in linear space. This method creates bins of equal absolute width (e.g., $[0, 0.001], [0.001, 0.002], \dots$) and fills them to a target count, oversampling from neighboring bins if any bin remains under-populated.

Algorithm 5.1 Linear Uniform Volume Sampling

procedure UniformVolumeSample($source, V_{range}, n_{bins}, N_{train}, N_{val}$)
▷ Initialize with linear-space parameters
    $V_{min}, V_{max} \leftarrow V_{range}$
    $bin\_edges \leftarrow \text{linspace}(V_{min}, V_{max}, n_{bins} + 1)$

▷ Phase 1: Setup and Initial Collection
    Calculate target counts $(T_{train,i}, T_{val,i})$ for each bin i.
    Initialize empty lists: $S_{train}[i]$ and $S_{val}[i]$ for each bin $i = 1 \dots n_{bins}$.
    Collect all available real samples from *source* into their respective linear bins, up to the target counts.

▷ Phase 2: Handle Unfilled Bins with Oversampling
    for $i = 1 \dots n_{bins}$ **do**
        if bin i is not full **then**
            $N_{needed} \leftarrow$ calculate needed samples for bin i.
            $S_{source} \leftarrow$ collect samples from adjacent bins $(\pm 1, \pm 2, \dots)$.
            if S_{source} is insufficient **then**
                $S_{source} \leftarrow$ collect samples from **all** other populated bins.
            end if
            Oversample N_{needed} items from S_{source} and add them to bin i.
        end if
    end for

▷ Phase 3: Finalize and Write Output
    Flatten, normalize, and write final samples to output files.
    return final files and counts.
end procedure

[6] Log-uniform sampling selects values such that each step is a constant multiplicative factor. For example, this ensures the number of samples between 10^{-4} and 10^{-3} is the same as between 10^{-3} and 10^{-2}, ensuring all orders of magnitude are equally represented.

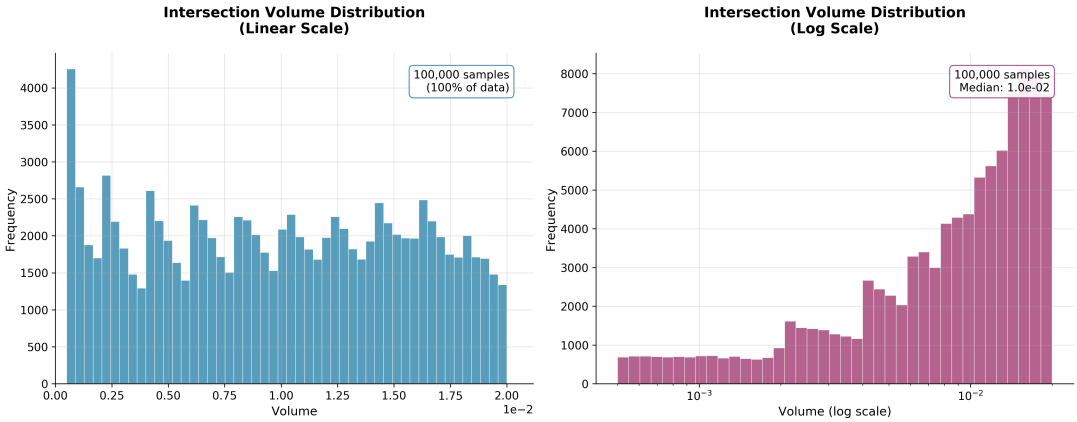

Figure 5.2: Linear uniform sampling strategy results. The figure compares the same data visualized in linear and logarithmic scales. The distribution shows oversampling toward high-volume regions, leading to imbalanced coverage across the volume spectrum.

Log-Uniform Sampling Strategy. Algorithm 5.2 details our primary strategy. It operates in log-space to ensure each order of magnitude receives equal representation. When real data is insufficient to populate a bin, it activates a cascade of augmentation techniques (Algorithm **??**), prioritizing geometrically similar samples before creating new ones.

Algorithm 5.2 Log-Uniform Volume Sampling with Geometric Augmentation

procedure LogUniformVolumeSample($source, V_{range}, n_{bins}, N_{train}, N_{val}$)
▷ Initialize with log-space parameters
    $\log_{min} \leftarrow \log_{10}(V_{range}[0])$; $\log_{max} \leftarrow \log_{10}(V_{range}[1])$
    $bin\_edges \leftarrow \text{logspace}(\log_{min}, \log_{max}, n_{bins} + 1)$

▷ Phase 1: Exhaustive Collection of Real Samples
    Calculate target counts $(T_{train,i}, T_{val,i})$ for each bin i.
    Initialize empty lists: $S_{train}[i]$ and $S_{val}[i]$ for each bin $i = 1 \dots n_{bins}$.
    Collect all available real samples from *source* into their respective log-bins, up to the target counts.

▷ Phase 2: Intelligent Geometric Augmentation
    for $i = 1 \dots n_{bins}$ **do**
        if bin i is not full **then**
            $N_{needed} \leftarrow$ calculate needed samples for bin i.
            $A_{samples} \leftarrow$ empty list for augmented samples.

            // Strategy 1: Augment from within the bin
            Add GenerateGeometricVariants($S_{train}[i] \cup S_{val}[i], N_{needed}$) to $A_{samples}$.

            // Strategy 2: Borrow from adjacent bins
            if $|A_{samples}| < N_{needed}$ **then**
                Add BorrowFromAdjacentBins(i, S, N_{needed}) to $A_{samples}$.
            end if

            // Strategy 3: Handle empty bins by oversampling
            if bin i was empty **and** $|A_{samples}| < N_{needed}$ **then**
                Add OversampleFromNearestBin(i, S, N_{needed}) to $A_{samples}$.
            end if

            Add samples from $A_{samples}$ to $S_{train}[i]$ and $S_{val}[i]$ to meet targets.
        end if
    end for

▷ Phase 3: Finalize and Write Output
    Flatten, normalize, and write final samples, then **return** .
end procedure

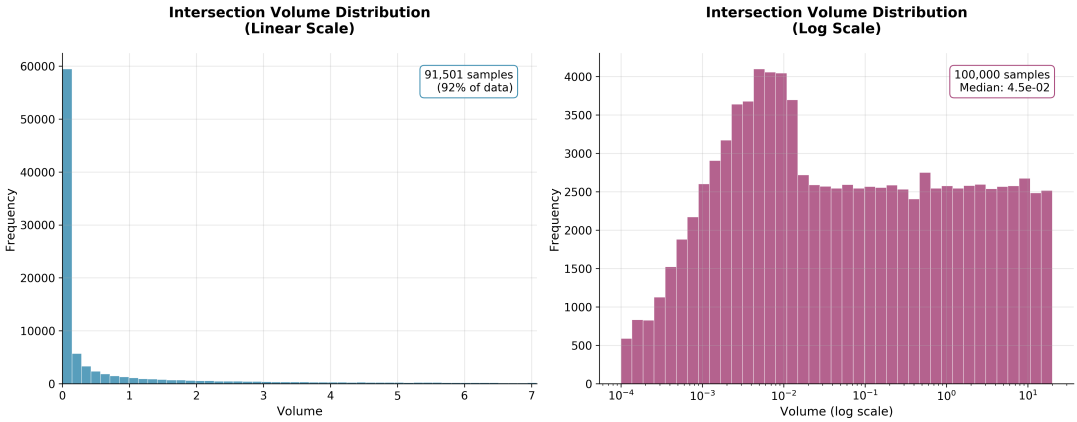

Figure 5.3: Log-uniform sampling strategy results. The figure compares the same data visualized in linear and logarithmic scales. Log-uniform sampling achieves significantly more balanced coverage across volume magnitudes, with a more even distribution in log space. However, perfect uniformity is constrained by the limitations of the underlying raw data.

5.3.2 Results

Table **??** summarizes the key characteristics of different volume sampling approaches, while (see Fig 5.3) provides visual evidence of their distribution properties. The comparison demonstrates why log-uniform sampling is essential for balanced training across the five-order-of-magnitude volume range.

Strategy	Scale	Key Characteristics	Primary Use
Linear Uniform	Linear	Concentrates 95% of samples in upper decade (10^{-3} to 10^{-2}); severely undersamples low volumes	Baseline showing absolute space bias
Linear Uniform	Log	Exposes extreme sparsity in lower decades; confirms magnitude-based sampling imbalance	Diagnostic visualization of coverage gaps
Log Uniform	Log	Achieves the most equal representation across all decades; maintains consistent sample density in log space	Optimal training distribution

Table 5.8: Volume Sampling Strategies and Their Characteristics

Linear uniform sampling allocates approximately 90% of training samples to volumes above 10^{-3}, leaving the critical low-volume regime (10^{-7} to 10^{-4}) with insufficient coverage for robust learning. This imbalance directly impacts model performance, as small-volume intersections—which are both more common in practice and more sensitive to numerical precision—become poorly represented.

In contrast, log-uniform sampling distributes samples evenly across each order of magnitude, ensuring that the model receives adequate exposure to the full spectrum of intersection volumes. This balanced approach is crucial for maintaining regression accuracy across the entire dynamic range, particularly in the numerically challenging low-volume regime where `float32` precision becomes a limiting factor.

Despite the improvements offered by log-uniform sampling, this strategy alone is insufficient to fully mitigate the challenges of learning in the low-volume regime.

5.4 Volume Scaling Factor

RMLSE 3.5.2 is well-suited for zero-inflated targets and outlier robustness. However, when most target volumes are extremely small—as in our dataset—it induces distorted gradients. Training on raw intersection volumes led to model outputs clustered around zero or the mean of the lowest bin. This behavior stems from the logarithmic compression of low values, which penalizes overestimates more harshly than underestimates, biasing the model toward underprediction.

To address this, we apply scalar amplification to target volumes prior to training. Scaling shifts magnitudes into a range where RMLSE provides stronger and more informative gradients.

We evaluate the impact of different scaling factors on model convergence and generalization. Using the same MLP architecture as in prior sections (three hidden layers of 128 units each, 36K parameters), we train on 100K polyhedron-intersection samples per split. Only the scaling factor applied to the target volumes is varied.

Scaling Factor	Min Volume	Max Volume	Description
10^0	1.7×10^{-7}	2.0×10^{-2}	No scaling
10^3	1.7×10^{-4}	20.0	Moderate
10^4	1.7×10^{-3}	200.0	Aggressive

Table 5.9: Scaling factors and resulting volume ranges

Parameter	Value
Model	MLP ([128, 128, 128]), ReLU
Samples	100K train / 100K val / 100K test (polyhedra only)
Sampling	Log-linear uniform
Optimizer	AdamW (lr = 0.001)
Training	20 epochs, batch size 32
Loss	Root Mean Log Squared Error (RMLSE)

Table 5.10: Fixed configuration for volume prediction

5.4.1 Results

Aggregate Performance

Scaling Factor	MAE (Test)	Bin Accuracy	Observation
10^0 (None)	**0.0005**	0.1113	Predicts mean of smallest bin
10^3	0.0024046	**0.18895**	Best overall generalization
10^4	0.0031896	0.14983	Poor large-volume performance

Table 5.11: Overall performance under different scaling regimes

Granular Performance: Interval $[0, 0.01]$

The following tables provide a fine-grained breakdown of prediction accuracy across this low-volume interval.

Interval	MAE	Samples	Correct Pred.	Bin Acc.
0.00000–0.00111	0.0005559	11,130	11,130	1.000
0.00111–0.01000	Gets better as predictions approach zero; >88K samples, 0 correct			

Table 5.12: Interval-wise performance for 10^0 (No scaling)

Interval	MAE	Samples	Correct Predictions	Bin Accuracy
0.00000–0.00111	0.0005559	11,130	11,130	1.000
0.00111–0.01000	—	>88,000	0	0.000

Table 5.13: Interval-wise performance for 10^0 (No scaling)

Interval	MAE	Samples	Correct Predictions	Bin Accuracy
0.00000–0.00111	0.001020	11130	7845	0.705
0.00111–0.00222	0.001615	11124	2035	0.183
0.00222–0.00333	0.001871	10972	1836	0.167
0.00333–0.00444	0.002066	11119	1749	0.157
0.00444–0.00556	0.002333	11135	1517	0.136
0.00556–0.00667	0.002602	11138	1280	0.115
0.00667–0.00778	0.002950	11121	1089	0.098
0.00778–0.00889	0.003373	11128	866	0.078
0.00889–0.01000	0.003805	11133	678	0.061

Table 5.14: Interval-wise performance for 10^3 (Best observed performance)

Interval	MAE	Samples	Correct Predictions	Bin Accuracy
0.00000–0.00111	0.000604	11130	9209	0.827
0.00111–0.00222	0.001187	11124	2226	0.200
0.00222–0.00333	0.001793	10972	1113	0.101
0.00333–0.00444	0.002454	11119	747	0.067
0.00444–0.00556	0.003118	11135	533	0.048
0.00556–0.00667	0.003788	11138	431	0.039
0.00667–0.00778	0.004519	11121	308	0.028
0.00778–0.00889	0.005239	11128	243	0.022
0.00889–0.01000	0.005983	11133	173	0.016

Table 5.15: Interval-wise performance for 10^4 (Over-scaled regime)

The unscaled configuration (10^0) consistently predicts near-zero values due to the extremely narrow raw volume range. This leads to under-performance in higher-volume intervals, where the model fails to generalize beyond its dominant prediction mode.

