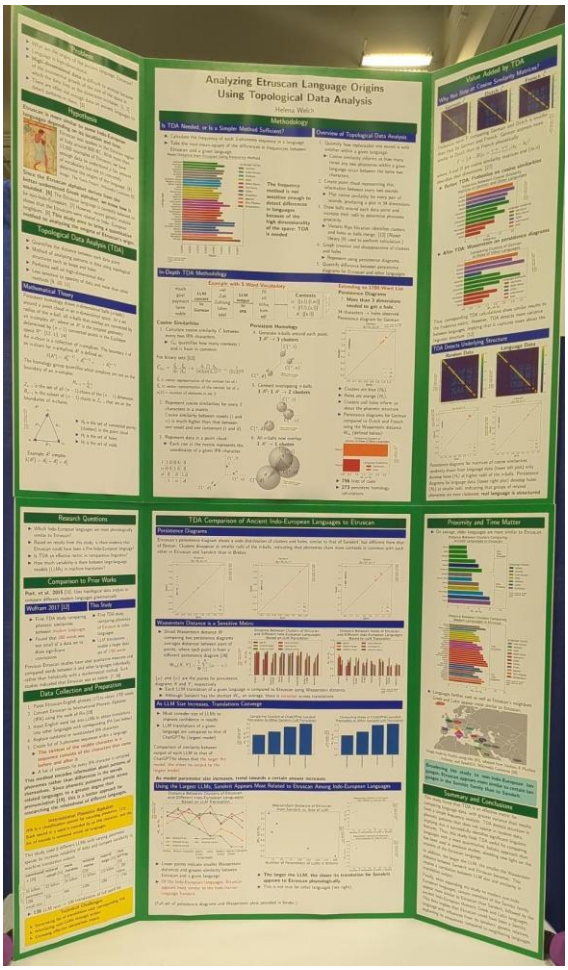
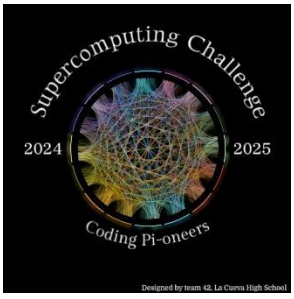
Finalist Reports

2024-2025



https://supercomputingchallenge.org



Printed by PNM
Compiled by the
Supercomputing Challenge

Cover: **Analyzing Pre-Indo-European Theory of Etruscan Language Origins Using Topological Data Analysis**

Welch Home School
Team Member: **Helena Welch**
Sponsor: **Cindy Welch**

Winner in the Technical Poster Competition

Notification: These final reports are presented in an un-abridged form, though printing might be done in black and white. Complete copies of the final reports, including color graphics, along with the code, are available from the archives of Supercomputing Challenge web site: https://supercomputingchallenge.org .

New Mexico Supercomputing Challenge
2024 – 2025 Finalist Reports

Table of Contents

About the New Mexico Supercomputing Challenge
For more information, please visit our website at https://supercomputingchallenge.org ……. **2**

2024—2025 Supercomputing Challenge Awards ……..…….. 4

Sponsors ……..…… 5

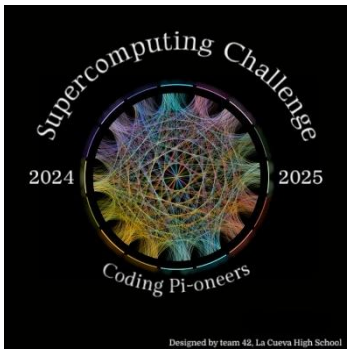
Scholarship winners ……..…… 6

Participants ……..... 8

Judges ……..…….…….... 10

Finalist Reports …… 13

1. Los Alamos High School, *Constraining the Neutron Star Equation of State with Observational Data*
2. Los Alamos High School, *Point Cloud Surface Reconstruction*
3. La Cueva High, *You Only Look Once Machine Learning Solution to Orbital Debris Detection*
4. Albuquerque Academy, *Predicting the drug and micro-/nano-plastic interactions inside the body using molecular dynamics modeling and machine learning*
5. Santa Fe Prep, *Understanding and Predicting Trail Maintenance Needs Using Machine Learning*
6. Santa Fe Prep, *Investigating Intersubjective Realities From Novel NLP and Chaos Theory Approach*
7. Welch Homeschool, *Analyzing Pre-Indo-European Theory of Etruscan Language Origins Using Topological Data Analysis*
8. Welch Homeschool, *Modeling Fast Moving Objects in Crowded Astronomical Neighborhoods*



Supercomputing Challenge Vision

The Vision of the Supercomputing Challenge is to be a nationally recognized program that promotes computational thinking in science and engineering so that the next generation of high school graduates is better prepared to compete in an information-based economy.

Supercomputing Challenge Mission

The Mission of the Supercomputing Challenge is to teach teams of middle and high school students how to use powerful computers to analyze, model and solve real world problems.

About the Supercomputing Challenge

The Supercomputing Challenge (the Challenge) is an exciting program that offers a truly unique experience to students in our state. The opportunity to work on the most powerful computers in the world is currently available to only a very few students in the entire United States, but in New Mexico, it is just one of the benefits of living in the "Land of Enchantment."

The Challenge is a program encompassing the school year in which teams of students complete science projects using high-performance computers. Each team of up to five students and a sponsoring teacher defines and works on a single computational project of its own choosing. Throughout the program, help and support are given to the teams by their project advisors and the Challenge organizers and sponsors.

The Challenge is open to all interested students in grades 5 through 12 on a nonselective basis. The program has no grade point, class enrollment or computer experience prerequisites. Participants come from public, private, parochial and home-based schools in all areas of New Mexico. The important requirement for participating is a real desire to learn about science and computing.

Challenge teams tackle a range of interesting problems to solve. The most successful projects address a topic that holds great interest for the team. In recent years, ideas for projects have come from Astronomy, Geology, Physics, Ecology, Mathematics, Economics, Sociology, and Computer Science. It is very important that the problem a team chooses is what we call "real world" and not imaginary. A "real world" problem has measurable components. We use the term Computational Science to refer to science problems that we wish to solve and explain using computer models.

Those teams who make significant progress on their projects can enter them in the competition for awards of cash and scholarships. Team plaques are also awarded for: Teamwork, Written Report, Professional Presentation, Research, Creativity and Innovation, Environmental Modeling, High Performance, Science is Fun and the Judges' Special Award, just to name a few.

The Challenge is offered at minimal cost to the participants or the school district. It is sponsored by a partnership of federal laboratories, universities, and businesses. They provide food and lodging for events such as the kickoff conference during which students and teachers are shown how to use computers, learn programming languages, how to analyze data, write reports and much more.

These sponsors also supply time on the supercomputers and lend equipment to schools that need it. Employees of the sponsoring groups conduct training sessions at workshops and advise teams throughout the year. The Challenge usually culminates with an Expo and Awards Ceremony in the spring at the Los Alamos National Laboratory or in Albuquerque. During the Covid pandemic, three Expo and Awards Ceremonies, as well as two Kickoffs, were held virtually.

History

The New Mexico High School Supercomputing Challenge was conceived in 1990 by former Los Alamos Director Sig Hecker and Tom Thornhill, president of New Mexico Technet Inc., a nonprofit company that in 1985 set up a computer network to link the state's national laboratories, universities, state government and some private companies. Sen. Pete Domenici, and John Rollwagen, then chairman and chief executive officer of Cray Research Inc., added their support.

In 2001, the Adventures in Supercomputing program formerly housed at Sandia National Laboratories and then at the Albuquerque High Performance Computing Center at the University of New Mexico merged with the former New Mexico High School Supercomputing Challenge to become the New Mexico High School Adventures in Supercomputing Challenge.

In 2002, the words "High School" were dropped from the name as middle school teams had been invited to participate in 2000 and had done well.

In the summer of 2005, the name was simplified to the Supercomputing Challenge.

In 2007, the Challenge began collaborating with the middle school Project GUTS, (Growing Up Thinking Scientifically), an NSF grant housed at the Santa Fe Institute.

In 2013, the Challenge began collaborating with New Mexico Computer Science for All, an NSF funded program based at the Santa Fe Institute that offers a comprehensive teacher professional development program in Computer Science including a University of New Mexico Computer Science course for teachers.

2024—2025 Supercomputing Challenge Awards

35th Annual New Mexico Supercomputing Challenge winners

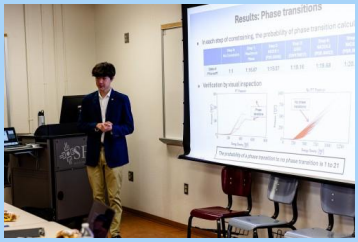
FIRST PLACE

Constraining the Neutron Star Equation of State with Observation Data

SCHOOL: Los Alamos High School

TEAM: Tate Plohr

SPONSOR: JeeYeon Plohr



VERY NICE WORK!!

SECOND PLACE

Point Cloud Surface Reconstruction

SCHOOL: Los Alamos High School

TEAM: Andrew Morgan

SPONSOR: Nathaniel Morgan

WELL DONE!!

Supercomputing Challenge students across the state have completed their computational projects on topics such as cancer, astrophysics, transportation, and environmental issues. This year the Supercomputing Challenge celebrated their 35th annual Expo and Awards Ceremony on April 25, 2025 which featured student final presentations, judging, tours and an award ceremony held at the Santa Fe Community College.

Sponsors

In this program, middle school and high school students are mentored by a community of volunteer scientists, computer programmers and professors. Several alumni also serve as volunteers. The Supercomputing Challenge partners include the New Mexico Consortium, Los Alamos National Laboratory, Sandia National Laboratory, Public Service Company of New Mexico, Bigbyte, Westwind, NMTIE, Simtable/Redfish Group and most New Mexico colleges and universities. A complete list of sponsors and supporters of the Challenge is on the website at https://supercomputingchallenge.org/24-25/sponsors. Do you want to become a supporter of the Supercomputing Challenge? Please email us at consult@supercomputingchallenge.org for details.

"These students never cease to amaze me with the breadth and depth of their projects," said David Kratzer, Executive Director of the Supercomputing Challenge. "We are so proud to showcase the incredible talent and persistence they bring to every phase of the competition."

By participating in the Challenge, students learn to be prepared and successful in any career or college. They learn to be persistent by practicing their computer programming skills, completing research, and meeting deadlines to cross the finish line. The Challenge likes to refer to itself as

an academic marathon, and these students should be recognized as critical thinkers, communicators, collaborators, and computer scientists.

Scholarships

Scholarships worth $5,400 were awarded to participating students. Many other awards were distributed ranging from $50 per team member for being finalists in the academic marathon to team prizes of up to $1000 for 1st and additional prizes for other categories such as teamwork, research, programming ability, and community impact. Random drawings were held for $50 door prizes.

The Supercomputing Challenge is open to New Mexico middle and high-school students, including home-schooled students. Students work in teams and follow their own interests to choose a topic to computationally model. New students interested in becoming involved can make plans to start the next academic year, by contacting consult@supercomputingchallenge.org

Finalist teams receive banners to hang in their schools

A complete list of all winning student teams.
https://supercomputingchallenge.org/24-25/expo-AllWinnersList.pdf

More about the Supercomputing Challenge

“The goal of the yearlong event is to teach student teams how to use powerful computers to analyze, model and solve real-world problems,” said David Kratzer, Executive Director. “Participating students improve their understanding of technology by developing skills in scientific inquiry, modeling, computing, communications, and teamwork.”

The New Mexico Supercomputing Challenge teaches written and oral communication, collaboration with peers and professionals, critical thinking including research and coding including computer modeling to middle and high school students throughout the state. Any New Mexico middle-school or high-school student, including home-schooled students are eligible to participate in the Supercomputing Challenge. Students follow their own interests to choose a topic to model.

Scholarship winners

Scholarships worth $5,400 were awarded at the Supercomputing Challenge Awards Ceremony to three seniors: Andrew Morgan, Ximena Serna and Elisea Jackson (not pictured).

All final reports are online.
https://supercomputingchallenge.org/24-25/final-reports-view

More information about the Supercomputing Challenge can be found at:
https://supercomputingchallenge.org

Teams Finishing the Challenge and submitting final reports:

School: Albuquerque Academy
Predicting the drug and micro-/nano-plastic interactions inside the body using molecular dynamics modeling and machine learning
Team Member: Ahana Koushik
Sponsor: Jay Garcia

School: Albuquerque Academy
Additive Manufacture Kinetics and Thermodynamics Model
Team Member: Harrison Schiek
Sponsors: Jay Garcia, Alex Benedict

School: Justice Code
The Impact Of Food Insecurity In New Mexico
Team Members: Mekhi Bradford, Lukas Lee Baires
Sponsor: Becky Campbell

School: Justice Code
THE EFFICIENCY OF MAGNETIC TRANSPORTATION
Team Members: Kolton Walker, Delight Emma-Asonye, Junior Offor, Leilani Baty
Sponsor: Becky Campbell
Mentor: Kevin Walker

School: La Cueva High School
You Only Look Once Machine Learning Solution to Orbital Debris Detection and Classification
Team Members: Hadwyn Link, Ximena Serna
Sponsor: Jeremy Jensen
Mentor: Mario Serna

School: Los Alamos High School
Constraining the Neutron Star Equation of State with Observational Data
Team Member: Tate Plohr
Sponsor: JeeYeon Plohr
Mentors: Ingo Tews, Rahul Somasundaram

School: Los Alamos High School
Point Cloud Surface Reconstruction
Team Member: Andrew Morgan
Mentor: Nathaniel Morgan

School: Mountain Elementary
Modeling Urban Heat Islands and Rural Areas
Team Member: Emmaline Fadner
Sponsor: Zeynep Unal
Mentor: Cristie Fadner

School: New Futures School
How Can Virtual Reality Help Kids Escape the Confines of Hospitals by Entering New Worlds?
Team Member: Xa'Ria Rush
Sponsor: Rachel Kilman
Mentor: Richard Barrett

School: New Futures School
Investigating the Effects of Screen Time on Infants Aged 0-2
Team Members: Neveah Birner, Emily Boucher, Chiara Haney, Amari Solomon-Iule
Sponsor: Rachel Kilman
Mentor: Gennie Barrett

School: New Mexico Academy for the Media Arts
The Effects of Mycorrhizal Fungal Networks and
Native Species on Plant Health in Arid Environments
Team Members: Eduardo Dorado, Ana Sofia Rodriguez, Zaaliyah Thomas
Sponsor: Tanya Mueller

School: New Mexico School for the Arts
Melenoma
Team Members: Elisea Jackson, Megan Odom
Sponsors: Sarah Rowe, Acacia McCombs
Mentor: Felina Rivera Calzadillas

School: Santa Fe Preparatory School
Investigating Intersubjective Realities From Novel NLP and Chaos Theory Approach
Team Member: Camila Carreon
Sponsor: Jocelyn Comstock
Mentor: Mark Galassi

School: Santa Fe Preparatory School
Utilizing YouTube Data as a Tool for Tracking the Spread of COVID-19
Team Member: Inho Ryu
Sponsor: Jocelyn Comstock
Mentor:

School: Santa Fe Preparatory School
Understanding and Predicting Trail Maintenance Needs Using Machine Learning Techniques
Team Members: Luke Rand, Isaac Olson
Sponsor: Jocelyn Comstock

School: Santa Fe Preparatory School
Simulating Interactions Between Varied Predators & Preys
Team Member: Marlow Lichty
Sponsor: Jocelyn Comstock
Mentor: Clint Hubbard

School: Truman Middle School
Exploring the Moon with VEX Robotics
Team Members: Josue Ochoa, Kaleb Martinez
Sponsor: Natali Barreto Baca

School: Truman Middle School
Growing Plants on Mars with CyberBot Robotics
Team Members: Carlos Cantu, Zinoc Fang
Sponsor: Natali Barreto Baca

School: Welch Homeschool
Modeling Fast Moving Objects in Crowded Astronomical Neighborhoods
Team Member: Kalliope Luna Welch
Sponsor: Cindy Welch
Mentor: Paul Welch

School: Welch Homeschool
Analyzing Pre-Indo-European Theory of Etruscan Language Origins Using Topological Data Analysis
Team Member: Helena Welch
Sponsor: Cindy Welch
Mentor: Paul Welch

Judges

Leila Alhemali, Simtable
Char Arias, Sandia National Laboratories/Retired
Richard Barrett, Sandia National Laboratory/Retired
Nick Bennett, TLG Learning
Hope Cahill, Santa Fe Public Schools
Sharmistha Chakrabarti, Los Alamos National Laboratory
Rusty Davis, Los Alamos National Laboratory
Creighton Edington, Rural Education Advancement Program
Susan Gibbs, Project GUTS
Stephen Guerin, Simtable
Omar Ishak, Los Alamos National Laboratory
Elizabeth Yakes Jimenez, University of New Mexico
Philip Jones, Los Alamos National Laboratory

Will Jones, Los Alamos National Laboratory
David Kratzer, Los Alamos National Laboratory/Retired
Maximo Lazo, Central New Mexico College
Scott Levey, Sandia National Laboratories
Kasra "Kaz" Manavi, Simtable
Patty Meyer, Supercomputing Challenge
Ziyi Niu, Eastern New Mexico University
Joseph Olonia, Central New Mexico Community College
James Overfelt, Sandia National Laboratories
Mark Petersen, Los Alamos National Laboratory
Lee Rand, Sun Mountain Capital
Lonnie Rednour, San Juan College
Dana Roberson, Central New Mexico College
Thomas Robey, Gaia Environmental Sciences
Dorian Sims, Westwind Computer Products Inc.
Tim Thomas, Sandia National Laboratories
Michael Trahan, Sandia National Laboratories
Geoff Valdez, Los Alamos National Laboratory
Eduardo Ceh Varela, Eastern New Mexico University
Anneliese Ward, University of New Mexico
Kyreen White, University of NM/Challenge Alumni
Christin Whitton, Los Alamos National Laboratory
Kaley Woelfel, BlueHalo

Do you want to become a supporter of the Supercomputing Challenge?
Please email us at consult@supercomputingchallenge.org for details.

Constraining the Neutron Star Equation of State with Observational Data

New Mexico
Supercomputing Challenge
Final Report
April 1, 2025

Los Alamos High School
Tate Plohr

Teacher:
JeeYeon Plohr

Mentors:
Ingo Tews, Rahul Somasundaram

EXECUTIVE SUMMARY

Neutron stars are remnants of the cores of massive stars after they undergo supernovae, powerful explosions, at the end of their lives. Neutron stars are so dense that electrons are forced into atomic nuclei, where they combine with protons to form neutrons. They were theoretically predicted in the 1930s and first observed in the 1960s. Since then, much research has been done to understand the properties of the neutron stars. In particular, the Equation of State for the neutron star is an active research area for its importance in astrophysics simulations as well as in condensed matter physics. In the last few years, unprecedented observational data from the Neutron star Interior Composition Explorer (NICER) and the Laser Interferometer Gravitational-wave Observatory (LIGO) on neutron stars became available, making it possible to test out numerous theories.

In nuclear physics, there are many proposed Equations of State (EOSs) for neutron stars based on theories with various assumptions and approximations. To find the true EOS for neutron stars, we adopted a data-oriented, model-agnostic approach, based on random sampling and Bayesian analysis. Starting with many possible EOSs, we utilized recent observational data of neutron stars to constrain these candidates. The prediction of the 90th percentile range of the radius of a 1.4-solar-mass neutron star provides a quantitative measure of the constraint and can be compared to other works.

One important question to be answered by research on the neutron star EOS is the composition of the core. That is, whether the cores of neutron stars are made of nucleonic matter, like the outer layers, or instead, it consists of quarks or exotic states of matter. Using the constrained set of EOSs, we investigated the state of matter at the core. By analyzing the structure of the likely EOS curves, I find the probability that the core is made of nucleons relative to the probability of deconfined quark matter. I find that nucleonic matter is strongly favored over deconfined quark matter. Specifically, nucleonic matter is 21 times more likely than quark matter.

Table of Contents

I. Introduction
1.1 Neutron Stars and EOS
1.2 Tolman-Oppenheimer-Volkoff (TOV) Equations

II. Data
2.1 Maximum Mass Data
2.2 NICER
2.3 LIGO

III. Methods

IV. Procedure

V. Computation
5.1 TOV solver
5.2 Probability Density Function: Discrete Data and Kernel Density Estimator (KDE)
5.3 Posterior Analysis: Phase Transition Locator

VI. Results

VII. Conclusion

Acknowledgements

Appendix: Bimodal Analysis of PSR J0030+0451

References

I. INTRODUCTION

1.1 Neutron Stars and Equation of State

All stars begin as stellar nebulae and are stabilized by nuclear fusion. Over millions or billions of years, they exhaust their fuel for fusion and can evolve into red giants or supergiants. The most massive stars undergo a supernova explosion, leaving behind a compact remnant. Lighter stars form white dwarfs, medium-mass stars evolve into neutron stars, and the most massive stars collapse into black holes. In this paper, I examine neutron stars, with a particular focus on their Equation of State (EOS), which is essential for understanding their properties.

Neutron stars are the densest observable objects in the universe, formed during the supernova explosions of red supergiant stars. A typical neutron star weighs about 1.4 times the mass of the Sun and has a radius between 10 and 12 km (Lattimer and Prakash, 2001), making the neutron star's core density exceed that of atomic nuclei. They are so dense that one sugar cube of neutron star matter would weigh 10 billion tons on Earth, or roughly 30,000 Empire State Buildings. However, we do not have reliable models of the extreme densities inside neutron stars. This is why studying neutron stars is crucial, since it allows physicists to create more refined models describing all matter at densities not currently achievable on Earth.

Neutron stars have several distinct layers: an atmosphere, outer crust, inner crust, outer core, and inner core (see Fig. 1). The atmosphere is a thin layer composed of hydrogen, helium, and carbon. Beneath it lies the solid outer crust, made up of ions and electrons arranged in a lattice structure. Below the outer crust is the inner crust, where nuclei are compressed so tightly that they can form “nuclear pasta” (Caplan and Horowitz, 2017). In the outer core, a neutron superfluid and a proton superconductor coexist. Finally, there is the inner core, where the composition is unknown due to the extremely high density. It may consist of neutrons and protons like the outer core, or it could be made of exotic matter, such as deconfined quark matter or (keep? ***) “hyperons,” which are subatomic particles that weigh more than protons and neutrons (Benhar, O. *et al.* (eds.), 2024).

Neutron stars provide an effective laboratory for studying matter and the laws of physics at extreme densities on the threshold of forming a black hole. Constraining the neutron star EOS improves our understanding of the interaction between the strong force and gravity.

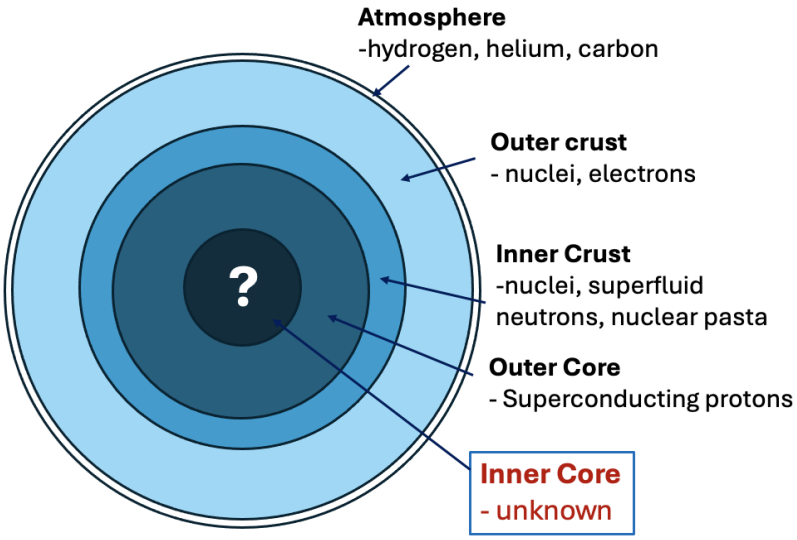


Figure 1: Inner structure of a neutron star.

The Equation of State (EOS) is a relationship between thermodynamic state variables such as density, pressure, temperature, energy, etc. For neutron star matter, we generally relate the pressure and the energy density or number density. At low densities, Chiral Effective Field Theory (Chiral EFT) provides a well-established description of the strong force between nucleons (neutrons and protons) (Epelbaum *et al.*, 2008). However, Chiral EFT breaks down at high densities. Beyond the breakdown density, many models have been proposed to describe the higher-density regime, each depending on different underlying physical assumptions and approximations. The uncertainty in this regime is represented by the blue region in Fig. 2. Even the precise value of the breakdown density remains uncertain (Capano *et al*., 2020). In this work, I assume the breakdown density to be 1.5 n_s, where n_s is the saturation density ($1.6 / fm^3$), the density of an atomic nucleus. This follows Dietrich *et al.,* 2020.

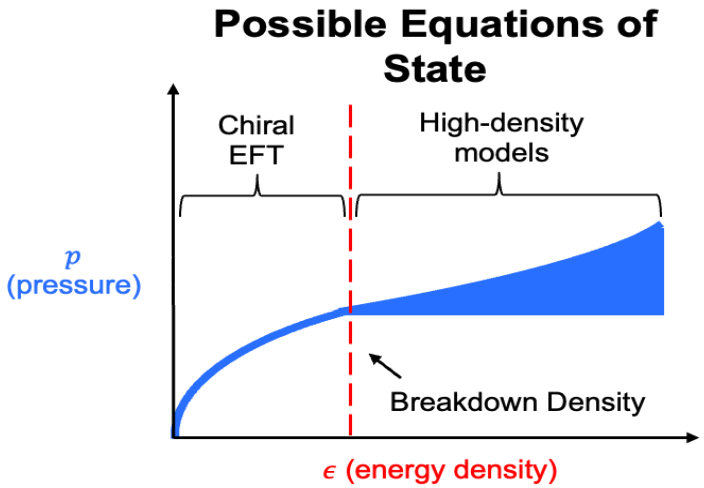


Figure 2: The Nuclear Matter Equation of State. The low-density regime is well constrained by theory. The regime beyond the breakdown density, however, is not well understood, and many possible EOSs have been proposed.

The purpose of this paper is to search for the unified EOS that describes all neutron stars. My approach involves generating 5,000 EOS samples that are constructed using Chiral EFT in the low-density regime and are random in the high-density regime (Tews, 2024). I use random samples to explore all possibilities in the high-density regime, to account for the numerous models that have been proposed. By adopting a model-agnostic approach, I allow the data to determine which EOSs are more probable. Following Dietrich et al., 2020, I construct an analysis framework with a Python program that converts the EOS curves into mass-radius curves, which are explained in the next section. Then, I combine the analysis of multiple neutron star datasets with existing nuclear theory to find the most probable EOSs using Bayesian statistical methods.

This framework is based on the hypothesis that, even though neutron stars have different sizes and masses, they all follow the one, true Neutron Star EOS. Therefore, every neutron star must fall somewhere on the one, true mass-radius curve. I am trying to find this mass-radius curve that predicts the mass-radius observations of several neutron stars. Once I calculate the probabilities of the EOS curves, I examine their predictions of neutron star properties. In particular, **the composition of the inner core, i.e., whether it is nucleonic (protons and neutrons) like the outer core or deconfined quark matter (QM), is encoded in the EOS curves.** To illustrate this point, let us consider water and vapor. When vapor condenses to water,

its density jumps with the sound speed increasing in proportion. This is manifested as a plateau in pressure vs. energy density. We can make analogous statements about nucleonic and deconfined quark matter. If the core consists of neutrons and protons, pressure will increase monotonically with energy. On the other hand, if the matter in the core has gone through a phase transition to become quark matter (QM), the pressure-energy curve will have a plateau. In other words, a change in the composition is likely manifested in the structure of EOS curves. Therefore, by analyzing the structure of the more probable EOS curves, I find out what the core is made of.

By reducing the uncertainty in the Equation of State, I identify which high-density models of particle physics are most accurate and best describe the properties of neutron stars. The EOS plays a crucial role in astrophysical simulations, so finding the true EOS helps us understand many phenomena of neutron stars. For example, when two neutron stars merge, new elements are created through what is known as the rapid process or r-process. A large fraction of natural elements heavier than iron are formed in such events, meaning that most objects in everyday life are made of elements from neutron stars. Physicists try to understand the r-process using computer simulations where the EOS is an essential component. Therefore, knowledge of the state of neutron stars is crucial in understanding how these elements were made and why some elements exhibit certain properties (Cowan *et al*. 2021).

1.2 Tolman-Oppenheimer-Volkoff Equations and Mass-Radius curves

To constrain the Neutron Star EOS with mass-radius data, I calculate the masses and radii of neutron stars predicted by each EOS by solving the Tolman-Oppenheimer-Volkoff (TOV) Equations, which describe a spherically symmetric object, such as a neutron star (Oppenheimer and Volkoff, 1939; Tolman, 1939; Camenzind, 2007; Gandolfi *et al.*, 2019). The details of a solver I wrote, using the forward Euler method in Python are in the appendix. These equations are based on the equilibrium between relativistic gravity and the pressure in nuclear matter. The TOV equations are Ordinary Differential Equations (ODEs):

$$\frac{dm}{dr} = 4\,\pi\, r^2\, \epsilon(P), \tag{1}$$

$$\frac{dP}{dr} = \frac{-G\, m(r)\, \epsilon(P)}{r^2\, c^2}\left(1 + \frac{4\,\pi\, r^3\, P}{m(r)\, c^2}\right)\left(1 + \frac{P}{\epsilon(P)}\right)\left(1 - \frac{2\, G\, m(r)}{r\, c^2}\right)^{-1}, \quad (2)$$

where $m(r)$ is the mass within the sphere of radius r; $P(r)$ is the pressure on the surface of that sphere; the total energy density $\epsilon(P)$ is specified by the EOS. The first factor corresponds to Newtonian gravity, the next three terms are relativistic corrections. To model a neutron star, I solve a boundary condition problem for the TOV equations with many choices of the central pressure P_c to find the total mass M and radius R of the neutron star. For each choice of P_c, I integrate the TOV equations with respect to r from $P = P_c$ and $m = 0$ at $r = 0$ until the pressure drops to zero, at which point $r = R$ and $m = M$. In this way, I obtain a curve relating the total mass M and radius R of the neutron star for a range of choices of the central pressure, which is the mass-radius curve for the EOS. A more detailed explanation of how to solve the TOV equations is in the Appendix.

In Fig. 3, I illustrate the result of integrating the TOV equations for an EOS to give its mass-radius curve. There is a one-to-one correspondence between the two curves so the probability of the mass-radius curve is equal to the probability of the corresponding EOS. In Fig. 4, I show my 5000 EOS samples and their corresponding mass-radius curves, obtained by my TOV solver.

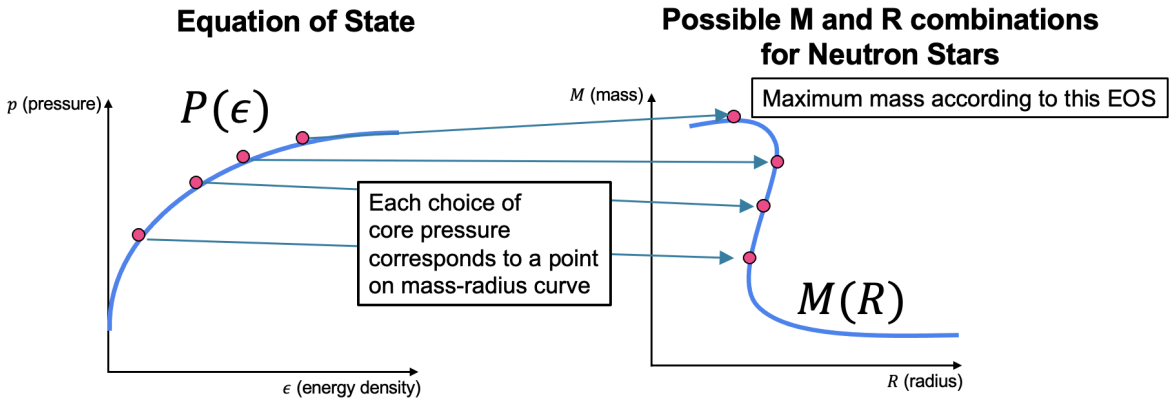


Figure 3: Integrating the TOV equations for each EOS results in a curve in the mass-radius plane.

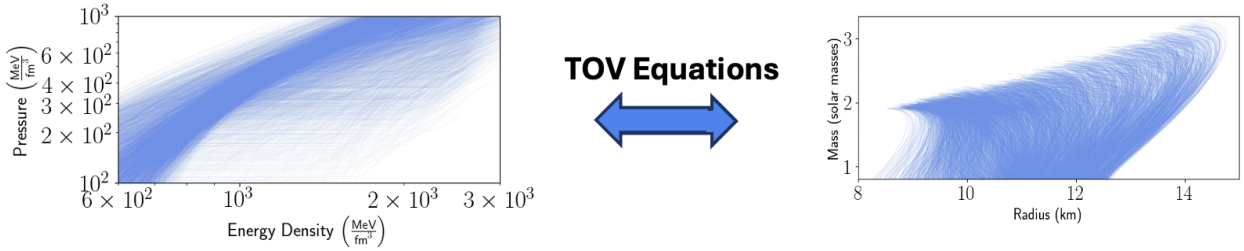


Figure 4: Neutron star mass-radius curves predicted by 5000 sample EOSs.