At the other extreme, scaling by 10^4 yields initially high bin accuracy (82.7% in the first interval) but rapidly declines in performance, with accuracy dropping below 2% in subsequent bins. This behavior indicates that excessive scaling amplifies small volumes disproportionately, biasing the model toward minimal predictions and severely impairing its capacity to learn the full volume distribution.

Moderate scaling (10^3) offers the best trade-off. It retains strong performance in the low-volume regime (70.5% accuracy in the first bin) while preserving resolution across later intervals. This balanced representation leads to more stable convergence and improved generalization across the entire spectrum.

5.5 Combined MLP

We now move to a model that can do perform both tasks at once. Our architecture is a multi-layer perceptron (MLP) that starts with a shared encoder to process all input features. After this shared part, the network splits into two separate "heads", one for classification (intersection or not) and another for regression (predicting volume). We train it on the previous analyzed configurations. The model architecture follows below

Component	**Layer Sizes**
Shared Representation	[128]
Classification Head	[128, 1]
Regression Head	[128, 1]
Activation	ReLU (except final layers)

Table 5.16: Combined model architecture (∼36K parameters).

the classification head gives raw output scores (logits) used in binary cross-entropy loss ('BCEWithLogits'). The regression head gives a single number representing the predicted volume

To train the model, we combine both objectives equally:

$$\mathcal{L}_{\text{combined}} = 0.5 \cdot \mathcal{L}_{\text{BCEWithLogits}} + 0.5 \cdot \mathcal{L}_{\text{RMLSE}}$$

This balanced loss encourages the model to do well at both tasks—without favoring one too much.

Setting	**Value**
Data Settings	
Data Type Distribution	(50, 5, 7, 20, 18) Volume Scaling 10^3
Volume Binning	100 bins
Train / Validation / Test	100k / 100k / 100k (intersection cases only)
Training Parameters	
Epochs	20
Batch Size	32
Optimizer	AdamW
Learning Rate	0.001
Loss Function	$0.5 \cdot \text{BCE} + 0.5 \cdot \text{RMSLE}$

Table 5.17: Training configuration for the combined model.

5.5.1 Results

In general, the combined model is capable of performing both tasks simultaneously, with a performance drop observed in each. Two key differences from previous experiments should be noted:

1. The current model predicts two outputs, which may require additional capacity to effectively handle both tasks.

Metric	Value	Observation
Classification Accuracy	0.8574	Worse than single-task baseline
MAE (Intersection Volume)	0.00388	Worse than single-task baseline
Mean Bin Accuracy (10 bins)	0.12842	Worse than single-task baseline

Table 5.18: Test performance of the baseline model.

2. For the regression component, earlier tests utilized 100K intersecting samples, whereas the current setting includes only 18K samples (i.e., 18% of the data).

In the following sections, we explore whether scaling the model architecture or adjusting the data distribution can help recover or even improve performance.

5.6 Model Scaling

To investigate the impact of capacity on multi-task learning, we trained a larger model containing approximately **70k parameters**. This architecture features a shared representation block followed by wider task-specific heads for classification and regression. All training procedures and data configurations remain consistent with prior experiments.

Component	Layer Sizes
Shared Representation	[128]
Classification Head	[128, 128, 1]
Regression Head	[128, 128, 1]
Activation	ReLU (except final layers)

Table 5.19: Scaled-up model architecture (∼70K parameters).

5.6.1 Results

Metric	Value	Performance
Classification Accuracy	**0.8592**	Better than baseline, worse than single-task
MAE (Intersection Volume)	0.0035955	Better than baseline, worse than single-task
Mean Bin Accuracy (10 bins)	0.13819	Better than baseline, worse than single-task

Table 5.20: Test performance of the scaled-up model

Performance across metrics shows small improvements over the previous multi-task architecture, though still trailing behind dedicated single-task models. Classification accuracy was sometimes higher than the baseline during a few experimentation, but results varied depending on initialization. This variability suggests that constraining the joint loss might help stabilize convergence, and performance is likely to benefit from further data scaling.

The classification head appears to benefit from the joint setup, leveraging shared features shaped by regression supervision. This implicit regularization seems to refine the representation space, leading to more reliable decision boundaries even in a multi-task context.

However, the regression task continues to underperform. The main limitation remains the size of the training data: out of 100K total samples, only ∼18K involve non-zero intersection volumes. This restricts the regression head's exposure to informative gradients, particularly in high-volume regions where samples are rare. As a result, the model struggles to generalize well on the tails of the volume distribution.

5.7 Data Scaling

To mitigate the performance limitations caused by label sparsity—particularly for the regression task—we expanded the training set from 100K to 1 million samples. This yielded approximately 180K intersecting cases (∼18%),

significantly increasing the density of informative samples without altering the underlying data distribution. The model architecture and training protocol remained unchanged to isolate the effect of dataset size.

5.7.1 Results

Metric	Value	Notes
Mean Classification Accuracy	**0.947**	+2.6% absolute improvement over previous best
MAE (Intersection Volume)	0.003198	Still worse than single-task
Mean Bin Accuracy (10 bins)	0.16122	Still worse than single-task

Table 5.21: Performance with 1M training samples.

Scaling the data notably improves classification accuracy and mean bin accuracy, with the latter showing a meaningful gain in predictive granularity across the volume spectrum. The regression mean absolute error also improves slightly, but still not reaching parity with the best earlier results despite the multi-task setting.

These improvements suggest that access to more diverse and frequent intersection examples allows the model to generalize better, particularly on rare volume cases. At the same time, classification performance benefits from richer shared features due to the additional supervisory signals.

Overall, data scaling proves more effective than model scaling alone in mitigating the sparsity-induced bottlenecks inherent to multi-task learning—particularly for the regression head, which benefits directly from denser gradient updates in high-volume regimes.

Importantly, the final combined model not only surpasses the performance of both single-task baselines but also demonstrates a more refined geometric understanding of the decision boundary between intersecting and non-intersecting cases. This suggests that the shared representation, regularized via joint optimization, is capable of capturing richer spatial patterns across tasks than either model could in isolation.

5.8 Inference Speed

To evaluate real-time deployment feasibility, we benchmarked the inference latency of the model on a standard CPU setup (Intel i7-12700H, 2.7 GHz) using single-threaded LibTorch in C++. The evaluation was performed on 100,000 randomly sampled inputs from the polyhedron test set, with results averaged over 50 runs. Latency was measured across different batch sizes to assess scalability.

Batch Size	Time per Sample (ms)	Samples per Second
32	0.016202	61,722
64	0.015306	65,331
1024	0.009205	108,636
2048	0.001534	110,079

Table 5.22: Inference speed benchmarks on CPU (LibTorch, single-threaded).

No performance degradation is observed up to a batch size of 2048. The architecture's simplicity enables consistent sub-millisecond per-sample latency, making it well-suited for real-time pipelines, tight inference loops, and deployment on resource-constrained systems such as embedded CPUs or edge devices. Its compact structure also facilitates low memory overhead and straightforward integration into production systems.

5.9 Raw MLP Overview

The final model architecture is the result of extensive empirical tuning across network capacity, data diversity, binning strategies, and volume scaling. It is a lightweight, multi-task MLP designed for simultaneous intersection classification and volume regression. The figure below illustrates the architecture, followed by a summary table of key parameters. Additional results are presented in Chapter 6.

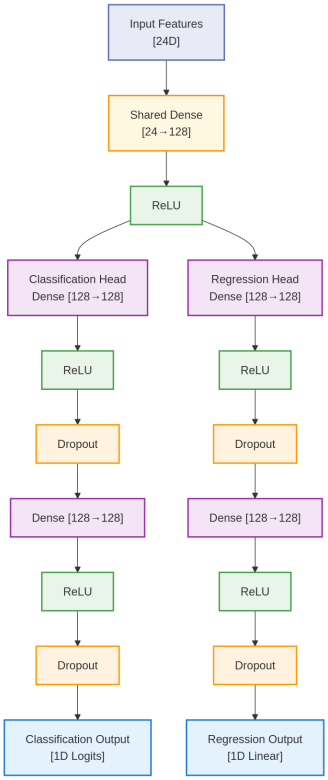

Figure 5.4: Multi-layer perceptron architecture for dual-task tetrahedron intersection analysis. The network processes two tetrahedra (T_1 and T_2), each represented by 12 input coordinates. A shared feature extraction layer is followed by two specialized heads: one for intersection classification and the other for volume regression.

Aspect	Setting
Architecture	
Shared Layers	[128]
Classification Head	[128, 128, 1] (Logits)
Regression Head	[128, 128, 1] (Linear)
Total Parameters	∼70,000
Training Setup	
Training Size	1M samples
Training Distribution	[50%, 5%, 7%, 20%, 18%]
Volume Range	[1e-7, 1e-2]
Volume Scaling Factor	10^3
Number of Bins	100
Epochs / Batch Size	20 / 32
Optimizer	AdamW (LR = 0.001)
Loss Function	$0.5 \cdot \text{BCE} + 0.5 \cdot \text{RMLSE}$
Performance	
Mean Classification Accuracy (across 5 data types)	**94.70%**
Min Class Accuracy (No Intersection)	**92.04%**
Max Class Accuracy (Polyhedron Intersection)	**97.44%**
Regression MAE (Test, original scale)	**0.003198**
Mean Bin Accuracy (10 bins)	**16.12%**
Latency (CPU, 1 thread, batch of 2048)	**1.53 μs**
Throughput (CPU, 1 thread, batch of 2048)	**110,079 samples/s**
Model Size (.pt)	**579.91 KB**
Model Weights (JSON)	**100.95 KB**

Table 5.23: Final combined MLP model configuration summary.

5.10 Predicate Powered Learning

While brute-force strategies can yield surprisingly competitive results due to the priors we incorporate, they remain computationally inefficient and oblivious to the problem's inherent structure[7]. To build a model that generalizes instead of memorizing, we design our ML to explicitly encode the problem's geometric structure and logical invariants, as detailed in Section 3.1. We begin with the data.

5.10.1 Transformations

we explore geometric transformations that normalize the input domain. These transformations aim to reduce variance, align data with a canonical frame, and embed inductive biases related to spatial relationships. Two key strategies are evaluated:

- **Unitary Tetrahedron Transformation**
- **Principal Axis Transformation**

Unitary Tetrahedron Transformation While a tetrahedron is defined by 4 vertices in $\mathbb{R}^3$, its geometric configuration introduces unnecessary degrees of freedom for learning. Instead of predicting intersections between arbitrary tetrahedra, we map the reference tetrahedron T_1 to a *unitary* frame—a fixed tetrahedron with maximum volume inside the unit cube—thus simplifying the geometric context.

[7] Currently, our model treats the input as a flat list where the first 12 coordinates represent T_1 and the next 12 represent T_2, without awareness of the underlying geometry. We know more about the problem and should leverage that knowledge.

Algorithm 5.3 Unitary Tetrahedron Transformation

Require: Two tetrahedra T_1 (reference), T_2, each defined by 4 vertices in $\mathbb{R}^3$
Ensure: Transformed T_2' relative to unitary frame and volume correction factor
1: **Anchor Geometry:** Select T_1 as the reference frame
2: **Translate:** Center T_1 by subtracting its centroid or anchor vertex
3: **Construct Basis:** Compute edge vectors of T_1 relative to base vertex
4: **Affine Mapping:** Derive transformation matrix A that maps $T_1 \to$ unitary tetrahedron
5: **Apply Transformation:** Compute $T_2' = A \cdot (T_2 - v_0)$, where v_0 is the reference vertex of T_1
6: **Model Input:** Use T_2' as input to the model; omit T_1 (it is fixed in this space)
7: **Volume Recovery:** Multiply model output by $|\det(A)|$ to rescale predicted volume[8]

Why apply this transformation? It reduces learning complexity by reframing the problem: *Does a given tetrahedron intersect a fixed unitary tetrahedron, and what is the volume of that intersection?* This affine transformation preserves intersection relationships and linearly scales volumes, simplifying both classification and regression tasks.