II. DATA

In this section, we review the three types of observational data on neutron stars that have been used in this work: data from radio observations, NICER X-ray observations, and LIGO gravitational-wave data. These public datasets are available in the form of tables. For instance, NICER data have thousands of pairs of mass and radius values. Collectively, they represent a probability distribution. The areas where there are more data points (of mass and radius) correspond to higher probabilities. I convert this table of discrete points into a probability distribution function using a Kernel Density Estimator or KDE, which is a standard statistical method. I utilized a KDE method in SciPy, a Python library. More details about the KDE can be found in the appendix.

2.1 *Maximum Mass Data*

The maximum mass data helps determine which mass-radius curve is more probable based on its prediction of the maximum mass of a neutron star. This data includes three pulsars that are contenders for the most massive neutron star observed (although we cannot be certain which one is the most massive due to measurement uncertainties). These massive pulsars provide a lower bound for how massive neutron stars can be (Rezzolla, *et al.*, 2018).

The data also includes a gravitational-wave event where two neutron stars collided, resulting in the lowest mass black hole we have observed. This black hole provides an upper bound on the maximum mass a neutron star can have (Abbott et al., 2017).

2.2 *NICER Data*

The Neutron Star Interior Composition Explorer (NICER) is an X-ray telescope onboard the International Space Station (ISS) that observes pulsars, rapidly rotating neutron stars. Pulsars have magnetic fields that emit X-rays from areas known as hotspots. NICER observes these

X-rays and obtains a “pulse profile.” These pulse profiles do not reveal much on their own, so researchers perform simulations of neutron stars with various parameters, including mass and radius. By comparing the resulting pulse profile from the simulation with the real pulse profile, mass and radius can be estimated. Among numerous neutron star observations, three analyses of neutron star pulse profiles are considered most credible and are utilized in this paper. For simplicity and clarity, I label these three NICER datasets in Table 1.

	NICER 1	NICER 2	NICER 3
Neutron star (pulsar)	PSR J0030+0451 (Miller *et al.,* 2019; Raaijmakers *et al.,* 2019)	PSR J0437-4715 (Choudhury *et al.*, 2024)	PSR J0740+6620 (Miller *et al.,* 2021; Riley *et al.*, 2021)

Table 1: Labels for the NICER data

I note that the PSR J0030+0451 pulse profiles have recently been reanalyzed using an updated framework (Vinciguerra *et al.*, 2024) that incorporates extra information. The new analysis yielded two modes in mass and radius, meaning two distinct pairs of mass and radius are likely. I identify which mode of mass and radius is more consistent with the observations of other neutron stars and present the result in an Appendix.

2.3 *LIGO/VIRGO Data*

The Laser Interferometer Gravitational-wave Observatory (LIGO) consists of two large observatories aimed at detecting gravitational waves using laser interferometry. VIRGO, named after a galaxy, is another such observatory in Europe. The collaboration between multiple detectors is crucial for gravitational wave measurement. Gravitational waves are released when two massive objects rapidly orbit each other and cause ripples in spacetime. These gravitational waves depend on the Chirp Mass (a combination of the masses of the two objects), the mass ratio between the objects, and the tidal deformability of each object, among other parameters. Roughly speaking, a greater Chirp Mass leads to a higher frequency and a larger change in frequency over time. Additionally, if the combined radius is smaller, the peak frequency is higher. Similar to NICER, researchers determine properties of neutron stars using simulations of gravitational-wave signals that are compared with observations (Abbott et al., 2017).

III. METHOD

In order to calculate the probability of an EOS using data, I utilized Bayesian analysis, a statistical method for finding the probability of a statement (e.g., that a hypothesis holds or certain data is observed) given another statement (Kurt, 2019). Bayesian analysis is based on Bayes' theorem, which states that the conditional probability of hypothesis H given data D equals the conditional probability of D given H times the probability of H divided by the probability of D:

$$P(H|D) = \frac{P(D|H)\,P(H)}{P(D)}, \quad (3)$$

which follows from the definition of conditional probability, $P(H|D) = P(H\ and\ D)/P(D)$. Bayes' Theorem provides a rule for updating the prior probability, $P(H)$, of the hypothesis to the posterior probability, $P(H|D)$, to account for the data D.

In the context of this paper, the conditional probability of an EOS given several datasets is equal to the product of the conditional probabilities of each dataset:

$$P(EOS|data) = P(EOS|M_{max}) \times P(EOS|NICER) \times P(EOS|LIGO) \quad (4)$$

since each observation is independent from each other. With Bayes' theorem, the probability of an EOS given data is proportional to the probability of each dataset given an EOS multiplied by the unconditional probability of that EOS. For instance,

$$P(EOS|M_{max}) = P(M_{max}|EOS)\,P(EOS)\,/\,P(M_{max}), \quad (5)$$

With similar relations for $P(EOS|NICER)$ and $P(EOS|LIGO)$, we get:

$$P(EOS|data) \propto P(M_{max}|EOS) \times P(NICER|EOS) \times P(LIGO|EOS) \quad (6)$$

I illustrate my Bayes' theorem methodology in Fig. 5. Because neutron star data is given in mass and radius, I start with many possible EOS candidates (a) and convert them into mass-radius curves (b). Then, in (c), I find two curves (colored yellow) that are most probable

according to dataset 1. Since the dataset consists of discrete points, I construct a probability density function using KDE, available in the SciPy Python library. The basic idea is that more data points in an area represents higher probability. To constrain the mass-radius curves, I calculate the probability of each EOS using Bayesian statistics. This involves determining the probability of each mass-radius point along the curve and summing to obtain the probability of that entire mass-radius curve. Next, in (d), I compute the joint probability using both datasets. The two datasets imply that the one curve (colored red) is more probable. Finally, in (e), I map the mass-radius curves back to their respective EOSs.

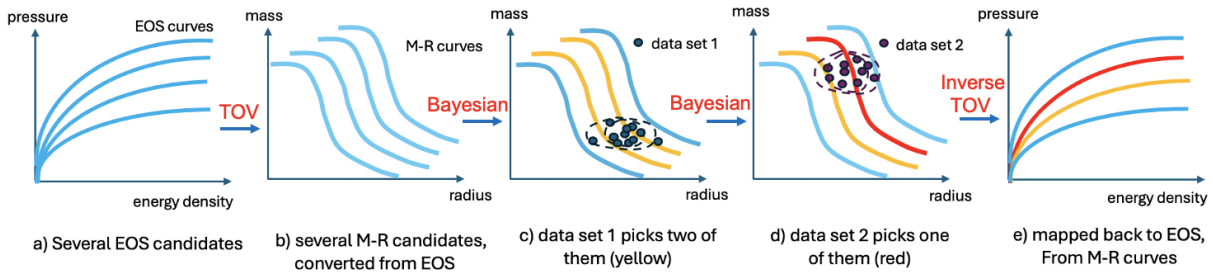


Figure 5: Schematic representation of how I constrain the EOSs using data. a) I start with several, equally-probable EOS candidates, which is indicated by the same blue color. b) Each EOSs is mapped to a mass-radius curve. c) Dataset 1 identifies two mass-radius curves (in yellow) as more probable. d) Dataset 2 further constrains the mass-radius curves by assigning higher probability to the red EOS. e) Finally, the mass-radius curves are mapped back to their respective EOSs along with their probabilities.

IV. PROCEDURE

The procedure of constraining EOSs is based on the methodology in the previous section; the flowchart of Fig. 6 follows the schematic diagram of Fig. 5. After converting EOS to mass-radius curves, I applied five datasets from NICER and LIGO/VIRGO (steps 1-5), and then converted the mass-radius curves with their associated probabilities back into the EOS curves. Afterwards, I analyze the EOSs to find the composition of neutron star cores. All computations were performed by Python programs written by the author.

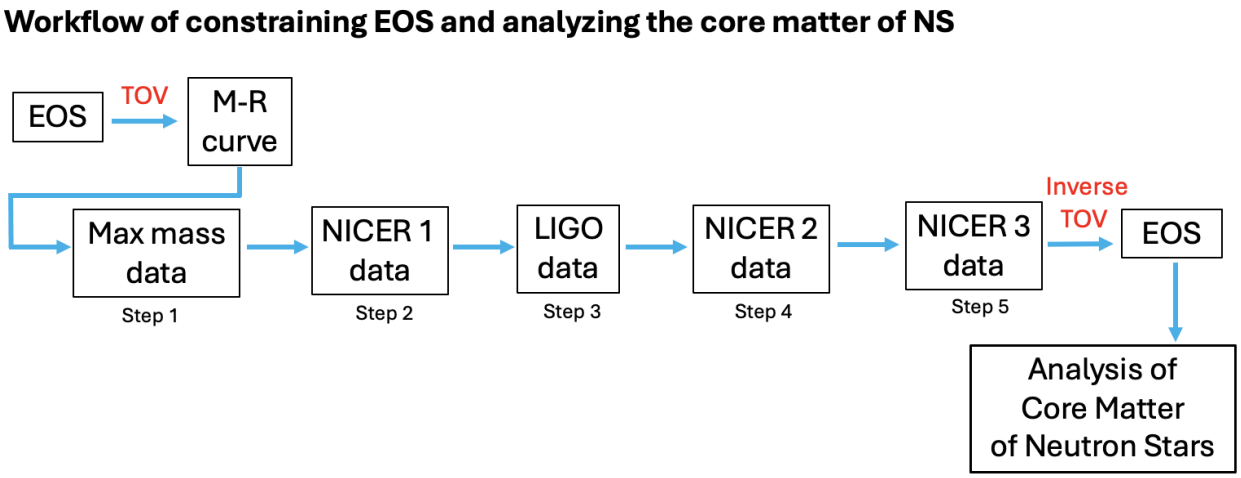


Figure 6: Flowchart of the multistep procedure. From the EOS samples, mass-radius curves are constructed by solving the TOV equations. These curves are then progressively constrained by incorporating the data (step 1-5). Next, the mass-radius curves are mapped back to EOS space. Finally, neutron star cores are analyzed using the probabilities of the EOSs.

To investigate the composition of the core, I separate the EOSs into two categories: those with a first-order phase transition and those without. I use my Python program to identify sections of the EOS where the sound speed is zero for a considerable range, which corresponds to a phase transition. After separating the EOSs, I find the relative probability between no phase transition and a phase transition, which corresponds to nucleonic matter and quark matter, respectively. I sum the probabilities of the EOSs without a phase transition and divide by the sum of the probabilities of the EOSs with a phase transition. Finally, I account for the bias that 11% of the 5000 EOSs have a phase transition.

V. COMPUTATION

I have written three programs for this project. The first is the TOV solver that calculates mass-radius for each EOS. The second is constraining EOS based on the data. And the last is for analyzing the constrained EOS to find the probability of phase transition in the neutron star cores. In this section, I explain these three codes. All codes are written in Python and provided along with the report.

5.1 TOV Solver

The Tolman-Oppenheimer-Volkoff equations for a spherically symmetric body consist of Eqs. 1 and 2, which represent the balance between relativistic gravity and the strong nuclear force. After setting $c = 1$ and simplifying, the TOV equations become

$$dm/dr = 4\,\pi\, r^2\, \epsilon(P)\ ,$$

$$dP/dr = -\ G\,[\epsilon(P) + P]\,\frac{m + 4\,\pi\, r^3\, P}{r\,(r - 2\,G\,m)}\ .$$

The TOV equations form a system of ordinary differential equations (ODEs) to be solved for certain initial conditions:

$$dy/dx = f(x, y),$$
$$y(x_0) = y_0.$$

I wrote a solver that integrates these equations using the forward Euler method. The discretized equations are

$$x_{k+1} = x_k + \Delta x,$$
$$y_{k+1} = y_k + f(x_k, y_k)\,\Delta x.$$

These equations are applied for $k = 0,\ 1,\ 2,\ \ldots$ until a stopping criterion is triggered.

For the TOV equations, y stands for the mass m and pressure P while x stands for the radius r. To model a neutron star, I start at the core of the star ($r = 0$) and integrate until the surface of the star is reached ($r = R$). The pressure at $r = 0$ is the core pressure P_c, whereas the mass is zero at $r = 0$. These values give the initial conditions. Since the core pressure is unknown and different in each neutron star, I choose a range of values for P_c. (See Fig. 3, where various core pressures correspond to their respective points on the mass-radius curve.) In contrast, the

pressure at the surface $r = R$ is zero because it is in equilibrium with the vacuum, whereas m at the surface equals the mass M of the star. The stopping criterion is $P = 0$, and then R is the value of r and M is the value of m.

However, it is important to note that the first step of the integration requires special treatment. The reason is that the right-hand side of Eq. (2) (the equation for pressure) is singular when $r = 0$. I analyzed the singularity and circumvented it in the following way.

While r remains small, $\epsilon(P)$ stays close to the constant $\epsilon_c = \epsilon(P_c)$. Therefore, Eq. (1) implies that

$$m \approx (4\pi r^3/3)\,\epsilon_c.$$

This result is easy to understand: the mass equals the mass-energy density times the volume of a sphere of radius r. After making this substitution, Eq. (2) becomes

$$dP/dr \approx -G(\epsilon_c + P_c)\frac{(4\pi r^3/3)\,\epsilon_c + 4\pi r^3 P_c}{r\left(r - 2G(4\pi r^3/3)\,\epsilon_c\right)}.$$

Since r^3 is much smaller than r, the denominator is approximately r^2, so that

$$dP/dr \approx -4\pi G(\epsilon_c + P_c)(\epsilon_c/3 + P_c)\,r.$$

Hence, for small r, the pressure approximately equals

$$P \approx P_c - 2\pi G(\epsilon_c + P_c)(\epsilon_c/3 + P_c)\,r^2.$$

These approximations replace the first step of the integration. After the first step, the forward Euler method is used.

5.2 Probability Density Function from Discrete Data using Kernel Density Estimator (KDE)

A Kernel Density Estimator (KDE) is a statistical tool to obtain a probability density function, given a data sample. That is, using KDE, we get a smooth function that approximates the underlying probability that could have generated the data themselves. In a sense, it is similar to making a histogram of data and connecting the bars to get a continuous function. To use the KDE, we need to specify the kernel function (K), a smooth function that peaks where the data point is, and the bandwidth. By overlapping the kernel functions with their centers at the data points, we get an estimate of the underlying functions for the data:

$$\widehat{f}(x) \ = \Sigma_{observation} K(\frac{x-observation}{bandwidth})$$

where $\widehat{f}(x)$ is the estimated probability density and the choices for kernel function and bandwidth are problem-dependent.

For the neutron star mass-radius data, I used a 2D KDE with a Gaussian kernel from which I calculated the probability of each mass-radius curve: I picked the fixed number of discrete points along a curve, calculated the probability at each point and then added them up to find the probability of that particular mass-radius (or equivalently, EOS) curve. I repeat this process for the 5000 mass-radius curves. Since I had multiple sets of data, there is a different KDE for the new dataset added in each step, described in Fig. 6.

5.3 Posterior Analysis: Phase Transition Locator

Phase transitions from nucleonic matter to deconfined quark matter are embedded in some of the EOSs I considered. They were randomly generated at different energy density and pressure for random strength (energy required to complete the transition). I wrote a Python script to locate any phase transition which would be manifested as a plateau in the pressure vs. energy

density curves. Specifically, for each EOS, I calculated slopes at grid points and found where they are sufficiently small. One requirement is that the difference in energy density from the start to the end of the phase transition must be at least 50 MeV fm^{-3}. The phase transition must also start within the range of core energy densities, from 150 MeV fm^{-3} to the maximum described by the EOS (Tews, 2024). I performed a further analysis on phase transitions in neutron stars by finding pair correlations among the initiation energy density, max slope, and strength of the phase transition using the corner module in Python.

VI. RESULTS

After generating the neutron star EOSs, converting them into mass-radius curves using the TOV equations, and obtaining mass-radius probability distributions from data, I applied Bayes' theorem to determine the probability of each EOS. As a quantitative measure of how much the mass-radius curves have been constrained, I also calculated the 90th percentile radius interval for a 1.4-solar-mass neutron star.

Step 1: Maximum Mass

We start with the maximum mass data to constrain the mass-radius curves. Each curve predicts a maximum mass, and if that prediction is lower than what has been observed, that particular curve cannot describe massive neutron stars. Therefore, we can exclude such mass-radius curves. In Fig. 7 b), more probable mass-radius curves are dark red while less probable curves are lighter. Since the maximum mass data itself has a probability distribution, the mass-radius curve that predicts the most likely maximum mass has a higher probability than a mass-radius curve with a different maximum mass. The green lines indicate the 90th percentile range of the maximum mass, which I call the upper and lower bounds.

Using these constrained mass-radius curves, we plot the probability distribution for the radius of a 1.4-solar-mass neutron star (Fig. 8b). The 90th percentile range of the predicted radius is $11.52^{+1.31}_{-1.58}$ km.

After constraining the mass-radius curves, I found that the odds of a first-order phase transition (FOPT) is about 1/18, indicating that most EOSs with FOPT are discarded.

Steps 2, 3, 4, and 5: NICER 1, LIGO, NICER 2, and NICER 3

Having selected the EOSs that satisfy the maximum mass constraint, I apply the NICER 1 (PSR J0030+0451) data. The green contours (at 1 and 2 standard deviations) in Fig. 7c represents the high probability region according to NICER 1. Hence, the EOS curves that go through the green contours are evaluated to have high probability. In Fig. 8c, I plot the probability distribution of the radius of a neutron star at 1.4 solar mass.

I progressively use the LIGO (GW170817) and the other NICER data in addition to the previous data. The LIGO data comes from observing two merging neutron stars. The green contours in Fig. 7d represent the more massive neutron star, and the blue contours represent the less massive neutron star. The NICER 2 (PSR J0437-4715) data, shown in Fig. 7e, has not been used in previous work. Lastly, I use the NICER 3 (PSR J0740+6620) data. This pulsar is heavy, at about 2 solar masses, and its NICER data has also not been used previously. As seen in Fig. 7f, I now have a much smaller number of EOS that are probable, according to the data. I note the population of the allowed EOSs is reduced as I constrain them with more data. In Fig. 8f, I plot the final radius probability distribution of a 1.4-solar-mass neutron star. The final distribution has a much narrower peak at about 11.5 km, compared with Fig. 8a, for which the radius is unconstrained.

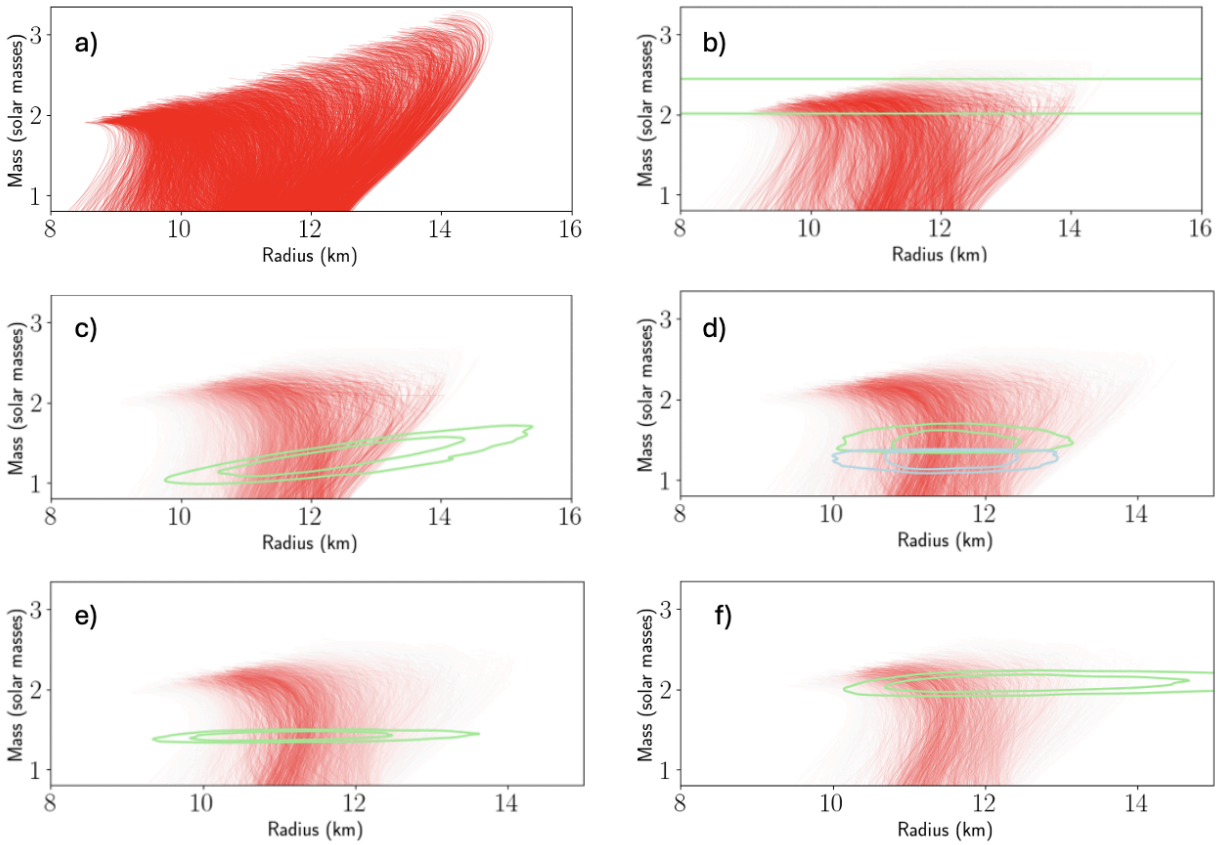


Figure 7: Constraining the mass-radius curves using the data. a) The initial set of 5000 EOSs with no constraints. b) Mass-radius curves constrained by the maximum mass data from PSR J0740+6620, PSR J0348+4032, PSR J0614-2230, and GW170817. The green lines represent the upper and lower mass bounds. c) Mass-radius curves further constrained by the NICER data of PSR J0030+0451. The green contours are at the 95% and 68% confidence level. d) Further constraints by LIGO (GW170817) data. Green contours are at the 95% and 68% confidence level for the more massive neutron star, with the blue contours representing the less massive neutron star. e) Further constraints on mass-radius curves by PSR J0437-4715 data. f) Further constraint by PSR J0740+6620.

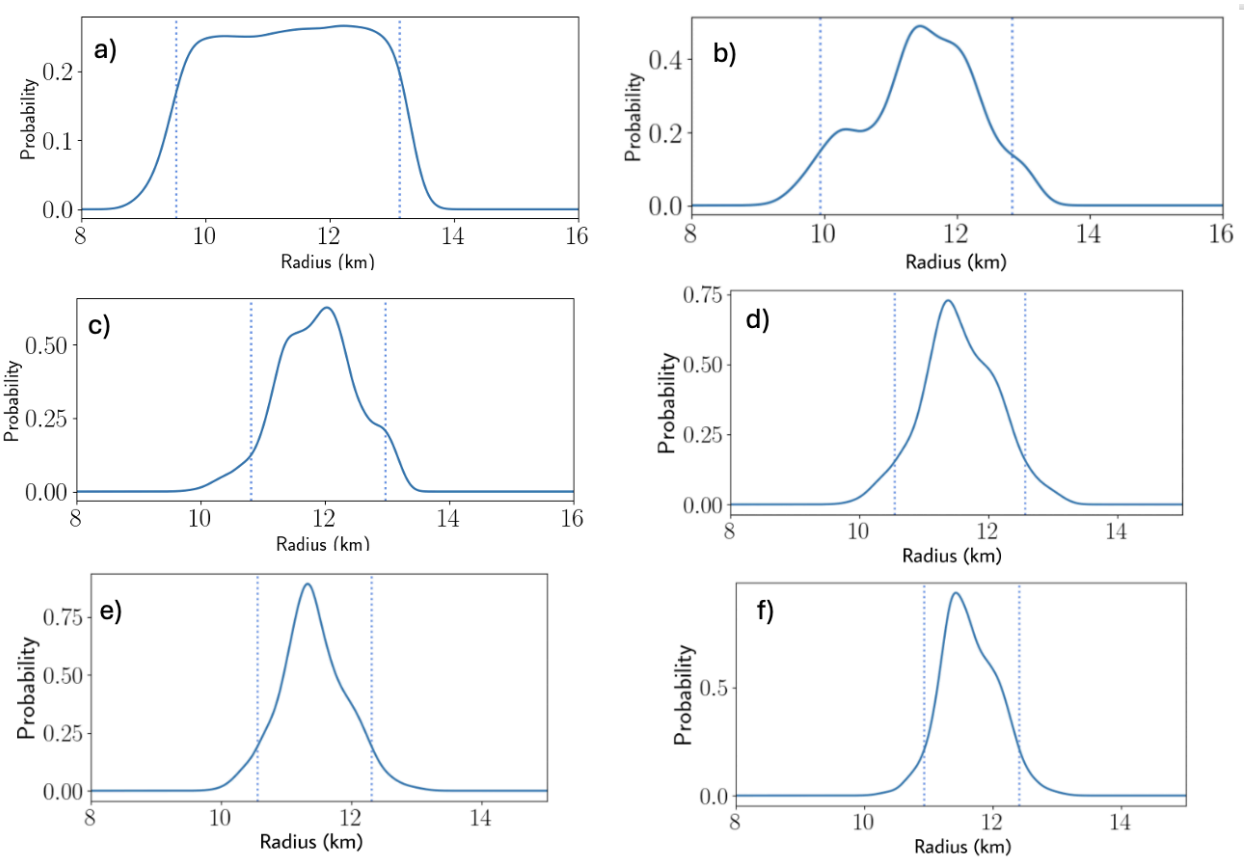


Figure 8: Probability distribution of radius for a neutron star with 1.4 solar masses. a) Before constraining. b) Constrained by the maximum mass data, c) the NICER 1 data, d) the LIGO data, e) the NICER 2 data, and f) the NICER 3 data.

In Table 2, I summarize the predictions of the radius of a 1.4-solar-mass neutron star from each step. The uncertainty is reduced in each step.

	Initial set	**Max mass**	**NICER 1**	**LIGO**	**NICER 2**	**NICER 3**
The radius of a 1.4 solar mass NS (90% confidence, km)	$11.37^{+1.75}_{-1.85}$	$11.52^{+1.31}_{-1.58}$	$11.88^{+1.09}_{-1.09}$	$11.53^{+1.04}_{-0.98}$	$11.38^{+0.92}_{-0.83}$	$11.60^{+0.81}_{-0.67}$

Table 2. The radius prediction of a neutron star with 1.4 solar mass. The intervals are the 90th percentile intervals and correspond to the vertical lines in Figs. 8a, 8b, 8c, 8d, 8e, and 8f.

Now that we have obtained the allowed set of EOSs, we can compare them with theory-based models. In Fig. 9, we plotted the predictions of mass and radius by the various models (Lattimer and Prakash, 2001, Gandolfi et al., 2019) overlapped onto Fig. 7 (f). Each of these models assume a certain composition of the core: the orange curves predict only quark matter (Prakash, M. *et al.,* 1995), the blue ones for nucleonic (Akmal and Pandharipande, 1997, Müller and Serot, 1996, Wiringa, R. B, 1988), and the green curves for mixtures of nucleonic and exotic matter (Prakash, M. *et al*, 1995, Glendenning *et al.,* 1991, Glendenning *et al.,* 1999). Clearly, many of the models don't agree with the current data-based predictions. Those that fall in the range of the allowed EOSs are for the nucleonic matter, hence it implies that the phase transition to quark matter or other exotic matter is unlikely.

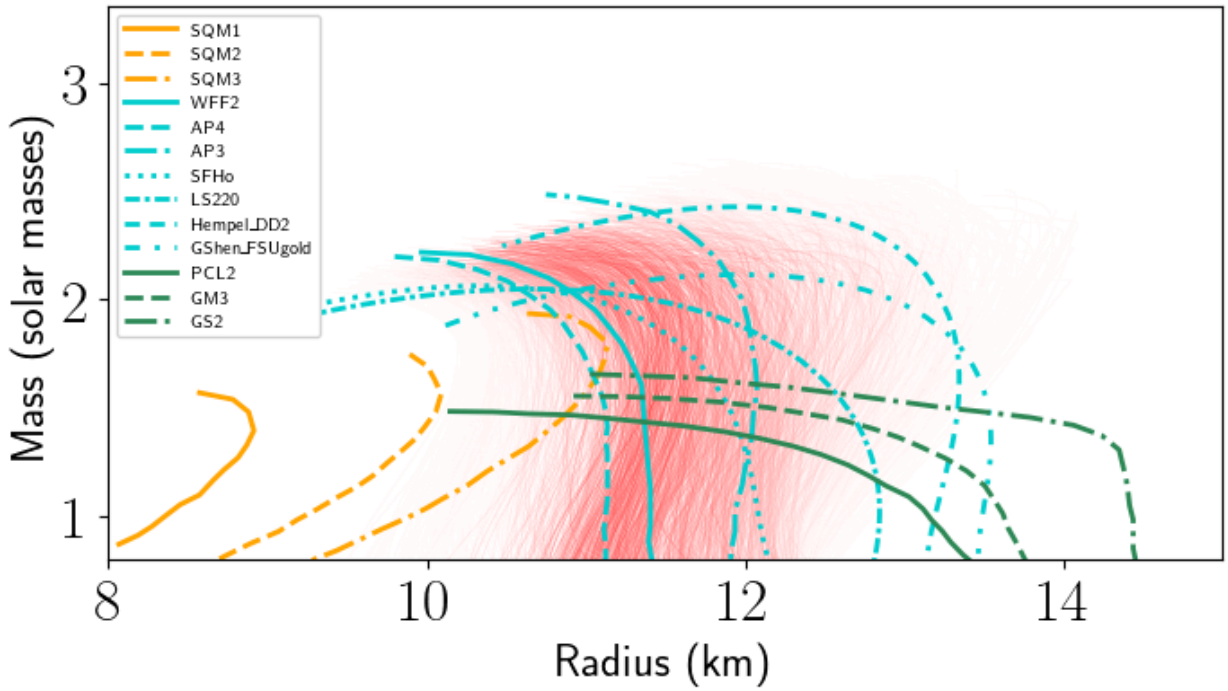


Figure 9: Comparison between the theory-based models and the data-based predictions of mass-radius curves. The orange curves are for quark matter only, the blue ones are for nucleonic matter, and the green curves for nucleonic, quark, and hyperon mixtures.

Having found the probabilities of the EOSs, I mapped the mass-radius curves back to EOS space. In Fig. 10, I show the spread of all EOSs I considered in dark blue and the allowed EOSs in light blue in pressure vs. energy density space. Up to the energy density at about 250

MeV/fm^3, all EOSs obey Chiral Effective Field Theory and there is little uncertainty. Beyond that point, I find the pressure increase was needed while some of the highest pressures were not feasible.

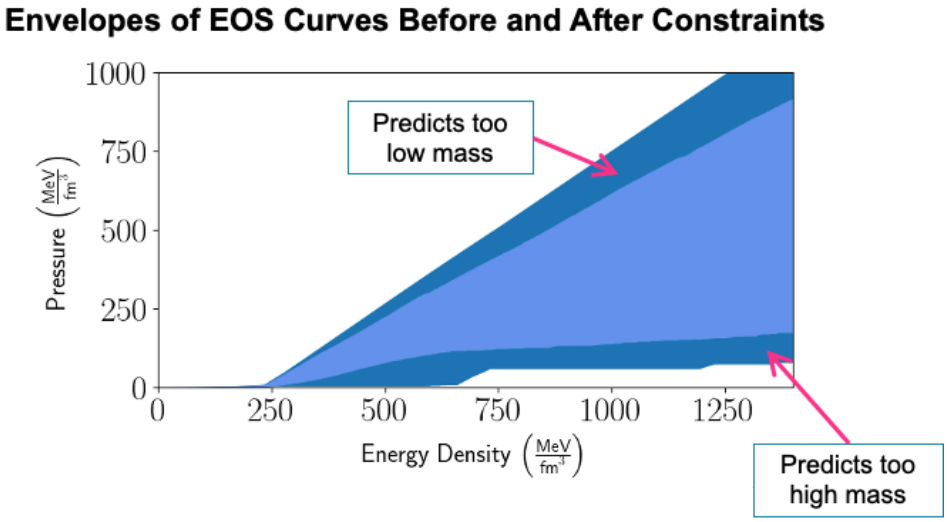


Figure 10: Envelopes of probable EOSs before and after constraints, in pressure vs. energy density space. The dark blue region represents the full set of EOSs considered, with every EOS lying within it. The light blue region shows the extent to which the EOSs were constrained, representing the 90th percentile range of EOSs after the constraints by the data. I emphasize that this plot shows the envelope of the EOS curves and does not imply that an arbitrary EOS going through the shaded region is allowed or even considered.

After constraining the mass-radius curves, finding the 90th percentile radius intervals of a 1.4 solar mass neutron star, and mapping the probabilities back to the corresponding EOSs, I separated the EOSs with and without phase transitions. In Table 3, I summarized the relative probabilities of first-order phase transitions (FOPT) to no phase transition calculated in each step of constraining EOSs. To find the relative probability, I calculate the likelihood of a phase transition by summing the probabilities of the EOSs with a phase transition. Then, I divide by the likelihood of no phase transition, which is calculated similarly, and multiply by a factor that accounts for the unequal number of EOSs with and without a PT.