Metric	Value	Notes
Mean Classification Accuracy	**0.9687**	+2.2% improvement over raw input
MAE (Intersection Volume)	**0.0022872**	Better than raw input
Mean Bin Accuracy (10 bins)	**0.20299**	+4.2% improvement over raw input
Total Inference Time	967.56 ms	Includes preprocessing and forward pass
Preprocessing Time	27.2079 ms	∼3% of total time
Forward Pass	940.3521 ms	∼97% of total time
Time per Sample	0.009676 ms	Batch size: 2048

Table 5.24: Performance of MLP after Unitary Tetrahedron Transformation.

This transformation significantly improves regression metrics by reducing geometric variance and focusing the model on essential structural information. The method is efficient and easily vectorized for batch processing.

Principal Axis Transformation An alternative normalization technique involves aligning the input with its *intrinsic* orientation. Principal Component Analysis (PCA) provides a data-driven method to rotate the object into a canonical pose without altering scale.

Algorithm 5.4 Principal Axis Transformation via PCA

Require: Two tetrahedra T_1, T_2, each defined by 4 vertices in $\mathbb{R}^3$
Ensure: Rotated tetrahedra T_1', T_2' aligned to principal axes of T_1
1: Compute centroid of T_1
2: Translate vertices of T_1 and T_2 so centroid of T_1 is at origin
3: Perform PCA on T_1 vertices to extract principal directions (eigenvectors)
4: Construct rotation matrix R from sorted eigenvectors of covariance matrix
5: Rotate $T_1' = R \cdot T_1$ and $T_2' = R \cdot T_2$

This transformation reduces rotational variance, letting the model focus on other geometric relationships.[9]

Applying PCA yields the best overall performance. The canonical orientation helps both regression and classification by exposing a more structured representation to the model.

[9] Unlike the unitary transformation, PCA does not transform the space itself; rather, it provides a specific viewpoint by aligning the data with its principal axis.

Metric	Value	Notes
Mean Classification Accuracy	**0.9700**	+3.3% improvement over raw input
MAE (Intersection Volume)	**0.001738**	Best result overall
Mean Bin Accuracy (10 bins)	**0.25751**	+6.8% improvement over raw input
Total Inference Time	1292.9411 ms	Includes preprocessing and forward pass
Preprocessing Time	182.4222 ms	∼14% of total time
Forward Pass	1110.5188 ms	∼86% of total time
Time per Sample	0.012929 ms	Batch size: 2048

Table 5.25: Performance of MLP after Principal Axis Transformation.

Comparison. Both transformations significantly outperform the raw input baseline, each with distinct trade-offs. The Unitary Tetrahedron method is computationally efficient and effectively reduces affine variance, making it suitable for latency-sensitive scenarios. In contrast, the Principal Axis Transformation incurs higher computational cost but consistently delivers superior accuracy, especially for regression tasks. Additionally, because it preserves the original coordinate system, feature representations remain interpretable in the input space. Accordingly, we adopt the Principal Axis Transformation for all subsequent experiments.

Next, we examine how to exploit the geometric structure of the problem within the model architecture itself.

5.10.2 TetrahedronPairNet

TetrahedronPairNet is a neural architecture tailored to infer intersection properties from one or two 3D tetrahedra. Inspired by DeepSets, PointNet, and related architectures 3.3, it is explicitly constructed to learn functions that are *permutation-invariant* with respect to both vertex ordering and tetrahedron pairing. This inductive bias ensures that the model respects geometric symmetries inherent in the input domain.

The architecture can be extended to encode additional structural priors—such as symmetry, translation invariance, and locality—through the integration of geometric transformation modules [55, 56].

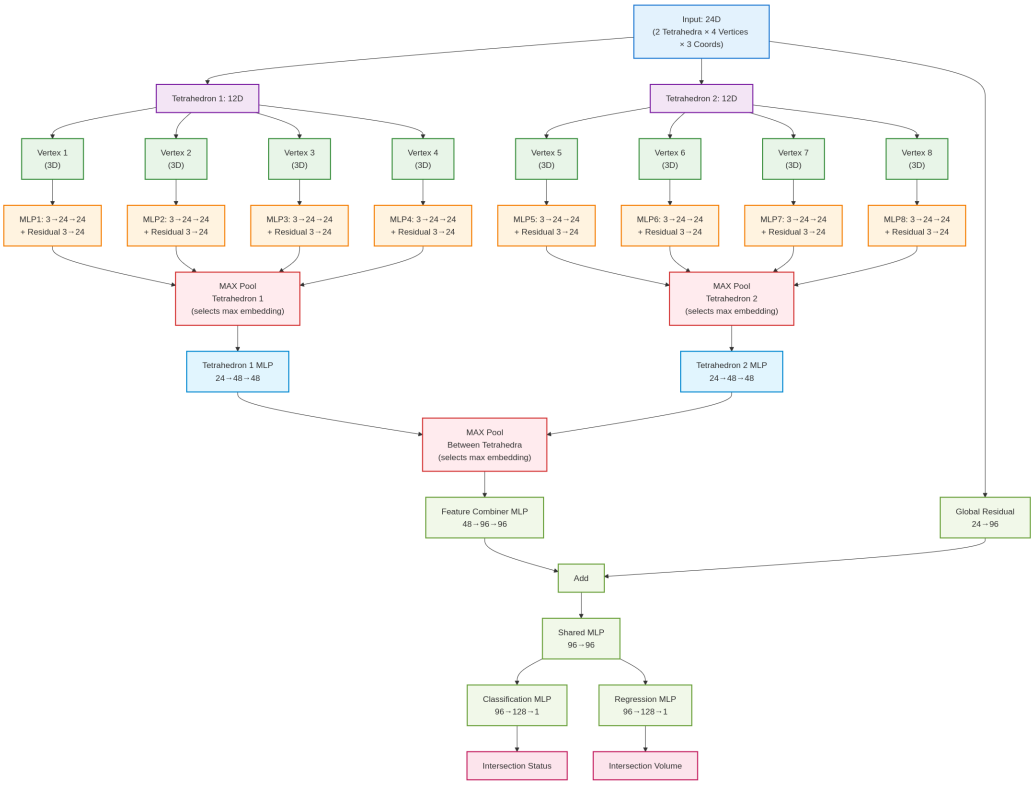

Figure 5.5: Schematic diagram of TetrahedronPairNet (M).

The model operates in three main stages:

- **Per-Vertex Encoding:** Each of the eight input vertices (2 tetrahedra × 4 vertices) is processed independently by a shared MLP. A residual connection from the original 3D coordinates to the final MLP layer enhances geometric fidelity and stabilizes gradient flow.
- **Tetrahedral Pooling:** Encoded vertex features within each tetrahedron are aggregated using a symmetric function—`max` by default—to yield a fixed-length, order-invariant representation. This is passed through a second MLP to capture higher-order intra-tetrahedron interactions.
- **Pairwise Refinement & Global Residual:** For paired inputs, the individual tetrahedron embeddings are concatenated and passed through another MLP to model inter-tetrahedron interactions. In parallel, the raw input (12D or 24D) is projected through a shallow residual path and added to the fused representation. This final embedding is passed through shared processing layers and bifurcated into classification and regression heads.

TetrahedronPairNet is modular and tunable. Aggregation functions (`max`, `sum`, `mean`) and MLP widths/depths can be configured to match task complexity and compute constraints. Unless otherwise stated, the configuration below is used in all experiments:

Variant	Vtx MLPs	Tet MLPs	Pair MLP	Shared	Cls Hd	Reg Hd	Params	Size (KB)
M	[24, 24]	[48, 48]	[96, 96]	[96]	[128, 1]	[128, 1]	∼64K	475.88

Table 5.26: TetrahedronPairNet (M) default configuration.

Metric	Value	Notes
Mean Classification Accuracy	**0.9764**	+0.64% improvement over simple mlp
MAE (Intersection Volume)	**0.001500**	Better than simple mlp
Mean Bin Accuracy (10 bins)	**0.29196**	+3.4% improvement over simple mlp
Total Inference Time	3099 ms	Includes preprocessing and forward pass
Preprocessing Time	804 ms	∼26% of total time
Forward Pass	2294 ms	∼74% of total time
Time per Sample	0.030992 ms	Batch size: 2048

Table 5.27: Performance of TetrahedronPairNet (M) after Principal Axis Transformation.

By explicitly leveraging geometric symmetries within the model architecture, *TetrahedronPairNet* is able to extract more meaningful structure from the input data. As a result, it consistently outperforms a baseline simple MLP in both classification and regression tasks. In particular, the model exhibits significantly higher bin accuracy, reflecting improved discrimination across intersection volume ranges and thus more reliable volume predictions. Despite its richer architecture, inference remains efficient due to batch-optimized processing, and the overall model size is smaller.

Perhaps increasing model capacity and dataset size might further enhance performance, but we are also interested in studying whether other aspects of the modeling pipeline can unlock additional gains.

5.10.3 Augmentations

In this section, we explore augmentation strategies aimed at improving generalization and robustness by manipulating sample order and permutations of the input. While such techniques are common in a lot of data-driven pipelines, their effect is nuanced in the context of geometric intersection tasks.

Sorting

An often-overlooked factor in dataset design is the *order* in which data samples are presented to the model. Both spatial and temporal sequencing can subtly influence learning dynamics and convergence behavior.

We distinguish two principal axes of sorting:

- **X-axis (Spatial Sorting):** Refers to the internal ordering of features within each sample. For example, tetrahedron vertices may be sorted by distance to the origin or by coordinate values. While such heuristics may seem intuitive, they can introduce geometric biases that impair model generalization, especially when the sorting logic encodes implicit assumptions about symmetry or structure.
- **Y-axis (Temporal Sorting):** Refers to the order in which samples are presented during training. Sorting based on scalar attributes—such as intersection volume, centroid distance, or bounding box overlap—can induce curriculum-like effects. While this may accelerate early learning, it risks biasing the model toward overfitting to early-stage patterns and failing to adapt to more complex samples encountered later.

In our experiments, spatial sorting (X-axis) yielded marginal to no performance gains and occasionally increased inference time. Its impact varied with model architecture, suggesting that inductive biases related to feature locality may be model-dependent.

Temporal sorting, especially by intersection volume, consistently led to premature convergence on low-volume or non-intersecting cases. This impaired the model's ability to learn high-overlap scenarios and complex boundary interactions.

Overall, sorting can be considered a tunable hyperparameter, but in our setup, its utility was limited. We adopted randomized sample ordering for all training runs, which provided the most stable and generalizable results across tasks and model sizes.

Permutations

Permutation-based augmentation aims to synthetically increase dataset diversity by reordering elements within each sample (e.g., permuting vertex indices or swapping tetrahedron order). This can, in theory, promote invariance to representation changes and enhance generalization.

However, capacity-limited training runs showed that such permutations underperformed compared to using entirely new synthetic samples. Our data generation pipeline is sufficiently fast and diverse, reducing the need to reuse existing examples via transformation.

When applied, permutation augmentations were restricted to underrepresented regions of the input space to address local overfitting. Despite these targeted uses, we found no consistent benefit compared to baseline training with more synthetic data.

We initially hypothesized that enforcing permutation consistency would improve learning stability. However, the model naturally encounters equivalent permutations across the dataset due to random sampling, which may already encourage sufficient invariance without explicit enforcement.

While we acknowledge that optimal sorting and permutation schemes could theoretically improve performance, **none of the tested strategies yielded meaningful gains**. Therefore, our final configuration excludes both sorting and permutation augmentations in favor of randomized, untransformed sample ordering.

5.11 Hyperparameter Tuning

Throughout our experiments, we adopted a broad definition of hyperparameters, extending beyond model architecture to include data distribution, training dynamics, and pipeline configuration. The system was treated as a set of interdependent tunable parameters, each influencing performance in subtle and often non-linear ways.

Tuning was performed iteratively through manual adjustments to key variables such as dataset complexity, model size, and loss weighting. These changes were guided by empirical observations rather than an exhaustive search, prioritizing rapid iteration over full coverage of the configuration space.