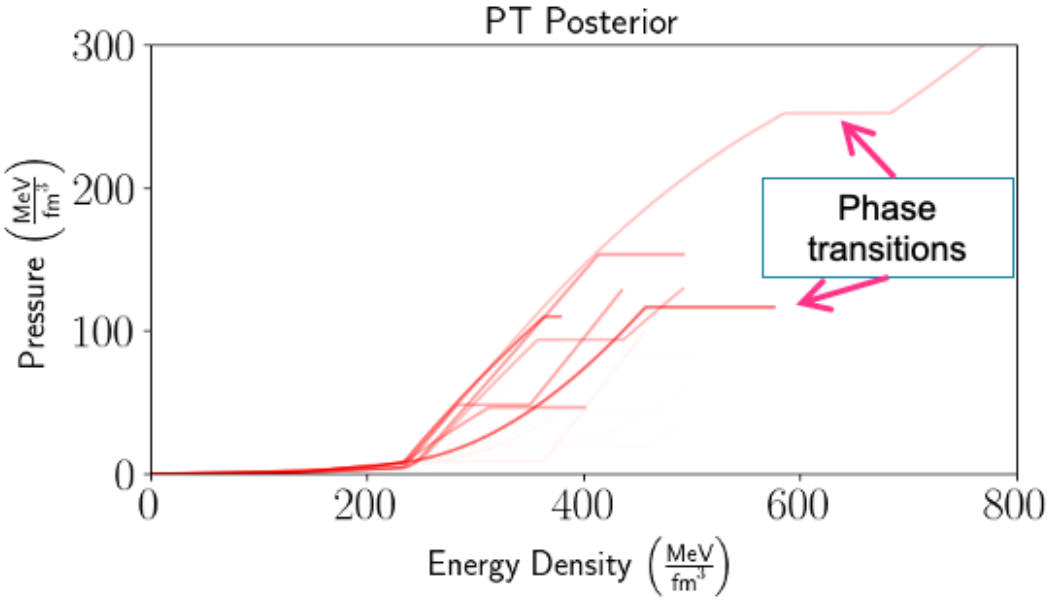
Most EOSs that have a PT were eliminated in the first step since those EOSs have a maximum possible mass above 2 solar masses. The EOSs without PTs are preferred by the remaining data sets as well. After the final step of constraining EOSs, the odds of a FOPT to no

PT is about 1 to 21. This is consistent with the findings from Fig. 9 that the EOSs with quark matter or other exotic matters do not predict the data as well as those with only nucleons.

	Initial set	Max mass	NICER 1	LIGO	NICER 2	NICER 3
Odds of FOPT to no PT	1:1	1:16.87	1:19.07	1:19.16	1:19.68	1:20.78

Table 3: Odds of a first-order phase transition to no phase transition in each step of constraintment.

In Figure 11, I plot the constrained EOSs in the two groups. Since I am investigating phase transitions in the cores of neutron stars, I truncated the EOS curves at their respective maximum central pressure, which varies from EOS to EOS.



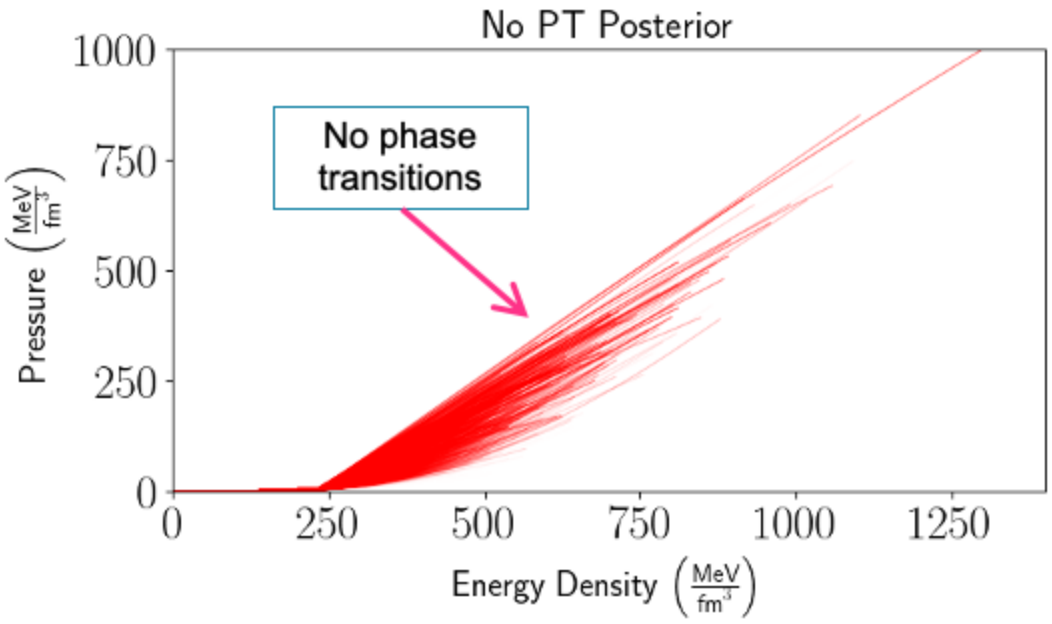


Figure 11: Likely EOSs in two groups. a) The EOSs that predict first-order phase transitions cc(PT) at the core; b) the EOSs that predict no first-order phase transitions in neutron stars. In both plots, curves are truncated at the maximum possible central pressure.

In Fig. 11 (a), I note that the phase transition starts and completes at different energy densities so I performed a statistical analysis on the parameters that characterize the phase transition as shown in Fig. 11 where epsilon is the energy density and sound speed squared (c^2) is the slope of pressure-energy density curve. We observe that the phase transition starts at an energy density (epsilon start) of about 276 MeV/fm^3 and that the change in energy density over the phase transition (epsilon difference) is about 218 MeV/fm^3. While the initiation energy density is not strongly correlated to the energy density jump, the latter and the sound speed after the phase transition indicate a positive correlation, meaning that the more energy needed for a phase transition, the larger the sound speed after the PT is.

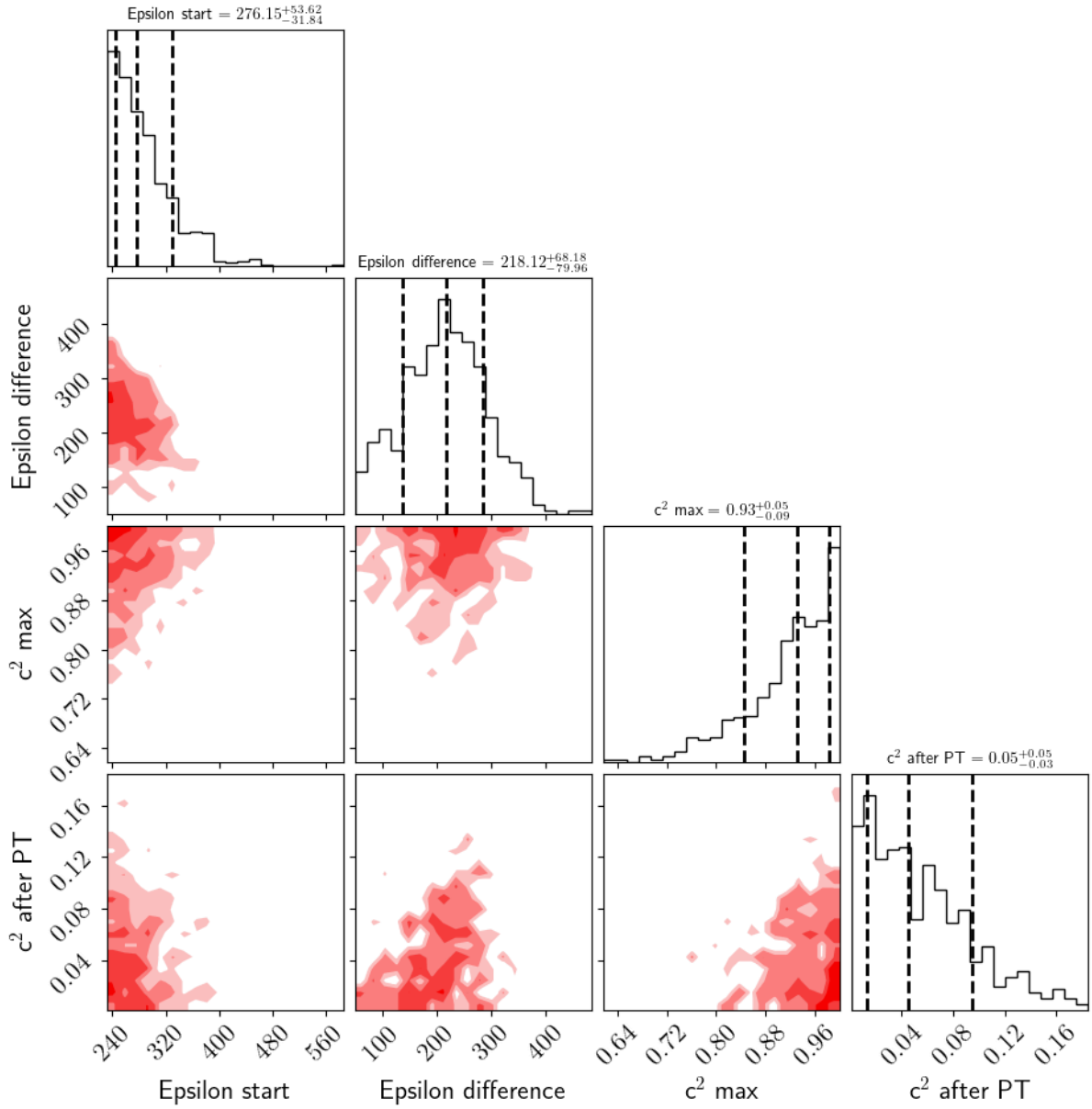


Figure 12: Pair plots for the phase transition parameters. For each parameter, we show the mean value and one standard deviation. "Epsilon start" is where the phase transition initiates in energy density and "Epsilon difference" is the jump in energy density occurring during the phase transition. "c^2 max" is the maximum sound speed squared for each EOS in the entire range of energy density and "c^2 after PT" is the value of the sound speed right after the phase transition.

VII. CONCLUSIONS

I successfully developed software that solves the TOV equations and analyzes observations of neutron stars in a Bayesian statistical framework to determine the probability of a given mass-radius curve. Among the five data sets used, the first three had been previously analyzed in a similar manner (Dietrich *et al*., 2020**)** and I verified that my results were consistent with theirs. I incorporated two new data sets, further constraining the neutron star EOS. In addition, I examined the allowed EOSs to investigate the possibility of a phase transition at the core. There are two main results in this paper.

1) First is a stronger constraint of the mass-radius curves using the new data from NICER (PSR J0437-4715, PSR J0740+6620). My result is that the 90th percentile interval for the radius of a 1.4 solar mass neutron star is $11.60^{+0.81}_{-0.67}$. After determining the mass-radius curves that are most probable given the data, I mapped those curves back to the pressure-energy density space to find the constrained EOS models of neutron star matter (Fig. 10). The dark blue region represents the range of EOSs considered before applying data constraints, while the light blue region shows the EOSs allowed by the data. My analysis eliminated many EOS curves. This can be also seen in Fig. 9 where the mass-radius prediction of the various theory-based models are plotted along with the constrained curves.
2) Second, I categorized the EOSs based on whether they had a first-order phase transition (FOPT) or not, which predicts deconfined quark matter or nucleonic matter, respectively, in the core. My data-oriented analysis shows that a FOPT is not likely in the core of the neutron stars. Which is consistent with the findings that EOSs that contain any exotic matter are unlikely, as seen in Fig 9. This contrasts with Annala et al. (2020), which claims that a phase transition is necessary. One thing to note is that I considered only first-order phase transitions since mixed phase transitions are not easily or definitively found from the EOS.

APPENDICES

Bimodal Reanalysis of NICER 1 (PSR J0030+0451)

Recently, the NICER team released the results of the reanalysis of PSR J0030+0451 using an updated calibration framework and simulation capabilities (Vinciguerra *et al.* 2024). The reanalysis contains two modes, meaning there are two likely sets of mass-radius values. These modes were obtained by using two different models for the hot spots (locations, morphology, etc.) on the neutron stars in their simulations. I use the methodology developed in this work to find which mode is more likely according to the other neutron star data. That is, I use the Max Mass, LIGO, NICER 2, and NICER 3 data to find the probabilities of the mass-radius curves as in the method and procedure sections. Then by comparing the constrained mass-radius curves to the reanalyzed mass-radius data of NICER 1, I found what mass and radius are probable for PSR J0030+0451. In Fig. 11, there are two sets of contours that correspond to the different modes. The first mode (blue contours) generally predicts lower mass and radius compared to the second mode (green contours). The constrained mass-radius curves fit the first mode of the analysis so I conclude that the first mode is more probable and a better model of neutron stars.

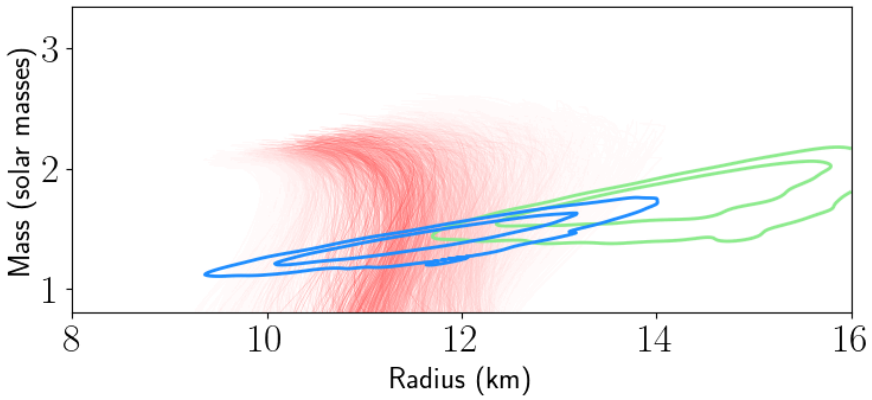


Fig. 11. Bimodal analysis of PSR J0030+0451 with constrained mass-radius curves. Mode 1 (blue contours) is found to be more probable since it fits the mass-radius curves better.

ACKNOWLEDGEMENTS

I thank Drs. Ingo Tews and Rahul Somasundaram for their mentorship on this project, both at Los Alamos National Laboratory. I also thank my parents, Dr. JeeYeon Plohr and Dr. Bradley Plohr, for guidance and support.

REFERENCES

Abbott *et al*. GW170817: Observation of gravitational waves from a binary neutron star inspiral, *Phys. Rev. Lett.* 2017, 119.

Akmal, A., and Pandharipande, V.R Spin-isospin structure and pion condensation in nucleon matter, Phys. Rev. C, 1997, 56, 226.

Annala et al. Evidence for quark-matter cores in massive neutron stars, *Nat. Phys*. 2020, 16.

Antoniadis, J. *et al.* A massive pulsar in a compact relativistic binary, *Science* 2013, 340.

Arzoumanian, Z. *et al.* The NANOGrav 11-year dataset: High-precision timing of 45 millisecond pulsars, *Astrophys. J. Suppl.* 2018, 235.

Benhar, O. *et al.* (eds.) Nuclear Theory in the Age of Multimessenger Astronomy. CRC Press, 2024.

Camenzind, M. Compact Objects in Astrophysics: White Dwarfs, Neutron Stars and Black Holes, Springer, 2007.

Capano, C. *et al*. Stringent constraints on neutron-star radii from multimessenger observations and nuclear theory, *Nature Astr*. 2020, 4.

Caplan, M.; Horowitz, C. Astromaterial Science and Nuclear Pasta, *Rev. Mod. Phys.* 89, 041002, 2017.

Choudhury, D. *et al.* A NICER View of the nearest and brightest millisecond pulsar: PSR J0437-4715, arXiv: 2407.06789, 2024.

Cowan, J. *et al.* Origin of the heaviest elements: The rapid neutron-capture process, *Rev. Mod. Phys.* 2021, 93.

Cromartie, H. T. *et al*. Relativistic Shapiro delay measurements of an extremely massive millisecond pulsar, *Nature Astron*. 019, 4.

Dietrich, T. *et a*l. Multimessenger constraints on the neutron-star equation of state and the Hubble constant, *Science* 2020, 370.

Epelbaum, E., *et al.* Modern Theory of Nuclear Forces, *Rev. Mod. Phys.* 2008, 81.

Gandolfi, S. *et al.* From the microscopic to the macroscopic world: from nucleons to neutron stars, *J. Phys. G: Nucl. Part. Phys.* 2019, 46.

Glendenning, N. K., & Moszkowski, S. A. Reconciliation of neutron star masses and binding of the Λ in hypernuclei, Phys. Rev. Lett., 1991, 67, 2414.

Glendenning, N. K., & Scha†ner-Bielich, J. First order kaon condensate, Phys. Rev. C., 1999, 60, 025803

Kobyakov, D. Application of superconducting-superfluid magnetohydrodynamics to nuclear "pasta" in neutron stars, *Phys. Rev. C*, 045803, 2023, 98.

Kurt, W. Bayesian statistics the fun way. No Starch Press, Inc., 2019.

Lattimer, J. M. and Prakash, M. Neutron Star Structure and the Equation of State, *Astrophys. J.* 2001, 550:426.

Miller, M. C. *et al.* PSR J0030+0451 Mass and radius from NICER data and implications for the properties of neutron star matter, *Astrophys. J. Lett*. 2019, 887.

Miller, M. C. *et al.* The Radius of PSR J0740+6620 from NICER and XMM-Newton Data, *Astrophys. J. Lett.* 2021, 918.

Müller, H. and Serot, B. Relativistic mean field theory and high density nuclear equation of state, Nuclear Physics A, 1996, 606.

Oppenheimer, J. R.; Volkoff, G. M. On massive neutron cores, *Phys. Rev*. 1939, 55.

Prakash, M., Cooke, J. R., & Lattimer, J. M. Quark-hadron phase transition in protoneutron star, Phys. Rev., D52, 1995, 661.

Raaijmakers, G. *et al.* A NICER view of PSR J0030+0451: Implications for the dense matter equation of state, *Astrophys. J. Lett.* 2019, 887.

Rezzolla, L. *et al.* Using gravitational-wave observations and quasi-universal relations to constrain the maximum mass of neutron stars, *Astrophys. J.* 2018, 852.

Riley, T. E. *et al.* A NICER View of the Massive Pulsar PSR J0740+6620 Informed by Radio Timing and XMM-Newton Spectroscopy, *Astrophys. J. Lett.* 2021, 918.

Somasundaram, R.; Tews, I.; Margueron, J. Investigating signatures of phase transitions in neutron-star cores, *Phys. Rev. C*, 2023, 107, 025801.

Tews, I. Private communication, 2024.

Tews, I. *et al*. Constraining the speed of sound inside neutron stars with chiral effective field theory interactions and observations, *Astrophys. J.* 2018, 860:149.

Tolman, R. C. Static Solutions of Einstein's Field Equations for Spheres of Fluid. *Phys. Rev.*1939, 55, 364.

Vinciguerra, S. *et al.* An updated mass-radius analysis of the 2017-2018 NICER dataset of PSR J0030+0451, *Astrophys. J.* 2024, 961.

Watts, A. *et al.* Measuring the neutron star equation of state using X-ray timing, *Rev. Mod. Phys.* 2016, 88(2):021001.

Wiringa, R. B., Fiks, V., & Fabrocine, A., Equation of state for dense Nucleon matter, Phys. Rev. C, 1988, 38, 1010.

Zhang, Z-W and Pethick, C. J. Proton superconductivity in pasta phases in neutron star crusts, *Phys. Rev. C* 2021, 103.

Point Cloud Surface Reconstruction

Andrew Morgan[1] (Sole Author), **Nathaniel Morgan**[2] (Project Mentor/Teacher)

[1]Los Alamos High School, Los Alamos, NM, USA

[2]Project Mentor and Teacher, Los Alamos, NM, USA

Abstract

The efficient and versatile reconstruction of the surface of point clouds remains a notable problem throughout computer science, physics, robotics, engineering, and other related fields. Many current methods struggle with noisy data, uneven density distribution, and discontinuities. This paper proposes a solution that uses a level-set-based method integrated with a linearized sparse octree, neighboring node caching, a min-heap binary tree, and surface tension simulation to parse large datasets to reconstruct point clouds as watertight meshes. A basic prototype in Python 3 validated the utility of this approach and provided a foundation on which to construct optimizations. Translations into C++ 17 and Rust implemented these additional concepts and demonstrated notable performance improvements over previous iterations.

1. Introduction

Point clouds are essential tools for countless fields and applications, including medicine, protein synthesis, robotics, computer graphics, video games, geology, lidar, 3D scans, and more. They provide a versatile and unique way to store many types of data and allow for novel algorithms. However, due to their unstructured nature, they have limited direct utility. Many applications require well-defined discrete surface topology, often as polygonal meshes, which point clouds cannot provide. Extracting the isosurface offers a solution and bridges this gap, extending its utility.

Many existing solutions utilize a wide range of methods and techniques. However, many of these have limitations, such as difficulty extracting the isosurface from noisy, incomplete, and

discontinuous data sets. Point clouds are often structureless and highly variable, making it challenging to form generalizations and find easy solutions. Additionally, specific applications require quick processing of point cloud data for real-time applications.

Point clouds can range from a few hundred points to sometimes over a billion, further complicating the matter. Creating a versatile algorithm to handle this extensive range of data and inconsistent and missing information proves challenging. Any solution has to balance performance and memory consumption, as these data sets can reach many gigabytes in size.

This paper proposes an optimized hybrid level-set-based method to help combat these problems. The method takes in an arbitrary point cloud and returns a watertight discrete mesh. Additionally, it is relatively versatile and can handle any number of points, from one to millions. Unless cited otherwise, all of the work and code created in this project was done by Andrew Morgan for the 2025 NM Supercomputing Challenge.

Section 2 discusses multiple approaches to this problem. Section 3 follows, breaking down the steps in the pipeline of the proposed solution. Then, Section 4 explores data structure optimizations, specifically a sparse, linearized, adaptive octree. Section 5 discusses the program's results and validates this approach's effectiveness. Section 6 summarizes the findings for the proposed method.

1.1 Background Information

A quick summary of some background information regarding a few topics mentioned may help in understanding some of the content of this paper.

1. A tree data structure is a structure that starts with a root node. Then, successively, each node starting at that root has a set number of children which may be filled, slowly branching outwards like how the branches of a tree start as one, and over time branch outwards with each branch having its own branches.

2. A perfect binary tree structure is a tree data structure where all branches get fully filled out and progress to the same depth. An imperfect tree would have branches that terminate early or don't have all their children. See [1] from *GeeksforGeeks* for further information.

3. Point clouds are an arbitrarily sized collection of unordered points; they often collectively represent a larger object or structure.

4. Hashes or hash codes refer to a unique index or value resulting from a given input. Some hashes are random, and others are structured. Hashes are fixed-size and often satisfy certain conditions that the former data couldn't. For example, converting a string into an unsigned integer using a hashing algorithm would allow the value to act as an index within an array.

5. Data structures refer to varying methods of storing memory as well as the associations one piece of memory has to another within the collection. Grids are single or multi-dimensional arrays representing a rectangular area in the form of evenly sized and spaced boxes. Two-dimensional grids are also called matrices.

7. Vectors are a continuous array with a dynamic size that never has holes in the middle (often called lists). Removing and adding items to the center or start does reduce performance, though; any values beyond it get shifted in memory to make room or fill in a void, resulting in a large amount of memory movement.

2. Related Work

Many solutions exist for surface reconstruction, with varying strengths and weaknesses. Some more traditional methods, like marching cubes, require a structured scalar field. However, the marching cubes algorithm is still practical as an intermediate step in a more extensive process; marching cubes standing alone is valuable in many other contexts involving more structured data. Other traditional algorithms, such as Delaunay triangulation, require structured data and can be very slow on large data sets. Requiring structured data presents a complication, as many

point clouds don't have an explicit structuring or order. However, this doesn't mean these methods don't have utility, as they're still widely used and play a key role in many areas.

Some newer approaches leverage artificial intelligence (AI) based methods, although they, too, have their strengths and weaknesses. High frequency, fine-tuned details, and more complex topology are intricate to capture with AIs. AIs often excel in a specific area, although they struggle in others. Additionally, AIs require extensive data sets, which, combined with the already significant size, complexity, and scale of point clouds, leads to high computational cost and time complexity. Furthermore, training an AI on complex surface topology proves challenging as there are countless variations and a lack of structure or unified patterns between or inside data sets. These factors limit the adaptability and generalization of AI implementations, making them fall short of the overarching goal of this project.

There are many other miscellaneous solutions, although this project focuses on level-set methods. Level-set-based methods rely on mathematically extracting the isosurface level through various means. Many implementations utilize signed distance fields (SDFs) to represent the point cloud. SDFs are often much more structured, even with an adaptive data structure (for example, an octree or kd-tree), allowing for more traditional methods, such as marching cubes or dual contouring, to be combined into a systematic pipeline. In other words, utilizing SDFs in conjunction with other techniques allows for hybrid methods, balancing accuracy, performance, memory consumption, and adaptability. This adaptability while maintaining reasonable performance makes a hybrid level-set-based method well-suited for the project's goal.

3. Signed Distance Field Representation

While point clouds may be highly variable, the signed distance to the nearest point at any given position has much more structure. A primitive way to choose which points to sample the signed distance is to create a 3D array with known bounds and positioning. This primitive solution is the exact approach taken for the prototype in Python 3. However, it has inherent flaws.

Because arrays are a fixed size and spacing, areas of low detail (i.e., very few or no points) require the same amount of memory allocation as an area with lots of detail. Additionally,

when computing the signed distance, low-detail regions will receive the same computation time and resources as those of high detail. Additionally, areas of low detail, which don't need a lot of expensive computation or significant memory allocation, receive a large portion of the available resources. The over-allocation of resources in low detail areas also takes away critical computation and memory necessary to evaluate complex topology regions accurately.

However, using an adaptive octree data structure can fix this issue. While octrees are far more complex than traditional grids, the implementation mentioned in this paper adaptively subdivides the structure in areas of high and complex detail while giving sparse areas more limited representation. This data structure, for one, saves a lot of memory. In a simple test case, it consumed nearly 29,000% less memory when storing just the signed distances (from 8MB down to 30KB for a basic grid of 64-bit floats, not including additional information on the actual structure). The benefit of the octree is further compounded because there are fewer nodes or points at which to sample the signed distance, and less computation is needed overall. This reduction in computation and memory allows for increased resources in more complex and intricate point cloud sections, resulting in greater detail and precision. More depth on this octree implementation and other data structures are in Section 4.

There is one issue with this current method. A known surface contour is necessary to create a signed distance field (SDF) instead of a regular distance field. Constructing an unsigned distance field from the point cloud instead of an SDF alleviates this problem. After this, an algorithm determines which sections are solid and which are hollow. A shell around the surface is then created by generating the exterior edges of the part(s). Because this shell is solid, the known surface contour allows for calculating a proper SDF. This pipeline process is broken down further in Section 3.1.

3.1 Signed Distance Pipeline

One inherent issue in generating a signed distance field, as discussed in Section 3, is that a known surface contour is necessary to get the signed part of an SDF. The solution is to break the

process into four steps: calculating an unsigned distance field, signs, solid edges, and finally, computing the complete SDF.

The initial step of creating an unsigned distance field is relatively trivial. The process involves looping over every node or grid cell in a given data structure and performing a nearest neighbor search on the point cloud (calculating the minimum distance to the nearest point). However, some complexity arises when optimizing and executing the search on an octree. Section 4.1 details the implementation of the nearest neighbor search on an octree.

A more straightforward optimized solution for a fixed grid is a chunking system, also referred to as hashing. The process relies on grouping all the points into unique vectors or arrays based on their local position. A good example is how the game Minecraft divides the world into 16x16 chunks. These chunks allow for a smaller, localized search to expand as needed to find the nearest point. Creating a smaller search radius improves performance by looking over fewer points in any given search. Implementing this solution in the prototype script in Python 3 gave decent performance gains, considering the reduced complexity compared to other algorithms.

Step two calculates the signs for the unsigned distance field using a novel algorithm. The algorithm calculates every grid cell by repeating the following set of 4 steps. (1) The initial step is to loop over all 1D slices facing a single axis and step through each cell one by one. (2) At each marched step through a given slice, check the unsigned distance; if the distance is less than the isosurface level, continue stepping along until the distance is greater than or equal to the isosurface level. (3) If the grid cell in the corresponding array for storing signs contains a filled point, save the current tracking sign as that sign and continue along; otherwise, flip the tracking sign and fill the entire region between the boundaries created by the isosurface and unsigned distance field with that sign. (4) Repeat these steps until every slice finishes its calculation. Like previous algorithms, octrees cause complications and require modifications to the underlying algorithm; Section 4.1 goes into these necessary modifications.

Step three involves calculating a shell around any object's exterior edges (in other words, voxelizing the distance field of the point cloud). Similar to the first step, the process is relatively simple. The primary step is to go through every grid cell or node and check a few conditions: if a

hollow point is directly adjacent to the cell or node (diagonals don't count) and the current position is solid, add a new surface point.

The final step builds upon the previous step to generate the final SDF. Similar to the first step, start by going through every point and calculating the unsigned distance. However, this time, use the surface shell rather than the point cloud to calculate the unsigned distance. After getting the distance, check the sign at the given node or grid cell position; if the sign indicates it's solid or the original unsigned distance is less than the isosurface level, flip the sign of the current distance. This final step concludes the calculation of a proper SDF, allowing a continuation in the larger pipeline.

3.2 Surface Tension Simulation

Due to the nature of the SDF generation, natural surface ungulations occur in the reconstructed part. However, a scalar field surface tension simulation solves this problem by smoothing higher frequency bumps on the surface; this method also preserves a lot of lower frequency bumps, although it won't work as well on every application. [2] breaks down the math behind the surface tension method. The surface tension simulation works by finding the curvature of the surface and raising the troughs while dropping the peaks. A summary of the math from [2] is as follows:

The first Equation (1) solves for the level set field while applying a front velocity of F. ϕ represents the level set field. i, j, k represent the position, and they can also represent the index within the grid. t represents time.

$$\frac{\phi_{i,j,k}^{n+1} - \phi_{i,j,k}^{n}}{\Delta t} = max(F, 0)\nabla_{i,j,k}^{+} + min(F, 0)\nabla_{i,j,k}^{-} \quad (1)$$

Where:

$$\nabla_{i,j,k}^{+} = [max(D^{-x}\phi, 0)^{2} + min(D^{+x}\phi, 0)^{2} + \ldots$$

$$\dots max(D^{-y}\phi, 0)^2 + min(D^{+y}\phi, 0)^2 + \dots$$

$$\dots max(D^{-z}\phi, 0)^2 + min(D^{+z}\phi, 0)^2]^{\frac{1}{2}}$$

$$\nabla^{-}_{i,j,k} = [min(D^{-x}\phi, 0)^2 + max(D^{+x}\phi, 0)^2 + \dots$$

$$\dots min(D^{-y}\phi, 0)^2 + max(D^{+y}\phi, 0)^2 + \dots$$

$$\dots min(D^{-z}\phi, 0)^2 + max(D^{+z}\phi, 0)^2]^{\frac{1}{2}}$$

D^{-x} refers to the backwards finite difference operation in the x direction. D^{x} refers to the forwards finite difference operation in the x direction. This applies to all three dimensions – x, y, and z. V is a constant representing a constant velocity inwards or outwards; it can either keep the object's size or shrink or expand the object depending on its value. F is defined as the front velocity and equals:

$$F = V - \kappa$$

V is a constant that moves the level set field in the normal direction. Kappa is the curvature of the front:

$$\kappa = \nabla \bullet \frac{\nabla\phi}{|\nabla\phi|}$$

The second term equates to the surface normal. Multiple iterations each solve these equations and adjust the scalar field, smoothing it over time. The more time steps (while shrinking the time duration), the better the results. Some parameters can lead to instability and undesired results if not correctly set.

3.3 Isosurface Extraction

Signed distance fields are the first step to reconstructing the surface of a point cloud. However, an intermediary step is necessary to provide a more discrete representation. Some traditional reconstruction algorithms, like marching cubes and dual contouring, become useful here. The previous year's submission, which implemented marching cubes, acted as an initial solution for the Python 3 prototype and basic C++ implementation.

However, complications arise when applying marching cubes to a more dynamic structure, like an octree. While many solutions exist, most create intersecting geometry, which can lead to inconsistencies in physics simulations and other applications, or have non-watertight gaps. A method that solves this, proposed in [3], not only creates watertight meshes but also does so efficiently and without modifying the underlying octree; in other words, the octree is unconstrained, allowing for optimizations tailored directly to point clouds. The method relies on constructing a set of edge trees and using them to properly align geometry to intersecting node boundaries when using a hybrid-dual contouring approach. Section 4.1 dives into the implementation a bit deeper.

3.4 Hybrid Reconstruction Pipeline

Combining these steps — unsigned distance field calculation (Section 3), calculating discontinuities and signs (Section 3.1), surface tension simulation (Section 3.2), and isosurface extraction (Section 3.3) — creates an efficient pipeline from a point cloud to a discrete polygonal mesh representing its approximate surface.

Each step serves a purpose in the greater pipeline. The first step is gathering a more useful and structured representation of the point cloud — this initial step contains multiple steps, which Section 3.1 breaks down. From there, the surface tension simulation can smooth any artifacts and unnatural surface undulations. Finally, extracting a level set of that final scalar field produces a discrete and water-tight polygonal mesh; this step produces an STL file, allowing seamless integration of the mesh into most commercial software along with many algorithms.

4. Octree Data Structures

While this pipeline can create great results, it also consumes excessive resources from unnecessary computation and a massive memory footprint. A solution to this overconsumption of resources, albeit far more complex than a fixed 3D array, is to use an octree data structure instead. By dynamically subdividing the octree's node structure — essentially just adding more children to create a deeper tree — in areas of high complexity and detail, the octree can represent sparse sections with limited memory and computation while also redirecting those resources to regions of higher topological complexity.