5.11.1 Final Optimization Pass

In the final phase of experimentation, our primary focus was improving regression performance using insights gathered from earlier stages. We began by increasing model capacity, selecting a larger variant, and doubling the dataset size from 1 million to 2 million samples. We also extended the number of training epochs to 50 and increased the batch size to 2048 to accelerate convergence. The goal was to assess how well the current pipeline scaled under more demanding conditions and whether further performance gains could still be unlocked.

Below we detail the configuration and setup used in the final model, along with the corresponding performance metrics.

Variant	Vtx MLPs	Tet MLPs	Pair MLP	Shared	Cls Hd	Reg Hd	Params	Size (MB)
L	[48, 48]	[48, 48]	[48, 48]	[128]	[128, 1]	[128, 1]	∼77K	714.3

Table 5.28: TetrahedronPairNet (L) architecture configuration.

Aspect	**Setting**
Architecture (TetrahedronPairNet L)	
Vertex MLPs	[48, 48]
Tetrahedron MLPs	[48, 48]
Pairwise MLP	[48, 48]
Shared Layers	[128]
Classification Head	[128, 1] (Logits)
Regression Head	[128, 1] (Linear)
Total Parameters	∼77,000
Model Size (.pt)	**714,339 KB**
Training Setup	
Training Dataset	2M samples
Volume Range	$[10^{-7},\ 10^{-2}]$
Volume Scaling Factor	10^3
Epochs / Batch Size	50 / 2048
Optimizer	AdamW (LR = 0.001)
Loss Function	$0.5 \cdot \text{BCE} + 0.5 \cdot \text{RMSLE}$
Performance Metrics	
Mean Classification Accuracy	**98.58%**
MAE (Intersection Volume, Original Scale)	**0.001190**
Mean Bin Accuracy (10 bins)	**0.34832**
Latency (CPU, 1 thread, batch of 2048)	**27,28 μs**
Throughput (CPU, 1 thread, batch of 2048)	**36,651 samples/s**

Table 5.29: Final model training setup and performance summary.

From these experiments (Table 5.29), we observe a consistent improvement across all performance metrics. The larger model and increased dataset size contribute to greater accuracy and lower regression error, albeit with a slight trade-off in inference speed. Provided that sufficient data and computational budget are available, further scaling is expected to yield additional gains, indicating headroom for even more performant models.

With the model architecture and optimization strategy finalized, we now introduce TetrahedraPairDatasetV1, a dataset purpose-built for supervised learning on 3D tetrahedral intersections. It supports both binary intersection classification and continuous volume regression, while preserving geometric variability and offering precise statistical control over the volume distribution.

5.12 TetrahedraPairDatasetV1

`TetrahedraPairDatasetV1` is a carefully curated dataset of 1 million 3D tetrahedron pairs designed for supervised learning on geometric intersection problems. The dataset supports two complementary tasks: binary intersection classification and continuous regression of intersection volume, providing a comprehensive foundation for 3D geometric reasoning.

5.12.1 Design Philosophy and Configuration

The dataset prioritizes geometric interpretability and algorithmic robustness through several key design decisions. No data augmentations, permutations, or vertex reorderings are applied, preserving the natural geometric relationships inherent in tetrahedral configurations. This approach maintains vertex order integrity and avoids coordinate-based sorting, ensuring that models learn from raw, unaltered geometric data.

Core Dataset Properties

Property	Value	Rationale
Size	1,000,000 samples (.csv)	Sufficient scale for deep learning
Coordinate Precision	double	High numerical accuracy
Input Features	24 (12 per tetrahedron)	Complete vertex representation
Labels	2 (binary, continuous)	Multi-task learning support
Intersection Distribution	[50%, 5%, 7%, 20%, 18%]	Balanced learning scenarios
Volume Sampling	`log_uniform_volume`	Wide scale coverage
Volume Binning	100 bins in $[10^{-7}, 0.02]$	Fine-grained volume discretization
Scaling Factor	10^3	Numerical stability
Augmentation	None	Geometric authenticity
Transformation	None	Natural coordinate space

Table 5.30: Core dataset properties and design rationale for `TetrahedraPairDatasetV1`.

Data Structure and Representation

Each sample contains 24 coordinates representing two complete tetrahedra, formatted as $\mathtt{T_t\_V_v}\_\alpha$ where $t \in \{1, 2\}$ identifies the tetrahedron, $v \in \{1, 2, 3, 4\}$ specifies the vertex, and $\alpha \in \{x, y, z\}$ denotes the coordinate axis. This systematic naming convention ensures consistent geometric interpretation across all samples.

The dual-label structure enables both classification and regression tasks: `HasIntersection` provides binary labels (0 for no intersection, 1 for intersection or contact), while `IntersectionVolume` offers continuous targets (0 for touching configurations, > 0 for overlapping volumes).

Component	Description	Details
Features	24 coordinates (12 per tetrahedron)	Format: $\mathtt{T_t\_V_v}\_\alpha$, where $t \in \{1, 2\}$, $v \in \{1, 2, 3, 4\}$, $\alpha \in \{x, y, z\}$
HasIntersection	Binary classification label	0: no intersection; 1: intersection or contact
IntersectionVolume	Continuous regression target	$= 0$ if touching; > 0 if overlapping

Table 5.31: Structure and semantic meaning of individual dataset samples.

5.12.2 Spatial Distribution and Geometric Foundations

Coordinate Space Design

Tetrahedron vertices are uniformly sampled within the unit cube $[0, 1]^3$, creating diverse and unbiased spatial configurations. This uniform distribution ensures comprehensive coverage of the geometric space while maintaining computational tractability.

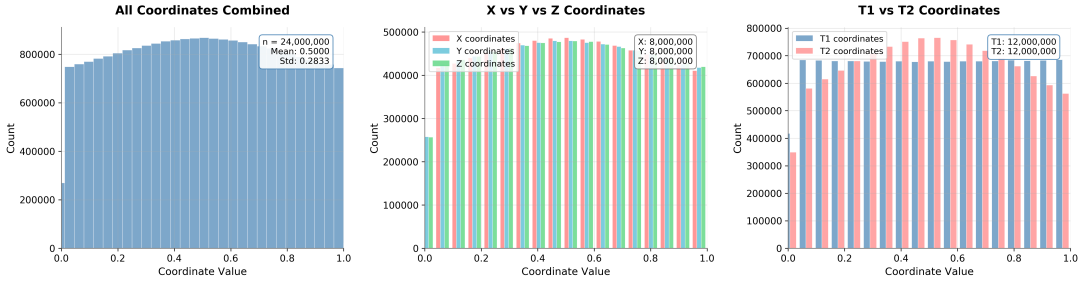

Figure 5.6: Distribution of vertex coordinates for T1 and T2. All coordinates are uniformly distributed in $[0, 1]$, confirming unbiased spatial sampling.

Metric	Expected	Observed
Mean	0.500	0.500040
Standard Deviation	≈ 0.289	0.283301

Table 5.32: Coordinate uniformity validation for tetrahedron vertices. Close alignment between expected and observed statistics confirms quality of uniform sampling process.

Volume Distribution Strategy

The dataset employs log-uniform volume sampling to capture the full spectrum of geometric scales encountered in real-world applications. This distribution strategy addresses the challenge that uniform vertex sampling naturally biases toward smaller volumes, potentially underrepresenting larger geometric structures.

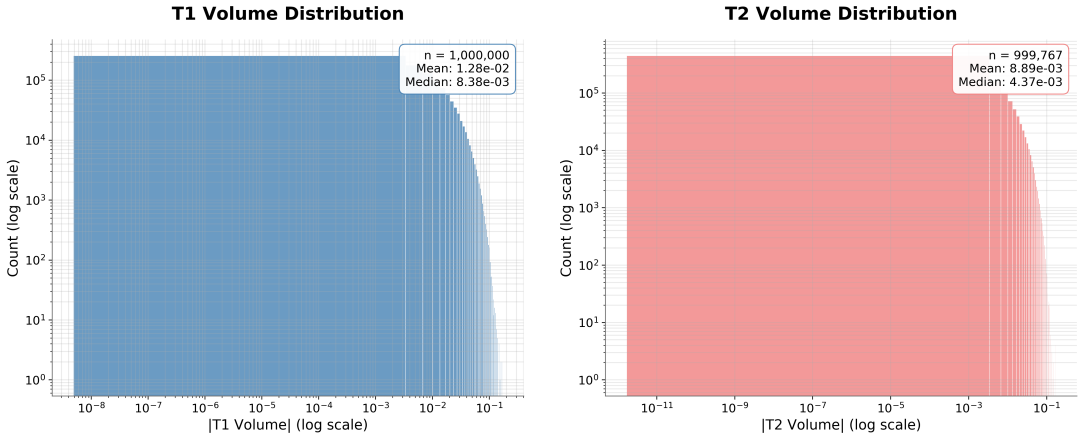

Figure 5.7: Volume distributions for T1 and T2. Log-uniform sampling ensures comprehensive coverage across multiple orders of magnitude, from 10^{-11} to 10^{-1}.

The volume ranges span: T1 volumes in $[10^{-8}, 10^{-1}]$ and T2 volumes in $[10^{-11}, 10^{-2}]$. The joint volume distribution reveals a natural concentration in the upper-right region (10^{-5} to 10^{-1}), indicating that most meaningful geometric interactions occur when both tetrahedra have moderate to large volumes.

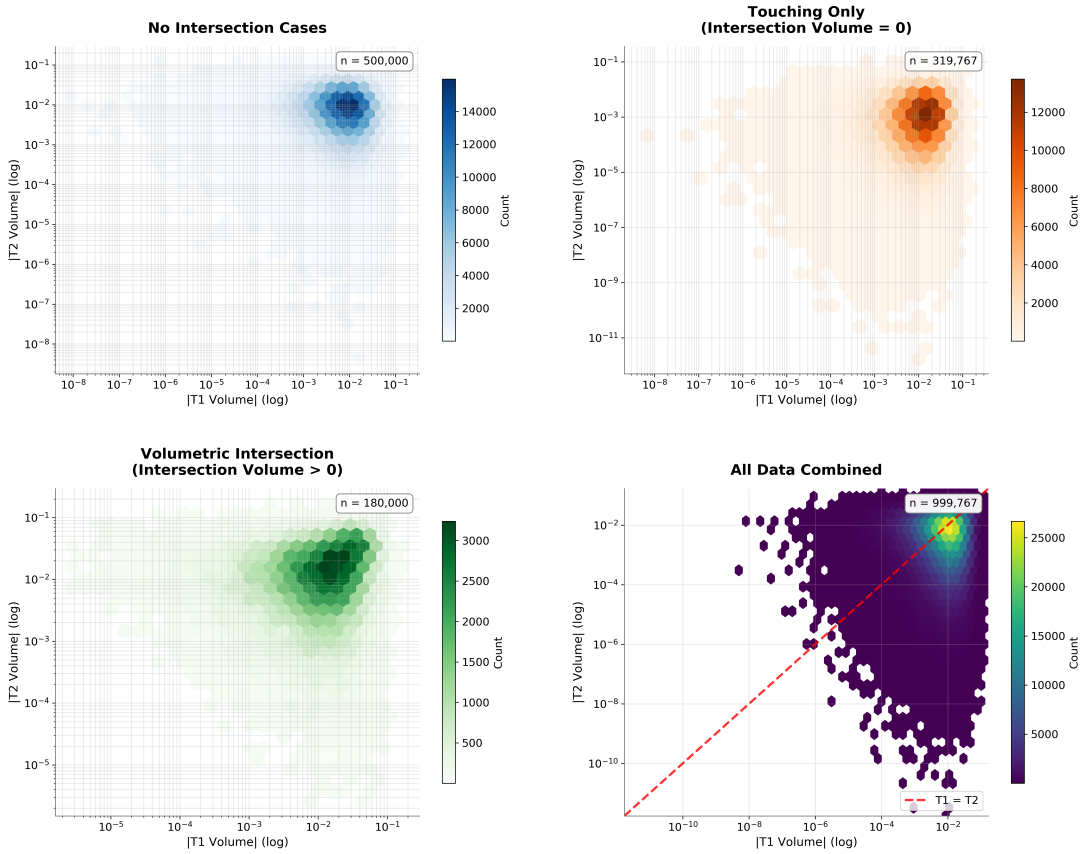

Figure 5.8: Joint distribution of T1 and T2 volumes. Concentration in the upper-right region indicates that most geometric interactions occur when both tetrahedra have moderate to large volumes.

5.12.3 Intersection Analysis and Label Distribution

Intersection Volume Characteristics

The dataset maintains a balanced 50-50 split between intersecting and non-intersecting samples, preventing classification bias while ensuring comprehensive coverage of both positive and negative cases. This distribution captures the full spectrum of geometric relationships, from complete separation through edge contact to full volumetric overlap.

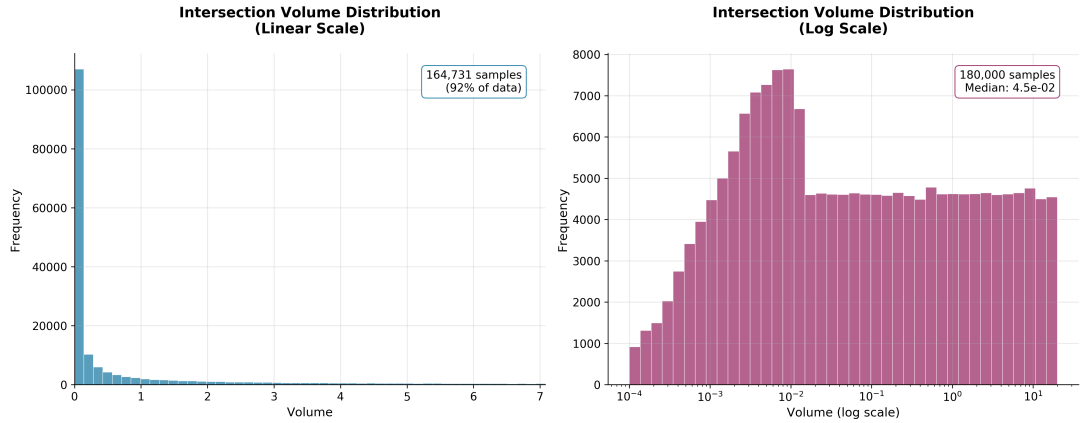

Figure 5.9: Joint volume distribution (V_1, V_2) across 1 million samples, showing balanced representation of intersecting and non-intersecting configurations.

Category	Count (%)	Geometric Significance
Total Samples	1,000,000	Complete dataset
Non-Intersecting	500,000 (50.0%)	Separated tetrahedra
Intersecting (Total)	500,000 (50.0%)	Contact or overlap
Touching Only	320,000 (32.0%)	Surface/edge contact
Overlapping	180,000 (18.0%)	Volumetric intersection

Table 5.33: Intersection distribution providing balanced learning scenarios across geometric relationship types.

Volume Scale Analysis

The intersection volumes span approximately 5.3 orders of magnitude, from 1.00×10^{-4} to 2.00×10^{1} (after 10^{3} scaling), with a median volume of 4.47×10^{-2}. This wide range ensures that models learn to handle both microscopic intersections and substantial overlaps.

Category	Scaled Range	Count (%)	Interaction Type
Small ($< 10^{-1}$)	$< 10^{-4}$	101,907 (57%)	Fine-scale interactions
Medium (10^{-1}–10^{1})	10^{-4}–10^{-2}	72,858 (40%)	Moderate overlaps
Large ($> 10^{1}$)	$> 10^{-2}$	5,235 (3%)	Substantial intersections

Table 5.34: Scaled volume categories emphasizing smaller intersections while maintaining adequate representation of larger overlaps.

5.12.4 Geometric Symmetry and Balance

Volume Relationship Analysis

The relative volumes between tetrahedron pairs significantly impact model generalization. The dataset exhibits a controlled asymmetry with T1 being larger in 62.4% of cases versus T2 in 37.6%, creating a 1.66 ratio imbalance that provides learning diversity without extreme bias.

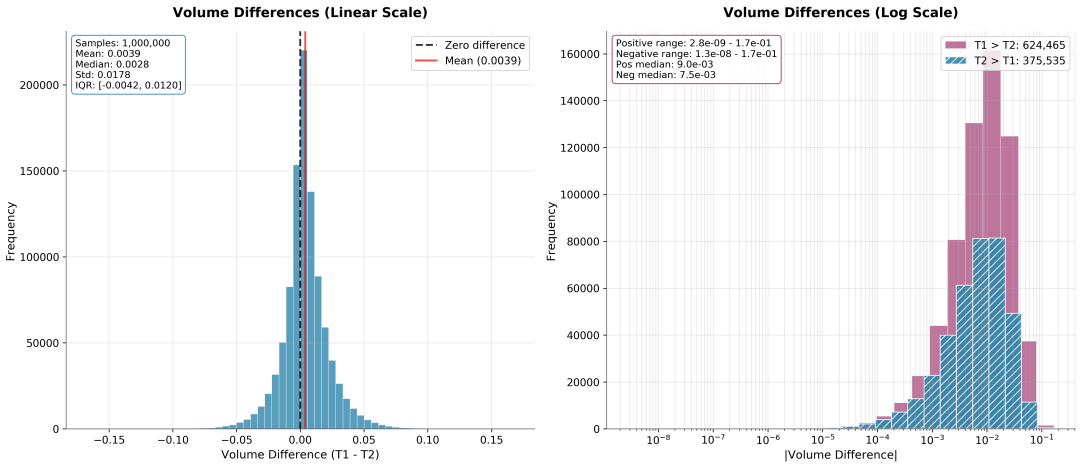

Figure 5.10: Distribution of volume differences between T1 and T2, showing controlled asymmetry that promotes learning diversity while maintaining numerical stability.

Metric	Value	Implication
Range	$[-0.1671,\ 0.1671]$	Bounded variation
Mean	0.003928	Slight T1 bias
Standard Deviation	0.017754	Controlled variation
T1 Larger	624,465 (62.4%)	Moderate asymmetry
T2 Larger	375,535 (37.6%)	Sufficient diversity
Ratio Imbalance	1.66	Manageable bias

Table 5.35: Volume difference statistics demonstrating controlled asymmetry that enhances learning without introducing extreme bias.

Size Relationship Categories

Category	Count (%)	Numerical Impact
Small Differences (< 0.01)	556,894 (55.7%)	High stability
Medium Differences (0.01–0.1)	442,801 (44.3%)	Moderate variation
Large Differences (> 0.1)	305 (0.0%)	Minimal extremes

Table 5.36: Size relationship distribution ensuring numerical stability while providing sufficient geometric diversity.

This distribution ensures that most tetrahedron pairs have similar volumes, promoting numerical stability and algorithmic robustness while still including sufficient asymmetric cases to prevent overfitting to balanced configurations.

Chapter 6

Evaluation and Results Discussion

Until now, model development has primarily focused on component-level tuning and architectural design. This chapter presents a comprehensive evaluation of the final model's behavior, emphasizing its reliability, throughput, and suitability for real-time tetrahedral intersection tasks. The analysis is structured around core performance metrics and practical deployment criteria.

We begin by analyzing classification and regression results, followed by a detailed examination of latency, generalization capacity, and limitations. Finally, we compare the learned model against traditional geometric algorithms to highlight trade-offs between accuracy and computational cost.

6.1 Error Analysis

The model exhibits negligible representational error on the order of 10^{-15}, stemming from inherent limitations of `float64` floating-point precision. Beyond this numerical floor, classification and regression performance depend primarily on the geometry of the tetrahedral pairs and the quality of the learned latent representation.

Below, we analyze errors in each task for our best-trained model, TetrahedronPairNet (L).

6.1.1 Classification Performance

Dataset Subset	Samples	Accuracy
No Intersection	100,000	97.40%
Point Intersection	100,000	98.07%
Segment Intersection	100,000	99.20%
Polygon Intersection	100,000	98.61%
Polyhedron Intersection	100,000	99.63%
Overall Accuracy	500,000	98.58%
Mean Accuracy	—	98.58%
AUC	—	0.9813

Table 6.1: Classification performance of TetrahedronPairNet (L) on the test set, broken down by intersection types and aggregated metrics.

Table 6.1 presents the main results for the classification task. An overall accuracy of 98.58% implies approximately 15 misclassifications per 1,000 predictions—sufficiently low for most 3D geometric processing pipelines.

The mean accuracy of 98.58%, computed uniformly across intersection types, shows that performance is stable across different geometric configurations, not just dominated by majority classes.

An AUC of 0.9813 indicates strong discrimination ability under varying decision thresholds, with a low overlap between positive and negative classes.

In summary, TetrahedronPairNet (L) demonstrates high classification fidelity and generalization across diverse intersection types, making it a reliable component for real-time tetrahedral analysis.

6.1.2 Regression Performance

The regression task involves estimating the *intersection volume* between two tetrahedra—a more complex challenge than binary classification, particularly when non-zero volumetric overlap is present. Table 6.2 shows the detailed results.

Dataset Subset	Samples	MAE	R^2	κ	Mean Bin Accuracy (10 bins)
No Intersection	100,000	6.22×10^{-7}	0	0	0.99992
Point Intersection	100,000	2.34×10^{-8}	0	0	1.00000
Segment Intersection	100,000	3.81×10^{-7}	0	0	0.99993
Polygon Intersection	100,000	2.99×10^{-6}	0	0	0.99962
Polyhedron Intersection	100,000	1.19×10^{-3}	0.68348	0.41463	0.34832

Table 6.2: Regression performance of TetrahedronPairNet (L) on the test set, broken down by intersection types.

In subsets where the ground truth volume is zero (no, point, segment, polygon), the model demonstrates exceptionally low MAE—typically below 10^{-6}—and near-perfect bin accuracy. These cases reflect strong reliability in detecting the absence of volumetric intersection. The R^2 and κ scores are zero, simply due to the lack of variance in the targets.

The **Polyhedron Intersection** subset presents the real regression challenge, requiring accurate prediction of continuous, non-zero volumes. Here, the model achieves:

- **MAE of** 1.19×10^{-3}, meaning average prediction errors are on the order of 0.0012—small in a unit-cube context, but potentially significant in high-precision applications.
- R^2 **of 0.683**, indicating that 68% of the variance in true volumes is explained by the model.
- **Cohen's** κ **of 0.415**, showing moderate agreement between predicted and true volume classes.
- **Mean bin accuracy of 34.8%**, reflecting challenges in assigning predictions to the correct quantized volume class, especially near bin boundaries.

To analyze this behavior more deeply, we partition the polyhedron subset into equal-width volume intervals (Table 6.3).

Volume Interval	MAE	MSE	Samples	Bin Accuracy
[0.0000, 0.0011)	3.42×10^{-4}	3.59×10^{-7}	11,130	86.09%
[0.0011, 0.0022)	6.64×10^{-4}	8.52×10^{-7}	11,124	47.48%
[0.0022, 0.0033)	9.14×10^{-4}	1.46×10^{-6}	10,972	35.07%
[0.0033, 0.0044)	1.11×10^{-3}	2.03×10^{-6}	11,119	30.03%
[0.0044, 0.0055)	1.28×10^{-3}	2.70×10^{-6}	11,135	26.60%
[0.0055, 0.0066)	1.45×10^{-3}	3.36×10^{-6}	11,138	23.69%
[0.0066, 0.0077)	1.56×10^{-3}	3.90×10^{-6}	11,121	22.93%
[0.0077, 0.0088)	1.66×10^{-3}	4.37×10^{-6}	11,128	21.10%
[0.0088, 0.0100]	1.73×10^{-3}	4.70×10^{-6}	11,133	20.50%

Table 6.3: Granular regression performance of TetrahedronPairNet (L) across binned intersection volumes in the polyhedron subset.

From this finer-grained view, three insights emerge:

- **Excellent accuracy in small overlaps:** For volumes below 0.0011, the model achieves high precision (MAE = 3.42×10^{-4}) and 86% bin accuracy, showing clear strength in estimating minor contacts.
- **Performance drops with volume:** As intersection size grows, both MAE and MSE increase, while bin accuracy drops—from 47% in the next bin down to 20% in the topmost one. This illustrates a clear decline in precision with volumetric complexity.
- **Latent resolution bottleneck:** The sharp decline in bin accuracy suggests that the model's internal representation lacks the resolution needed to capture fine-grained geometric cues in high-overlap scenarios.

In sum, TetrahedronPairNet (L) excels at detecting small or zero-volume interactions, but struggles to maintain precision as volume increases. This trade-off highlights the need for improved latent representations or volume-aware learning objectives.