Like other tree-data structures, octrees have a root node but, from there, have exactly eight children, each of which are nodes capable of creating more children nodes to branch the tree further out. Each node represents a cube or rectangle, and each subdivision divides the parent bounding box into eight equally sized boxes. The leaf nodes, which represent the deepest nodes that have no children of their own, store a vector of indexes referencing which points in a static array, representing the point cloud, fit within their bounding box. One way to do this is to keep a constant address or pointer referencing the original array. Alternatively, when needed, provide a parameter for the point cloud in the methods of the octree structure in languages like Rust. By passing the point cloud in as a parameter, it keeps its ownership within its original scope, preventing ownership and borrowing errors.

To dynamically subdivide the octree, follow a set of 3 rules. (1) If the current depth has reached the maximum specified depth for the octree, push all point indexes within the bounding box into an array or vector for the current node, and then stop subdividing as it's now a leaf node. (2) If there is either one or no points within the current node's bounding box, turn the node into a leaf node; in other words, stop subdividing the particular node. (3) If there are multiple points within the node's bounding box, which means the previous two rules weren't satisfied, subdivide the current node into eight children nodes and continue the rule-set for each of those individual nodes.

Just these steps alone can provide significant performance gains. However, there are other possible improvements. These other optimizations rely on linearizing the data structure. Linearizing an octree involves representing all the data in a single, contiguous array. This paper's linearization implementation involves utilizing an array where each element contains another array of eight integers. Each of those eight integers represents an index to that exact same array to act as a pointer to the node's children. In the case of a leaf node, the eight indexes can either be replaced by null or by a value representing null; in the context of Rust, *Some(index)* represents standard indexes, while leaf nodes contain 8 *None's*. This structuring provides an efficient way to follow the tree's many branches and determine whether a node has children or is a leaf node. To store points in a leaf node, another array aligning with the original contains vectors to store references to points in the point cloud; an array with a predetermined size also works for representing the vector in memory. Utilizing that vector, adding a point is as easy as accessing the array at the index of the leaf node and pushing the points' indexes in the point cloud to the vector.

This linearized design has a few notable advantages. The first benefit is that the octree's data lines up contiguously in memory, allowing for more cache hits and quicker data fetches; cached information is also naturally aligned sequentially in increasing order, allowing for more efficient lookup algorithms, such as binary searches (also known as a bisect search).

```
pub fn BinarySearch <T: Eq> (points: &Vec <T>, searchValue: &T) ->
    Option <T> {
  let mut currentIndex: usize = 0;
  let mut dividedSize = points.len();

  let mut halfWidth: usize;
  // this could also be a loop; however, this prevents runaway code
  for _ in 0..MAX_BINARY_SEARCH_ITERATIONS {
    halfWidth = dividedSize / 2;
    dividedSize -= halfWidth;  // splitting the bounding size
    if let Some(middleValue) = points.get(currentIndex + halfWidth) {
      if middleValue == searchValue {
         return Some(currentIndex + halfWidth);
      } if middleValue < searchValue {
        // splitting the search
```

```
        currentIndex += halfWidth;
      }
    }
  } None
}  // Rust
```

Mortan codes (Z-order curves) [4] are another useful and performant hashing technique involving bit manipulation to create unique codes for any position that maintain spatial locality (two neighboring points will have similar hash codes). Hash maps are sometimes useful due to some variability in the codes' values. The algorithm works by taking three 32-bit numbers representing the three axes. The bits interweave, so every three bits contain a bit from the x, y, and z coordinates, creating the pattern: $x_1y_1z_1x_2y_2z_2\ldots$. This results in a 96-bit unique hash, although Rust has 64-bit and 128-bit integers, so a larger number than the code is necessary. Because of the implementation within the octree, the code gets represented as an unsigned integer (this makes the final type a 128-bit unsigned integer or, in Rust, u128).

```
pub fn GetMortonCode (&self, xi: u32, yi: u32, zi: u32) -> u128 {
    let mut x = xi as u128;
    x = (x | (x << 16)) & 0x030000FF;  // magic nums -> stackoverflow
    x = (x | (x <<  8)) & 0x0300F00F;  // spacing the x bits out
    x = (x | (x <<  4)) & 0x030C30C3;  // creates room for y, and z
    x = (x | (x <<  2)) & 0x09249249;
    //... the same as above for y and z
    x | (y << 1) as u128 | (z << 2) as u128  // interweaving all bits
}  // Rust
```

Another performance gain occurs as accessing an array at an index is faster than chasing repeated pointers to other instances of nodes. This second point compounds with the previous ones and also alleviates the issue of slow neighboring node computation times; neighboring nodes are expensive to find and scale alongside the number of points and depth of the octree. This expensive calculation becomes problematic because the nearest neighbor search traverses the octree from node to node to identify nearby points while minimizing the search radius, and many neighboring nodes are needed to do this. Further compounding that issue, the nearest neighbor search must run for every node multiple times. However, because each node in the linearized structure is represented by a single integer index/identifier, another array that aligns

with every node's index can store a vector containing the indexes to every neighboring node for that given node. While this caching system requires an expensive computation for every node to find all its neighbors upfront, it prevents the necessity of doing these computations many times during each search. In fact, with this caching method, traversing the octree is possible in constant time, regardless of the number of points or the octree's depth (the search algorithm runs in log n time due to incorporating additional algorithms beyond traversal). Implementing a similar caching system with a nonlinear structure would be incredibly difficult; any solution would still involve an expensive descent from pointer to pointer every single time a cache lookup happens. While challenging to implement, these two major optimizations provide significant performance gains, making them worthwhile. The specific implementation of the nearest neighbor query on an octree is very complex compared to chunk or grid-based methods. Section 4.2 details the exact implementation used in this paper.

4.1 Octree Pipeline Integration

While the octree provides notable improvements, other algorithms within the greater pipeline are inherently unable to handle the variability. The sign generation algorithm falls short here because it traverses line by line, row by row; however, octrees don't have a perfectly aligned structure because of their adaptive structure. One solution is using a system where all nodes bounding the edges of the outline get pushed to a vector, and that vector acts as the starting point for further iterations; every iteration, the sign gets flipped, beginning at a point on the outside with a value of one representing hollow space.

The second issue falls within the surface tension algorithm. Octree-based scalar-field surface tension simulations are much more complex than their grid-based counterpart. The solutions go beyond the scope of this paper. Because of the complexity and scope, the surface tension simulation was discluded for the octree-optimized pipeline despite the notable improvements on lower resolution point clouds.

The final problem arises when reconstructing the surface of the scalar field. With a grid-based solution, marching cubes provided excellent results. However, Marching Cubes

doesn't translate as well to an octree. A solution proposed by [3] uses a hybrid method stemming from dual contouring. The approach proposed in that paper took in an unconstrained octree and returned a water-tight mesh. Again, this solution goes a bit beyond the scope of this paper. The paper cited below [3] provides an excellent breakdown, though.

4.2 Nearest Neighbor Query

One of the more complex aspects of integrating the octree is the nearest neighbor search algorithm (related to voronoi cells). [5] proposes an elegant solution and inspired some optimizations. For this paper's implementation, an expanding search radius provides the fastest results by pruning unnecessary data. However, finding the neighboring nodes to any given leaf node proves challenging. Additionally, searching for neighbors is too costly. A solution is to use a neighbor caching system; an array the length of the number of leaf nodes stores vectors containing the integer indexes of all neighboring nodes (discussed in Section 4).

From there, the solution is relatively trivial. Get all the neighbors starting at the node nearest to the sample position. Add all those neighbors to a priority queue (4.3) based on their distance to the query point. Then, for every iteration, pop the root node from the queue and continue. For every point encountered, keep track of the shortest distance. Finally, once the queue is empty or the shortest distance to the nearest node falls beyond the minimum distance found, return that minimum distance.

Locating the nearest leaf node to a given position is another challenge, though not nearly as complex. The first step is to force the query point into the bounding box of the octree using min and max. From there, start at the root node and iterate the number of times as the octree is deep. While iterating, keep track of the current node. The size of all nodes at a given depth is stored in a pre-computed array ($1.0 / 2.0^{depth}$). Take the query point and find its distance from the node's base. From there, divide that difference's x, y, and z components by half the node's size. Take the result and cast it to an integer of 0 or 1. Each node's eight children have different offsets from the node's base, which get stored in a constant order; using that known order, the current set of three integers allows for a reverse lookup of offsets to get the child's index within the current

node. Repeat that until encountering either a leaf node or the maximum depth (which would also be a leaf node). Some algorithms may require tracking the path to the node for traversal up the tree; usually, this only requires the last calculated leaf node, resulting in a negligible memory size.

4.3 Min-Heap Binary Trees

Min-heap binary trees, also referred to as priority queues, are binary trees where the root node always contains the smallest value. The counterpart would be a max heap binary tree; however, in the context of this paper, it isn't needed. Binary trees are similar to octrees. However, each node only has two children. *GeeksforGeeks* [6] provides a great article that breaks down binary heaps and inspired the implementation used in this paper.

The C++ standard library has a priority queue implementation that has min and max heap variants (the following is a min-heap binary tree, dictated by std::greater):

```
std::priority_queue<double, std::vector<double>,
    std::greater<double> > nodeQueue;
```

Some other languages' standard libraries may include an implementation with varying performance and versatility. For a manual implementation, the following dictates a min-heap binary tree; max-heap trees would be the same except search for the maximum instead of the minimum value. Insertion, popping, and swapping are the most important methods for this binary tree.

Swapping. The backbone of the other two algorithms relies on swapping values to satisfy the tree's rules: no value should be below a value greater than itself. When inspecting a node, look at the first branch; if the value is less than the current one, swap their values (ideally without altering the underlying data structure to reduce memory movement). Otherwise, check the right branch and do the same. If the condition fails for both branches, the value is in the correct position. Usually, this method gets called until the condition fails or the value reaches the bottom of the tree. This swapping method can also begin at the bottom of the tree and swap upwards to meet the condition until the parent is equal to or smaller than the current value.

Insertion. The first step requires constructing a new node with the given value. This new node becomes a child for one of the leaf nodes. After that, the swapping method iteratively places the value into the proper position.

```
pub fn Push (&mut self, value: (f64, usize)) {  ...  }  // Rust
```

Popping. The first step involves popping the root node and returning its value (and possibly co-value or index so it can reference additional information) – the root is always the smallest value. However, the binary tree requires a root node to function. The solution involves first removing one of the leaf nodes. From there, the leaf node replaces the root node. Then, the swapping method satisfies the conditions by swapping the root node downwards.

```
pub fn Pop (&mut self) -> Option <(f64, usize)> {  ...  }  // Rust
```

One way to optimize the queue is to attempt to balance the tree in the form of a perfect binary tree (1.1). Because the tree repeatedly gets restructured, there isn't always a perfect solution. The approximate solution explored in this paper relies on tracking the children of all leaf nodes. Two buffers are necessary to do this: one for future children below the maximum depth reached and one for the next deepest layer. The first buffer gets used when appending a new node; the final index of the first buffer represents the child, which receives the new value. After placing the value, the swapping method moves the value into position to satisfy the conditions. By having the two buffers, the tree will initially fill voids within the maximum depth of the tree before expanding the tree's depth. The second buffer dumps its contents into the first when it becomes empty. Most of the complexity comes from maintaining both these buffers as the tree mutates.

5. Validation

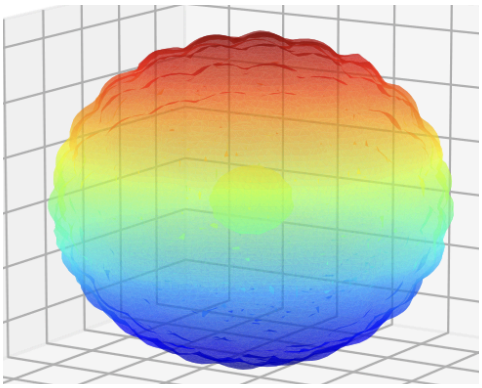


Figure 1. Hollow Fib. Sphere

A few different scenarios validated the effectiveness of the approach and implementation. The first method used mathematical equations to generate a point cloud around a known shape. After that, the reconstruction pipeline took in the point cloud, and the results reasonably matched the inputted shape, as seen in Fig. 1. A

known geometric model further tested the pipeline—an obj file from the Stanford Bunny (Fig. 2) allowed for a known comparison while also containing points that, on their own, have no spatial connection to each other. Similar to the first test, the results aligned reasonably well with the test model (Fig. 3). The specific Stanford Bunny model used had multiple holes in the surface geometry. However, the pipeline properly filled those holes while maintaining a reasonable level of accuracy in the overall model. This specific test didn't use surface tension smoothing.

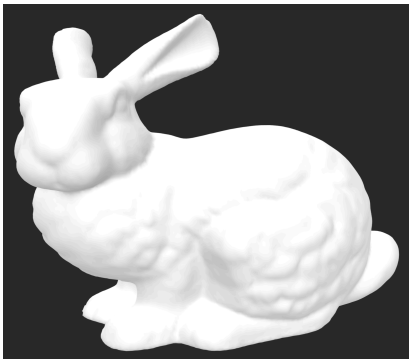


Figure 2. Stanford Bunny

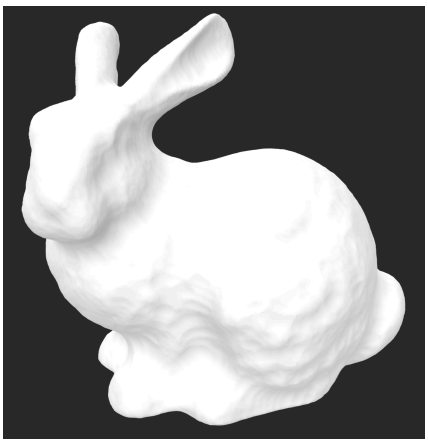


Figure 3. Reconstructed Bunny

The adaptive subdivision of the octree required rendering all corner points for every node to verify the structure and subdivision method. The corner points aligned with expectations, forming smaller nodes in regions of denser data while minimizing nodes in lighter areas (Fig. 4).

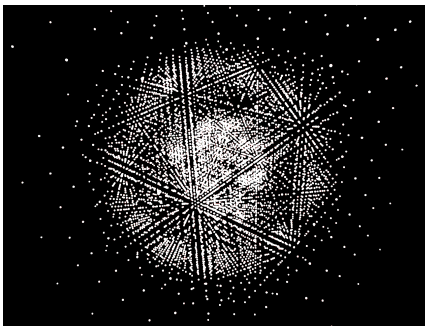
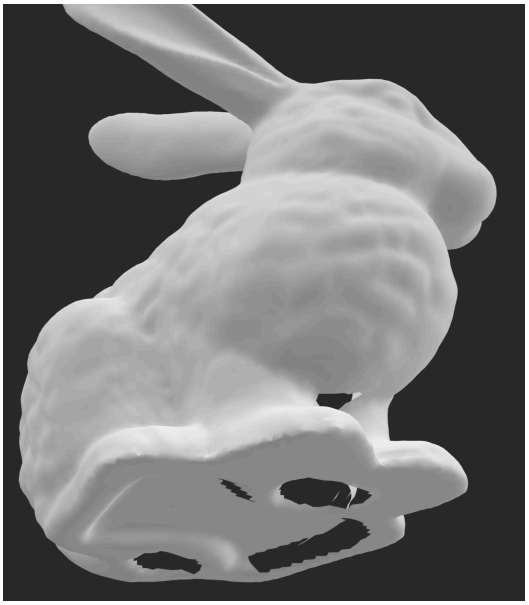
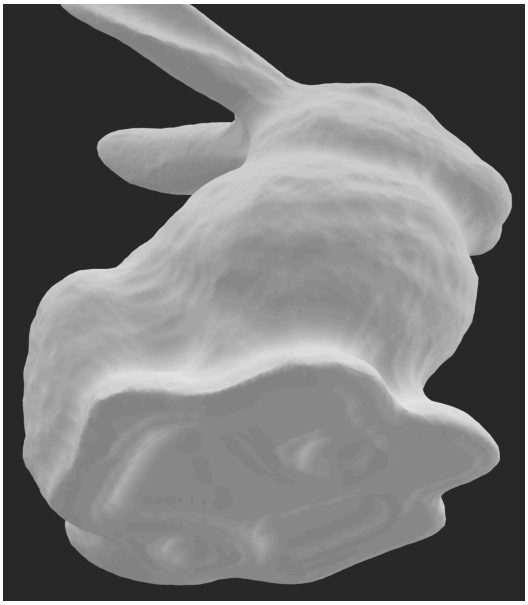


Figure 4. Hollow Sphere[1] Octree

The following are examples of the holes before and after reconstruction:





5.1 Results

All of the code and assets within this project are available on GitHub. The link is in Section 6.1. The point cloud data files (the program can load .pcd files, which are similar to .ply files that contain a slightly different header) used for reconstruction came from [7]. It provided a number of detailed point clouds, which were invaluable and allowed for excellent results.