6.1.3 Consistency

To evaluate the robustness of TetrahedronPairNet to input permutations, we assess its *consistency* across two dimensions: classification and regression outputs, under both pointwise and tetrahedronwise permutations. This tests the model's invariance to vertex reordering and tetrahedron swapping. A consistency rate of 100% indicates perfect invariance; lower values reveal sensitivity to input ordering.

For the regression task, exact numerical agreement is neither expected nor required due to floating-point imprecision. Instead, a prediction is considered consistent if it satisfies both a relative error threshold ($\texttt{rtol} = 0.1$) and an absolute error threshold ($\texttt{atol} = 0.001$) compared to the original output. This dual-threshold rule tolerates minor fluctuations for small volumes, while enforcing stricter agreement for larger ones. It balances numerical tolerance with practical reliability in downstream applications. Table 6.4 details the results for the consistency test.

Subset	**Samples**	**Classification (%)**		**Regression (%)**	
		Pointwise	**Tetrawise**	**Pointwise**	**Tetrawise**
No Intersection	100,000	100.00	2.61	100.00	0.68
Point Intersection	100,000	100.00	98.07	100.00	15.38
Segment Intersection	100,000	100.00	99.20	100.00	14.47
Polygon Intersection	100,000	100.00	98.61	100.00	13.31
Polyhedron Intersection	100,000	100.00	99.63	100.00	0.41

Table 6.4: Classification and regression consistency (%) of TetrahedronPairNet (L) under vertex (pointwise) and tetrahedron-level permutations.

Classification Consistency. Pointwise permutation invariance is perfect across all subsets—each classification decision remains unchanged under vertex reordering. Tetrahedronwise classification consistency is also very high, exceeding 98% across all intersection types. This indicates that the model's binary decision function is largely unaffected by reordering of tetrahedral inputs.

Regression Consistency. The regression task is more sensitive to input ordering. While pointwise permutation consistency remains perfect under the defined thresholds, tetrahedronwise consistency varies significantly across subsets. In particular, the *no intersection* and *polyhedron intersection* cases exhibit the lowest consistency scores. In practice, this means that swapping the order of the tetrahedra often changes the predicted intersection volume by more than the 0.001 threshold. This behavior reveals a structural weakness: the model lacks true permutation invariance and relies on the implicit ordering of its inputs, which introduces variability in its volumetric predictions.

These consistency evaluations are performed entirely on synthetic data sampled from the same distribution used during training. Such a controlled environment simplifies many corner cases and may obscure robustness issues that would arise under real-world conditions. In the next section, we investigate the model's behavior in a more challenging, out-of-distribution setting that better reflects practical deployment scenarios.

6.2 Simulation Proxy Evaluation

To evaluate the practical behavior of TetrahedronPairNet under dynamic and near-realistic conditions, we developed a C++-based proxy simulation. This framework approximates runtime behavior in scenarios involving continuous movement and varying geometric configurations.

6.2.1 Setup

The simulation models a dynamic scene where one tetrahedron is in motion:

1. Two initially non-intersecting tetrahedra are generated within a bounded 3D domain.
2. A translation is applied incrementally to one tetrahedron, moving it across the space.
3. At each timestep, the model is queried to classify intersection status and estimate the intersecting volume.
4. The end-to-end latency, including data preprocessing and model inference, is recorded.

To run this evaluation, we load a pre-generated dataset and convert all inputs to tensor format. The scripted model is then loaded and evaluated. Input normalization is applied to fit all coordinates inside a unit cube, with additional

preprocessing such as Principal Axis Transformation (PAT) applied conditionally, depending on the training regime. Although these preprocessing steps introduce latency, they are crucial for achieving high-quality predictions. For convenience and performance, two model wrappers were implemented in both C++ and Python, abstracting away all low-level details from the end user.

6.2.2 Results

The evaluation on 10,000 samples achieved 98.01% accuracy with strong volume regression performance and practical runtime characteristics. Table 6.5 and Table 6.6 summarize the key findings.

Metric	**Overall**	**No Intersection**	**Intersection**	**Unit**
Samples	10,000	6,665	3,335	–
Accuracy	0.9801	–	–	–
Precision	0.9612	–	–	–
Recall	0.9799	–	–	–
F1-Score	0.9705	–	–	–
MAE	5.621×10^{-5}	0.000	1.685×10^{-4}	volume units
RMSE	1.212×10^{-4}	0.000	2.098×10^{-4}	volume units

Table 6.5: Classification and volume regression performance

Runtime Performance		**Error Analysis**		
Component	**Value**	**Range**	**Length**	**Percentage**
Preprocessing	19.13 ms	[3729–3795]	67	0.67%
Inference	249.20 ms	[7064–7195]	132	1.32%
Total	**268.32 ms**	**Total Errors**	**199**	**1.99%**
Per sample	0.027 ms	*Errors occur in contiguous ranges*		
Throughput	37,269 samples/s	*near tetrahedron contact zones*		

Table 6.6: Runtime performance and error distribution

The results demonstrate strong performance across all metrics. High recall (0.9799) ensures effective intersection detection, while low regression errors (RMSE $< 2.1 \times 10^{-4}$) confirm stable volume prediction. With a throughput of 37K samples per second, the model supports real-time applications. Mispredictions occur in contiguous ranges near-contact scenarios, indicating systematic failure modes tied to geometric ambiguity—highlighting targets for future refinement in close-proximity edge cases.

A natural question arises: how can we be confident the model will generalize beyond the training data's infinite possible tetrahedron configurations? Inspired by similarity principles in geometry, we propose that the model implicitly learns a form of "tetrahedral similarity." This means that when arbitrary tetrahedra are rescaled to the training distribution's scale, accurate predictions remain feasible—even if original sizes vary widely. Though this remains to be empirically confirmed, it frames the model's robustness conceptually.

From a theoretical standpoint, statistical learning theory predicts that strong results on a well-curated dataset will generalize to new samples drawn from the same distribution [41]. Our diverse training set, encompassing many intersection scenarios and configurations, reinforces this expectation.

Together, these observations suggest the model goes beyond memorization, learning structural representations that reliably extrapolate to unseen but statistically similar tetrahedron pairs, within the scope of the generative assumptions made during training.

Building on this, the next section compares our learned approach to traditional geometric methods for tetrahedron intersection and volume estimation. Classical methods provide exact results but suffer from computational complexity and limited scalability, while our model offers a pragmatic trade-off—slightly reduced precision in exchange for orders-of-magnitude faster inference and real-time throughput.

6.3 Comparison with Traditional Methods

Table 6.7 highlights the key differences between our learned model and classical geometric approaches for tetrahedron–tetrahedron intersection detection and volume estimation.

Metric	**Traditional Geometric Methods**	**Our Learned Model**
Classification Accuracy	100% (Exact)	∼98% (Empirical)
Regression Accuracy (Volume Estimation)	Exact (machine precision limited)	Approximate, average absolute error 0.001190 on intersection cases (explains 68% variance)
Inference Speed	Slow (complex geometric predicates)	Fast (≈30,000 samples/s, batch size 2048, single-thread CPU)
Batching Support	Limited or none	Fully supported
GPU Native	Rare or unsupported	Yes, GPU-deployable
Memory Footprint	Variable	Compact (<500 KB)
Robustness	Exact but computationally expensive; approximate methods brittle due to floating-point errors	Reliable in typical cases; challenges in close proximity; limited interpretability

Table 6.7: Comparison of Proposed Model vs. Traditional Methods

Traditional methods offer exact guarantees but at significant computational cost and limited scalability. In contrast, our model offers an efficient approximation: high classification and regression accuracy with orders-of-magnitude faster inference. This makes it especially suited for real-time, high-throughput scenarios where small trade-offs in precision are acceptable.

6.4 Limitations

The evaluation reveals several critical limitations to consider for deployment:

Performance Degradation with Volume Complexity: The model's accuracy declines as intersection volume increases—achieving 86% bin accuracy for small overlaps (<0.0011 volume units) but dropping to 20% for larger intersections. This limits utility in applications demanding high precision across all overlap scales.

Tetrahedronwise Permutation Sensitivity: Despite perfect pointwise permutation invariance, regression consistency under tetrahedronwise permutations falls as low as 0.41% for intersecting and 0.68% for non-intersecting cases, causing inconsistent predictions when tetrahedra are swapped.

Close-Proximity Failure Modes: Systematic failures cluster near contact zones, affecting 1.99% of predictions where geometric ambiguity peaks—critical for robust boundary detection.

Black-box Interpretability: As a learned, non-symbolic model, it lacks explicit geometric reasoning or intermediate representations, complicating debugging and integration with symbolic systems.

Training Distribution Dependence: Validated only on synthetic data matching training distribution; generalization to real-world meshes with differing size, aspect ratios, or pathologies remains unverified. The "tetrahedral similarity" hypothesis requires empirical support.

Preprocessing Coupling: Accuracy tightly depends on consistent normalization and canonicalization; deviations between training and deployment degrade performance, creating a fragile pipeline.

Volume Resolution Floor: The minimum reliably predicted volume (10^{-7}) may be insufficient for extremely small intersections or high-precision tasks.

Statistical Nature: Unlike exact geometric methods, the model's probabilistic outputs (1.42% classification error, volume prediction uncertainty) complicate integration where guaranteed correctness is required.

Robust validation, comprehensive monitoring, and careful failure mode consideration are essential. **Performance guarantees must not be assumed outside the validated training distribution without extensive empirical verification.**

6.5 Deployment Scenarios

TetrahedronPairNet delivers exceptional computational performance: over 37,000 tetrahedron pairs processed per second on a single CPU thread, with 10,000 samples completed in under 270 milliseconds. This represents a paradigm shift in geometric processing efficiency.

The model's compact architecture (under 500 KB) supports deployment across a wide range of hardware platforms—from edge devices like Raspberry Pi to high-performance AI accelerators such as NVIDIA Jetson—democratizing advanced geometric computation in resource-constrained environments. Its batch-optimized design further unlocks substantial gains on GPUs, making it well-suited for large-scale simulations and real-time graphics applications.

Performance by Application Domain

Ultra-High Throughput Applications benefit most from the model's real-time speed. Use cases include physics engines, VR/AR collision detection, and interactive simulations, where sub-millisecond inference and 98.01% classification accuracy enable responsive experiences at scales that were previously impractical.

Importantly, the model's systematic failure modes—particularly in close-contact zones—are well understood. These limitations can be mitigated through robust safeguards such as fallback to exact methods and confidence-based prediction filtering, ensuring system reliability even in ambiguous configurations.

Medium-Throughput Workflows such as adaptive mesh refinement, CAD model validation, or dynamic geometry optimization, capitalize on the model's regression performance: explaining 68% of variance in intersection volumes with a mean absolute error of 1.19×10^{-3}. This accuracy–speed trade-off is suitable for most engineering-grade applications where perfect precision is not mandatory.

High-Precision Domains demanding strict geometric correctness—e.g., safety-critical systems or precision manufacturing pipelines—can use TetrahedronPairNet as a high-speed screening stage. Final verification via exact geometric algorithms ensures deterministic guarantees.

Deployment Architecture Considerations

Preprocessing Pipeline Integrity is vital. The model's accuracy depends on strict consistency with training-time preprocessing, including normalization, coordinate transformations, and feature extraction. Deviations can severely degrade prediction quality. Deployment must therefore incorporate rigorous pipeline validation, unit tests, and preprocessing version control.

Robust Error Handling elevates TetrahedronPairNet from a fast approximation tool to a trustworthy subsystem. Recommended strategies include:

- **Confidence scoring:** Quantify uncertainty and gate downstream decisions.
- **Ensemble techniques:** Improve prediction stability by averaging multiple model outputs.
- **Fallback systems:** Dynamically switch to exact methods in ambiguous or high-risk regions.
- **Real-time monitoring:** Detect distribution shifts and performance drift during operation.

Validation Requirements are essential due to the synthetic nature of the training data. Comprehensive testing on target domain meshes—across different scales, topologies, and geometric pathologies—is mandatory. Downstream systems must be architected to handle the model's probabilistic nature and to trigger alerts or fallbacks upon confidence loss or distribution mismatch.

In summary, TetrahedronPairNet offers a compelling blend of speed, efficiency, and versatility for geometric processing pipelines. When paired with rigorous safeguards, hybrid validation schemes, and architectural discipline, it becomes a practical tool for accelerating real-world computational geometry.

Chapter 7

Conclusion

This thesis presents a data-driven framework that redefines how geometric simulations are approached—replacing symbolic computation with learned inference to achieve high-speed, approximate solutions within practical error bounds. This shift challenges conventional priorities in computational geometry, particularly in contexts where exactitude is either unnecessary or computationally restrictive.

At the core of this contribution is the TetrahedronPairDatasetV1, a purpose-built dataset comprising one million labeled tetrahedron pairs designed to fuel research in volumetric intersection learning. Building on this foundation, we introduce TetrahedronPairNet, the first neural architecture capable of concurrently predicting binary intersection and estimating intersection volume for tetrahedra at scale. In its raw form—with minimal preprocessing—the model processes over 100,000 samples per second at batch size 2048, achieving 85% classification accuracy. After transformation and tuning, it reaches 30,000 samples per second with 98% classification accuracy and strong regression performance (R^2 = 0.68, MAE $\approx$ 0.00119). These results demonstrate a compelling alternative to traditional algorithmic pipelines, emphasizing scalability, robustness, and real-time capability.

7.1 Impact

TetrahedronPairNet achieves real-time, large-scale narrow-phase intersection testing, long considered infeasible. Traditional methods (subsection 3.1.3) depend on symbolic formulations that suffer from: poor scalability due to sequential, low-level operations, limited parallelism on modern hardware, instability near degenerate configurations (e.g., coplanar faces or tiny volumes).

7.1.1 Why This Was Previously Infeasible

Two primary limitations hindered progress:

- **Lack of data**: No large-scale, labeled datasets of tetrahedral interactions were available; generating reliable volume ground truth is computationally expensive.
- **Architectural mismatch**: Traditional deep learning models (e.g., CNNs) are unsuitable for unordered, irregular geometric inputs. Only recent architectures like PointNet, DeepSets, and attention-based networks enable permutation-invariant learning over 3D structures.

TetrahedronPairNet builds on these advances, using shared vertex encoders, hierarchical pooling, and joint classification-regression objectives to bypass symbolic logic entirely. Its homogeneous, data-parallel design supports massive-scale inference without manual geometry code or predicates.

7.2 Architectural Contributions

Each component serves a specific purpose in enabling accurate, fast, and generalizable inference across a wide variety of tetrahedral intersection scenarios:

- **Permutation-Invariance:** Vertex encoders operate on unordered sets, removing the need for sorting or canonicalization.
- **Hierarchical Design:** Tetrahedron-level encoders capture shape; pairwise fusion layers reason jointly about interactions.
- **Dual-Task Output:** A shared latent space supports simultaneous classification and regression.
- **Batch Scalability:** The architecture is GPU-friendly, processing tens of thousands of examples per frame.
- **Learned Reasoning:** The network infers geometric interactions implicitly, robust even near degenerate cases.

7.3 Application Potential

TetrahedronPairNet enables a broad class of real-time applications:

- **Interactive Physics Simulations**: Cloth simulations can now process fiber-level interactions during manipulation, with collision detection running at frame rates that support responsive user interaction. Deformable bodies respond to touch with millions of collision checks per frame, turning smartphones into physics laboratories and enabling surgical training simulators that feel genuinely real.
- **Live CAD/CAM Systems**: Engineers can validate part geometry and manufacturing constraints while sketching, with interference detection and clearance analysis running continuously. Complex assemblies show interference and clearances in real-time, transforming design from iterative guesswork into fluid creative expression. Digital twins mirror physical manufacturing processes with zero computational delay.
- **Immersive AR/VR**: Virtual objects maintain consistent physical properties through haptic feedback systems. Architectural walkthroughs provide tactile surface information where you feel every surface, medical training where virtual organs respond exactly like real tissue, and gaming worlds that react to your every movement with physical authenticity.
- **Scientific Computing**: Simulations that previously required distributed computing resources can run on individual workstations. Climate models update in real-time as you adjust parameters, protein folding simulations respond instantly to molecular tweaks, and engineering stress analyses guide design decisions as fast as you can think.
- **Autonomous Systems**: Spatial reasoning for navigation happens at the speeds required for real-time decision making. Self-driving cars understand 3D space as intuitively as humans, robots navigate crowded environments with fluid grace, and drones dance between obstacles at high speed—all powered by geometric reasoning that happens faster than human perception.

The model's joint output enables hybrid systems combining binary decisions with spatial metrics. All operations are batched and differentiable—opening the door to end-to-end geometric learning pipelines.

7.4 Future Work

While the core architecture demonstrates strong performance across classification and regression tasks, several directions remain open to enhance its effectiveness and extend its scope:

- **Improving Regression Performance:** Classification reaches near-perfect accuracy, but volume regression remains less precise, especially near ambiguous or degenerate configurations. Future efforts could explore quantile regression, uncertainty-aware loss functions, or hybrid models that combine symbolic priors with learned components.
- **Predicting Intersection Geometry:** Reducing the intersection to a scalar volume discards rich spatial information. An exciting extension would involve predicting the actual shape of the intersecting region—via voxel grids, meshes, or implicit functions—enabling applications in simulation, reconstruction, and physical reasoning.
- **Inverse Problems:** A natural next challenge is to reverse the setup: given a target intersection volume or shape, infer plausible tetrahedron pairs that could have generated it. This inverse formulation is harder but deeply related, with potential applications in generative modeling and geometric synthesis.
- **Architectural Enhancements:** The current architecture is performant but can be extended. Integrating transformation-equivariant modules, attention-based reasoning, or learned canonicalization strategies (e.g., T-Net, GAM) could further improve generalization across more diverse spatial configurations.

7.5 Concluding Remarks

TetrahedronPairNet represents a fundamental shift from procedural geometry to learned inference. The convergence of neural architectures, scalable data generation, and efficient batching unlocks real-time geometric reasoning at scale.

This work offers more than a performance boost, it reframes how we pose and solve geometric problems. Learned models, once peripheral to core simulation and CAD workflows, are now capable of replacing brittle, hand-crafted algorithms with robust, flexible alternatives. Where precision once demanded symbolic rigor, we now harness pattern recognition. Where geometry was once coded, it can now be learned.

In doing so, this thesis lays the foundation for a new era in computational geometry, one where inference replaces logic, data replaces rules, and interactive geometric intelligence becomes a practical reality.

This is not just a model. It is a declaration: that geometry can learn, that interaction can be immediate, and that intelligence can be built not from logic alone, but from data, scale, and abstraction.

As the boundaries between algorithm and inference dissolve, we begin to glimpse the future of simulation, not as a sequence of deterministic steps, but as a fluid, responsive system grounded in learned geometric intuition. The tools are ready. The ideas are here. What happens next is limited only by how far we dare to push the frontier.

Bibliography

[1] P Tiwari and A Shinde. "History and applications of Geometry in real life". In: *JBNB* 12 (2022), pp. 57–67. issn: 2454-2776.

[2] Punam K. Saha, Robin Strand, and Gunilla Borgefors. "Digital Topology and Geometry in Medical Imaging: A Survey". In: *IEEE Transactions on Medical Imaging* 34.9 (2015), pp. 1940–1964. doi: 10.1109/TMI.2015.2417112.

[3] Vitor J. Katz. *A History of Mathematics*. 3rd ed. Pearson Education, Inc, 2009.

[4] Mark De Berg et al. *Computational Geometry. Algorithms and Applications*. Springer Science & Business Media, Apr. 2008.

[5] Özer Çelik. "A Research on Machine Learning Methods and Its Applications". In: *Journal of Educational Technology and Online Learning* (Sept. 2018). doi: 10.31681/jetol.457046.

[6] Khadija El Bouchefry and Rafael S. de Souza. "Chapter 12 - Learning in Big Data: Introduction to Machine Learning". In: *Knowledge Discovery in Big Data from Astronomy and Earth Observation*. Ed. by Petr Škoda and Fathalrahman Adam. Elsevier, 2020, pp. 225–249. isbn: 978-0-12-819154-5. doi: https://doi.org/10.1016/B978-0-12-819154-5.00023-0. url: https://www.sciencedirect.com/science/article/pii/B9780128191545000230.

[7] Weinan E Weinan E. "Machine Learning and Computational Mathematics". In: *Communications in Computational Physics* 28.5 (Jan. 2020), pp. 1639–1670. issn: 1815-2406. doi: 10.4208/cicp.oa-2020-0185. url: http://dx.doi.org/10.4208/cicp.OA-2020-0185.

[8] Umutcan Önal and Zafeirakis Zafeirakopoulos. "A Machine Learning Framework for Volume Prediction". In: *Analysis of Experimental Algorithms*. Ed. by Ilias Kotsireas et al. Cham: Springer International Publishing, 2019, pp. 408–423.

[9] Harilaos Psaraftis, Min Wen, and Christos Kontovas. "Dynamic Vehicle Routing Problems: Three Decades and Counting". In: *Networks* 67 (Aug. 2015), pp. 3–31. doi: 10.1002/net.21628.

[10] W. J. B. van Eeghen, O. W. van Gaans, and M. van der Schans. "Analysis of Near-Optimal Portfolio Regions and Polytope Theory". Master's Thesis. Leiden University, 2018.

[11] José Salgado-Rojas et al. "A mixed integer programming approach for multi-action planning for threat management". In: *Ecological modelling* 418 (2020), p. 108901.

[12] Lucas Pahl. "Polytope-form games and index/degree theories for extensive-form games". In: *Games and Economic Behavior* 141 (2023), pp. 444–471. issn: 0899-8256. doi: https://doi.org/10.1016/j.geb.2023.07.001. url: https://www.sciencedirect.com/science/article/pii/S0899825623000945.

[13] Ambarish Kulkarni et al. "Virtual prototyping used as validation tool in automotive design". In: *Built Environ.* (Jan. 2011).

[14] Navid Mirzayousef Jadid. "Tetrahedral Kdet: Linear Time Collision Detection for Tetrahedral Meshes". Master's thesis. University of Bremen, 2023.

[15] Huang Zhang et al. "Deep learning-based 3D point cloud classification: A systematic survey and outlook". In: *Displays* 79 (2023), p. 102456. issn: 0141-9382. doi: https://doi.org/10.1016/j.displa.2023.102456. url: https://www.sciencedirect.com/science/article/pii/S0141938223000896.

[16] Hang Si and Ideen Sadrehaghighi. *Tetgen -A Delaunay-Based Quality Tetrahedral Mesh Generator*. July 2022. doi: 10.13140/RG.2.2.13915.85284/2.

[17] René Weller, Nicole Debowski, and Gabriel Zachmann. "kDet: Parallel Constant Time Collision Detection for Polygonal Objects". In: *Computer Graphics Forum* 36.2 (2017), pp. 131–141. doi: https://doi.org/10.1111/cgf.13113. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13113. url: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13113.

[18] Newcastle University. *Physics - Collision Detection*. Newcastle University. url: https://research.ncl.ac.uk/game/mastersdegree/gametechnologies/physicstutorials/4collisiondetection/Physics%20-%20Collision%20Detection.pdf (visited on 11/30/2024).

[19] Yang-Hui He, Elli Heyes, and Edward Hirst. *Machine Learning in Physics and Geometry*. 2023. arXiv: 2303.12626 [hep-th].

[20] Vera Viana. *Tetrahedra inside the cube*. Accessed on 16/12/2023. 2020. url: https://veraviana.net/enclosing/inside-the-cube/.

[21] Paul Ernest. "Mathematics, ethics and purism: an application of MacIntyre's virtue theory". In: *Synthese* 199.2 (2021), pp. 3137–3167. doi: 10.1007/s11229-020-02928-1.

[22] H. J. M. Bos and H. Mehrtens. "The interactions of mathematics and society in history: Some exploratory remarks". In: *Historia Mathematica* 4.1 (1977), pp. 7–30. doi: 10.1016/0315-0860(77)90025-7.

[23] R. Jayanthi. "Mathematics in Society Development - A Study". In: *IRE Journals* 3.3 (2019), pp. 59–64. issn: 2456-8880.

[24] Sarah Giest and Annemarie Samuels. "'For good measure': data gaps in a big data world". In: *Policy Sciences* 53 (2020), pp. 559–569. doi: 10.1007/s11077-020-09384-1.

[25] Yasemin J. Erden. "IDENTITY AND BIAS IN PHILOSOPHY: WHAT PHILOSOPHERS CAN LEARN FROM STEM SUBJECTS". In: *Think* 20.59 (2021), pp. 117–131. doi: 10.1017/S1477175621000245.

[26] Melissa Heikkilä. "AI: Decoded: Spain's flawed domestic abuse algorithm — Ban debate heats up — Holding the police accountable". In: *POLITICO* (2023). AI: Decoded newsletter. url: https://www.politico.eu/newsletter/ai-decoded/spains-flawed-domestic-abuse-algorithm-ban-debate-heats-up-holding-the-police-accountable-2/.

[27] Anders Nordgren. "Artificial intelligence and climate change: ethical issues". In: *Journal of Information, Communication and Ethics in Society* 21.1 (2023), pp. 1–15. doi: 10.1108/JICES-11-2021-0106.

[28] S. Kockara et al. "Collision detection: A survey". In: *2007 IEEE International Conference on Systems, Man and Cybernetics*. 2007, pp. 4046–4051. doi: 10.1109/ICSMC.2007.4414258.

[29] John Burkardt. *Computational Geometry Lab: TETRAHEDRONS*. http://people.sc.fsu.edu/~jburkardt/presentations/cg_lab_tetrahedrons.pdf. Accessed: December 2, 2024. Aug. 2018.

[30] Online Etymology Dictionary. *Tetrahedron - Etymology, Origin & Meaning*. Accessed June 18, 2025. 2025. url: https://www.etymonline.com/word/tetrahedron.

[31] Wikipedia contributors. *Simplex*. Accessed June 18, 2025. 2025. url: https://en.wikipedia.org/wiki/Simplex.

[32] Wei Li and Cindy X. Chen. "Efficient data modeling and querying system for multi-dimensional spatial data". In: *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. GIS '08. Irvine, California: Association for Computing Machinery, 2008. isbn: 9781605583235. doi: 10.1145/1463434.1463503. url: https://doi.org/10.1145/1463434.1463503.

[33] Conor McCoid and Martin J. Gander. "Intersection of Tetrahedra". Unpublished manuscript. 2020. url: https://www.unige.ch/~mccoid/ongoing/mccoid2020tetrahedra.pdf.

[34] Csaba D. Toth. "Geometric Intersection". In: *Handbook of Discrete and Computational Geometry*. Ed. by Jacob E. Goodman and Joseph O'Rourke. Chapman and Hall/CRC, 2004. Chap. 42. url: https://www.csun.edu/~ctoth/Handbook/chap42.pdf.

[35] Gino van den Bergen. "A Fast and Robust GJK Implementation for Collision Detection of Convex Objects". In: *Journal of Graphics Tools* 4.2 (1999), pp. 7–25. doi: 10.1080/10867651.1999.10487502.

[36] Fabio Ganovelli, Federico Ponchio, and Claudio Rocchini. "Fast Tetrahedron-Tetrahedron Overlap Algorithm". In: *Journal of Graphics Tools* 7.2 (2002), pp. 17–25. doi: 10.1080/10867651.2002.10487557.

[37] Philip Schneider and David H. Eberly. *Geometric Tools for Computer Graphics*. San Francisco: Morgan Kaufmann, 2002. isbn: 1558605940.

[38] Nicolai Baldin. "Estimating the volume of a convex body". In: *Snapshots of modern mathematics from Oberwolfach* 15 (2018).

[39] Peter Hachenberger and Lutz Kettner. *3D Boolean Operations on Nef Polyhedra*. 5.6. CGAL User and Reference Manual. CGAL Editorial Board. 2023. url: https://doc.cgal.org/5.6/Manual/packages.html#PkgNef3.

[40] Dirk P. Kroese and Reuven Y. Rubinstein. "Monte Carlo methods". In: *Wiley Interdisciplinary Reviews: Computational Statistics* 4.1 (2012), pp. 48–58. doi: 10.1002/wics.194.

[41] Michael M. Bronstein et al. "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges". In: *CoRR* abs/2104.13478 (2021). arXiv: 2104.13478. url: https://arxiv.org/abs/2104.13478.

[42] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge, UK: Cambridge University Press, 2014. url: https://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/.

[43] Simon J.D. Prince. *Understanding Deep Learning*. The MIT Press, 2023. url: http://udlbook.com.

[44] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[45] R. Magdalena-Benedicto, S. Pérez-Díaz, and A. Costa-Roig. "Challenges and Opportunities in Machine Learning for Geometry". In: *Mathematics* 11.11 (2023), p. 2576. doi: 10.3390/math11112576. url: https://doi.org/10.3390/math11112576.

[46] Vladimir Vapnik and Rauf Izmailov. "Rethinking statistical learning theory: learning using statistical invariants". In: *Machine Learning* 108 (2019), pp. 381–423. doi: 10.1007/s10994-018-5742-0.

[47] Geordie Williamson. "Is deep learning a useful tool for the pure mathematician?" In: *arXiv preprint arXiv:2304.12602* (2023).

[48] Saifullahi Aminu Bello et al. "Review: Deep Learning on 3D Point Clouds". In: *Remote Sensing* 12.11 (2020). issn: 2072-4292. doi: 10.3390/rs12111729. url: https://www.mdpi.com/2072-4292/12/11/1729.

[49] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. Expanded Edition, Reissue with new foreword by Léon Bottou. Originally published 1969; expanded edition 1988; reissued 2017 with new foreword. Cambridge, MA: MIT Press, 2017. isbn: 978-0-262-53477-2. url: https://leon.bottou.org/publications/pdf/perceptrons-2017.pdf.

[50] Aaron Zweig. "Symmetric Functions and Neural Networks". A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy. PhD thesis. Department of Computer Science, New York University, Jan. 2024. url: https://cs.nyu.edu/media/publications/Aaron_Zweig_Thesis.pdf.

[51] Yang Liu, Ernest Fokoue, and Daniel Krutz. *Towards Predicate-powered Learning*. 2024. url: https://openreview.net/forum?id=V2LpzVNtCT.

[52] Charles Ruizhongtai Qi et al. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *CoRR* abs/1706.02413 (2017). arXiv: 1706.02413. url: http://arxiv.org/abs/1706.02413.

[53] Manzil Zaheer et al. *Deep Sets*. 2018. arXiv: 1703.06114 [cs.LG]. url: https://arxiv.org/abs/1703.06114.

[54] A. Zinani. "The Expected Volume of a Tetrahedron whose Vertices are Chosen at Random in the Interior of a Cube". In: *Monatshefte für Mathematik* 139.4 (Aug. 2003), pp. 341–348. doi: 10.1007/s00605-002-0531-y.

[55] Charles R. Qi et al. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. 2017. arXiv: 1612.00593 [cs.CV]. url: https://arxiv.org/abs/1612.00593.

[56] Xu Ma et al. *Rethinking Network Design and Local Geometry in Point Cloud: A Simple Residual MLP Framework*. 2022. arXiv: 2202.07123 [cs.CV]. url: https://arxiv.org/abs/2202.07123.

[57] Guocheng Qian et al. *PointNeXt: Revisiting PointNet++ with Improved Training and Scaling Strategies*. 2022. arXiv: 2206.04670 [cs.CV]. url: https://arxiv.org/abs/2206.04670.

[58] Jiakang Bao et al. "Polytopes and machine learning". In: *arXiv preprint arXiv:2109.09602* (2021).

[59] Richard Sutton. *The Bitter Lesson*. Accessed: 2024-12-04. 2019. url: http://www.incompleteideas.net/IncIdeas/BitterLesson.html.

Appendix A

Model Capacity Full Experiments

# Parameters	# Hidden Layers	Heuristic	# Hidden Units	AUC
417	1	none	[16]	0.86
833	1	none	[32]	0.87
1665	1	none	[64]	0.89
3329	1	none	[128]	0.89
6657	1	none	[256]	0.90
13313	1	none	[512]	0.91
26625	1	none	[1024]	0.92
53249	1	none	[2048]	0.92
689	2	constant	[16, 16]	0.87
1889	2	constant	[32, 32]	0.91
5825	2	constant	[64, 64]	0.94
19841	2	constant	[128, 128]	0.96
72449	2	constant	[256, 256]	0.97
275969	2	constant	[512, 512]	0.97
961	3	constant	[16, 16, 16]	0.88
2945	3	constant	[32, 32, 32]	0.92
9985	3	constant	[64, 64, 64]	0.95
36353	3	constant	[128, 128, 128]	0.97
361	2	prog. doubling	[8, 16]	0.85
977	2	prog. doubling	[16, 32]	0.87
2977	2	prog. doubling	[32, 64]	0.91
10049	2	prog. doubling	[64, 128]	0.94
36481	2	prog. doubling	[128, 256]	0.96
138497	2	prog. doubling	[256, 512]	0.97
921	3	prog. doubling	[8, 16, 32]	0.86
3121	3	prog. doubling	[16, 32, 64]	0.89
11361	3	prog. doubling	[32, 64, 128]	0.95
43201	3	prog. doubling	[64, 128, 256]	0.96
545	2	prog. halving	[16, 8]	0.87
1345	2	prog. halving	[32, 16]	0.90
3713	2	prog. halving	[64, 32]	0.94
11521	2	prog. halving	[128, 64]	0.96
39425	2	prog. halving	[256, 128]	0.96
144385	2	prog. halving	[512, 256]	0.97
1473	3	prog. halving	[32, 16, 8]	0.90
4225	3	prog. halving	[64, 32, 16]	0.94
13569	3	prog. halving	[128, 64, 32]	0.96
47617	3	prog. halving	[256, 128, 64]	0.97
697	3	mirror CE	[16, 8, 16]	0.87
1905	3	mirror CE	[32, 16, 32]	0.91
5857	3	mirror CE	[64, 32, 64]	0.94
19905	3	mirror CE	[128, 64, 128]	0.96
72577	3	mirror CE	[256, 128, 256]	0.97

Table A.1: Comparison of model configurations with corresponding AUC scores

Appendix A

Dataset Generator System

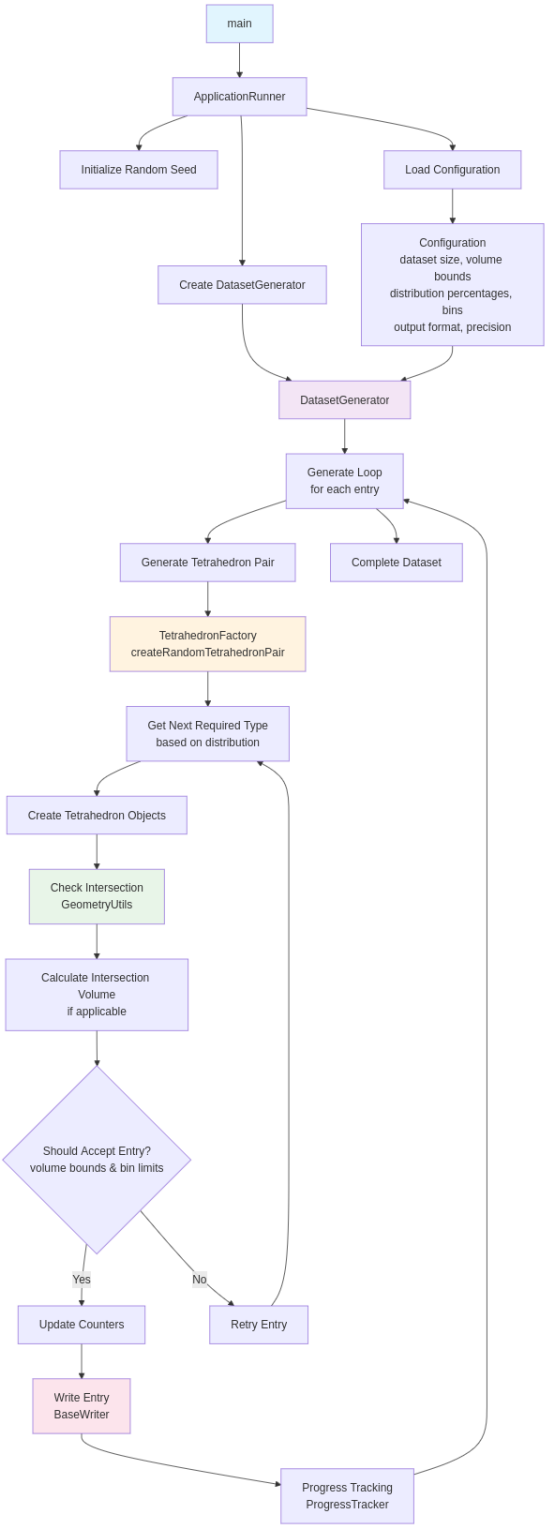

Figure A.1: Full architectural diagram of the dataset generator system. Each module is functionally isolated and interacts through explicit data interfaces.