Fig. 5 and Fig. 6 show the reconstruction of the Stanford Dragon. The reconstruction shows a decent improvement after running the surface tension simulation (Fig. 6). Additionally, the reconstructed model shows a decent handling of sharp corners (Fig. 7) and finer details (Fig. 8) while also keeping small gaps separated correctly (Fig. 9). Note that the mesh is water-tight.

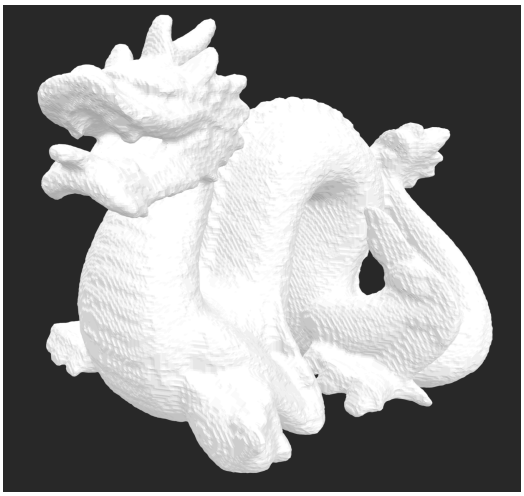


Figure 5. Unsmoothed Dragon

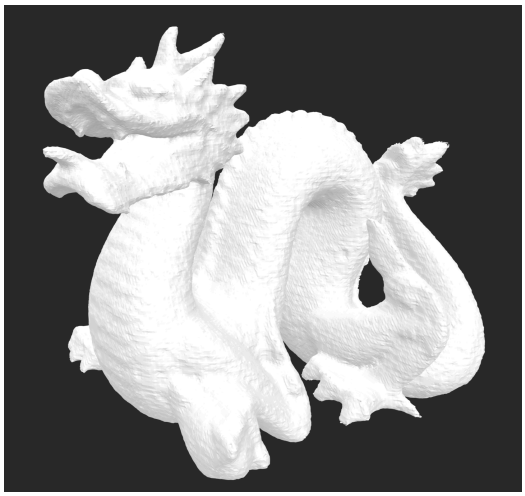


Figure 6. Semi-Smoothed Dragon

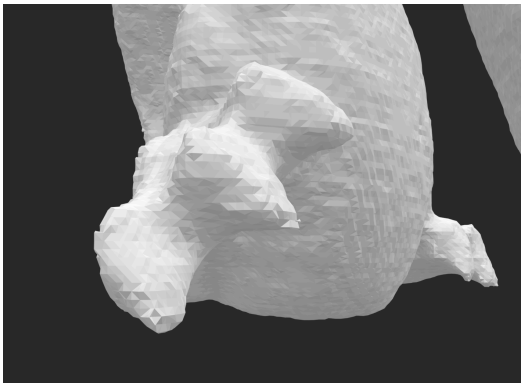


Figure 7. Sharp Claws

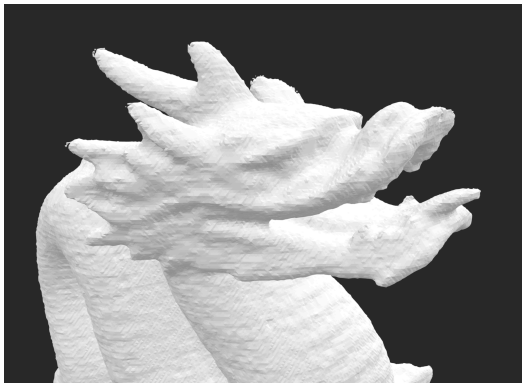


Figure 8. Finer Face Features

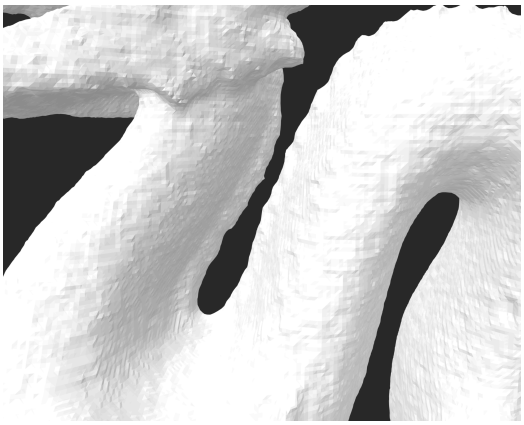


Figure 9. Narrow Gaps

The second result comes from the Egyptian mask [7]. The model contains a thin mask without a bottom. Additionally, the point cloud is somewhat noisy, complicating the reconstruction. Fig. 10 shows the reconstructed mesh, which handled the noisy data and gaps in the model well, only containing a few minor artifacts. Fig. 11 shows the bottom of the mask where the opening is along with the thinness of the mask (the mask should be fairly thin).

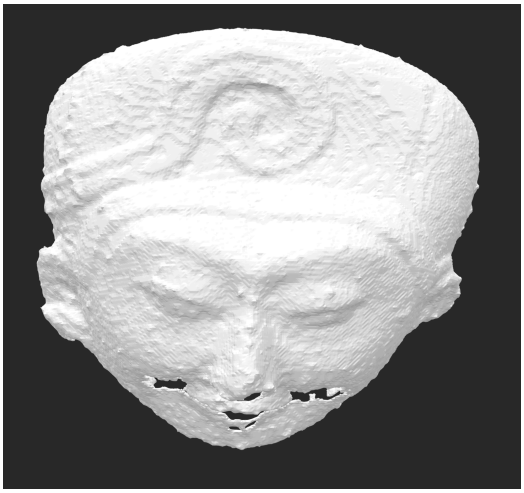


Figure 10. Egyptian Mask Front

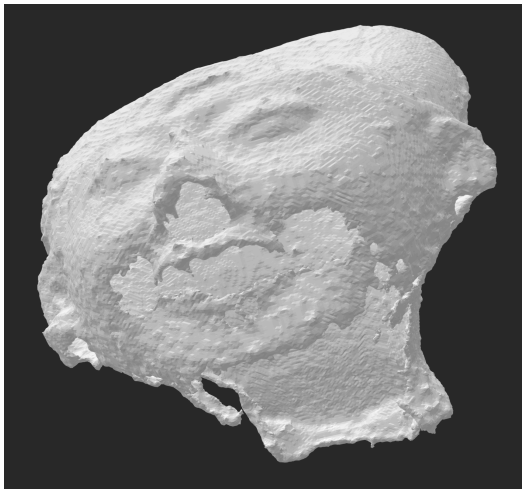
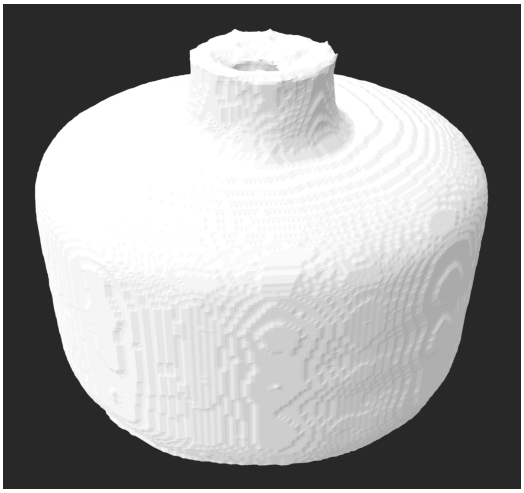
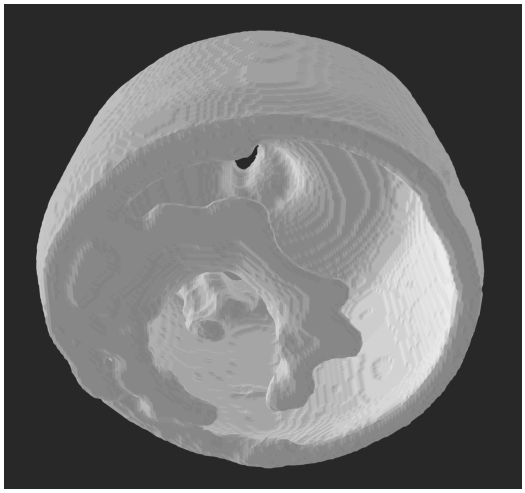


Figure 11. Bottom of the Mask

The final point cloud is a 3D scanned vase [7]. Similar to the mask, the vase has thin walls and a hollow interior (the Stanford Bunny and Dragon both had solid interiors). Additionally, the vase has a couple of sharper corners, presenting a challenge that the algorithm handled well.





5.2 Versitility and Limitations

The implementation covered has many strengths but also many weaknesses. Starting with the strengths, the algorithm can handle anywhere from a single point to millions – some other algorithms can't handle such significant disparities in data sets. Additionally, the algorithm can perform well on noisy or incomplete data sets (as seen with the Stanford Bunny in Section 5). In addition, while computational parallelization hasn't been implemented yet, many steps within the pipeline are well suited to run in parallel, which would provide significant performance improvements over the current results.

However, there are some limitations facing the current iteration of this project. The first is that the many steps in the pipeline each require allocated memory, and when combined, they result in a larger memory footprint than ideal; according to the activity monitor, mac's version of task manager, the program was using upwards of 1.25GB of ram while running on a dataset of a quarter million points when not using an octree. Additionally, thin objects can get slightly thickened due to the SDF generation. Furthermore, because of the SDF generation, surface tension simulation, and Marching Cubes in the non-octree pipeline, sharp corners and higher-frequency data can sometimes get smoothed over if the resolution becomes too small relative to the data. Also, the pipeline as a whole isn't running quite as efficiently as desired, and as such can't be used in real-time applications – although, with additional improvements, there is potential for significant performance gains over the current iteration. Reconstruction speeds varied from 30 seconds to 45 minutes (from around 40,000 points to upwards of 500,000 on a high-resolution grid), depending on the version, optimizations, and parameters like grid size or the number of points in the point cloud. Note that all benchmarks were taken on an older Mac M1, so a more performant computer would likely provide better benchmarks.

6. Summary

A hybrid level-set method provides a dynamic and versatile means to reconstruct the surface of arbitrary point clouds. Combinations of other algorithms expand that versatility and also offer greater performance while reducing the memory footprint. The first step of approximately voxelizing the point cloud provides a decent base to work from – some applications may only want the voxelized data and nothing beyond. From there, the generation of a proper SDF allows for a smoother and more accurate representation of the object. Additionally, that SDF is compatible with the surface tension simulation, Marching Cubes, and other algorithms.

A surface tension simulation is one such algorithm, providing a smooth output that removes artifacts and surface ungulations formed from the nature of SDFs. Additionally, a seamless integration of the simulation into the greater pipeline is relatively trivial.

Sparse, linearized octrees prove promising, providing excellent performance and memory with the only tradeoff being the complexity. Caching neighboring cells utilizing the benefits of the linearization further improves the nearest neighbor search, which has to run many times. Min-heap binary trees can further optimize the search query, efficiently generating signed and unsigned distance fields.

6.1 Final Remarks

I would like to thank Nathaniel Morgan for helping me come up with the initial idea for this project. I would also like to thank him for helping me interpret some of the math involved in the surface tension simulation.

In addition, I would like to thank a group at LANL for allowing me to present my project and for providing feedback on everything. Their library, MATAR, was also very helpful by providing a memory-safe multi-dimensional array structure for C++ (in place of alternatives like std::shared_ptr, std::unique_ptr, or unsafe raw pointers).

All of the code and assets used within this project are on GitHub: https://github.com/AndrewDMorgan/Point-Cloud-Surface-Reconstruction. The code is 54%

C++, 29% Rust, and 17% Python 3. In total, all three program versions came out to a total of a little over 4,800 lines. Through development, prototyping, iteration, and revision, over 8,000 lines were written, although much of that got refined and compressed over time.

References

[1] "Types of Binary Tree," *GeeksforGeeks*. [Online]. Available: https://www.geeksforgeeks.org/types-of-binary-tree/. [Accessed: Mar. 30, 2025]

[2] S. Osher; R. and Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*, Springer-Verlag, 2003. ISBM 978-0-387-95482-0

[3] M. Kazhdan, A. Klein, K. Dalal, and H. Hoppe, "Unconstrained isosurface extraction on arbitrary octrees," in *Proc. Eurographics Symp. Geometry Processing*, 2007

[4] "Z-order curve," *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Z-order_curve. [Accessed: Mar. 30, 2025]

[5] B. H. Drost, S. Lilc, "Almost constant-time 3D nearest-neighbor loopup using implicit octrees," 2018 *Machine Vision and Applications*. [Online]. Available: https://doi.org/10.1007/s00138-017-0889-4. [Accessed: Mar. 30, 2025]

[6] "Priority Queue using Binary Heap," *GeeksforGeeks*. [Online]. Available: https://www.geeksforgeeks.org/priority-queue-using-binary-heap/. [Accessed: Mar. 28, 2025]

[7] "RG-PCD: Reconstructed Geomatry Point Cloud Dataset," [Online]. Available: https://www.epfl.ch/labs/mmspg/downloads/reconstructed-point-clouds-results/. [Accessed: Mar. 30, 2025]

Future Reading

E. Alexiou, M. Bernardo, L. S. Cruz, L. G. Dmitrovic, R. Duarte, E. Dumic, T. Ebrahimi, D. Matkovic, M. Pereira, A. Pinheiro and A. Skodras, "Point Cloud Subjective Evaluation Methodology based on 2D Rendering," 2018 *Tenth International Conference on Quality of Multimedia Experiance (QoMEX)*, Cagliari, 2018, pp. 1-6. Doi: 10.1109/QoMEX.2018.8463406

B. H. Drost, "Almost constant-time 3D nearest-neighbor loopup using implicit octrees," *Springer Nature Link*. [Online]. Available: https://link.springer.com/article/10.1007/s00138-017-0889-4. [Accessed: Feb. 14, 2025]

S. Lague, "Coding Adventure: Marching Cubes," *Youtube*. [Online]. Available: https://www.youtube.com/watch?v=M3iI2l0ltbE. [Accessed: Feb. 12, 2025]

"The PCD (Point Cloud Data) file format — Point Cloud Library 1.14.1-dev documentation," *Point Cloud Library*. [Online]. Available: https://pointclouds.org/documentation/tutorials/pcd_file_format.html. [Accessed: Dec. 20, 2024]

You Only Look Once Machine Learning Solution to Orbital Debris Detection and Classification

New Mexico

Supercomputing Challenge

Final Report

April 2, 2025

La Cueva High School

Team Members:

Hadwyn Link

Ximena Serna

Teacher:

Jeremy Jensen

Project Mentor:

Mario Serna

Table of Contents

Table of Contents **2**

Abstract **3**

Introduction **4**

The Problem 4

The Objective 5

Solution **6**

Orbit Generation 6

Orbit Visualization 7

The Setup 7

Optimization 8

Dataset Generation 10

Object Detection 10

The Structure of YOLO 10

Attempts at Making a Custom Network 12

Training YOLOv5 Instead 12

Results **13**

Conclusion **14**

Acknowledgements **15**

Works Cited **16**

Links To Products **19**

Graphs and Tables **20**

Graph of loss and precision during training 20

Confidence Curve 20

Precision-Recall Curve 21

Precision-Confidence Curve 21

Recall-Confidence Curve 22

Abstract

As the amount of debris in Low Earth Orbit (LEO) increases, satellites are more likely to collide with it. Debris travels at such high speeds that even small debris colliding with a satellite could cause catastrophic damage. As satellites are crucial to many important systems, it is important to protect these satellites. The common methods of preventing collision are by either removing the debris, forcing it to re-enter the atmosphere and burn up, or by maneuvering satellites around debris to avoid it entirely. Additionally, it is important to be able to classify objects in orbit to catalogue the type of debris. Classifying debris allows us to better determine risk and which method of removal to use. However, both classifying and detecting debris are extremely difficult with the current strategies. By using novel machine learning algorithms to more efficiently analyze debris classes and orbits, we can vastly improve the performance of satellite systems and debris removal protocols.

Recent studies pertaining to this field suggest trying You Only Look Once (YOLO), a new type of machine learning. It is extremely fast at detecting objects and works much more efficiently than any previous object detection models, making it more likely that it would work on less powerful space hardware. Our goal was to find how effective YOLO is at classifying debris in LEO.

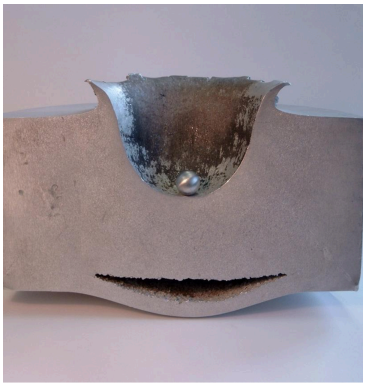
Our solution was to create a simulation which can test YOLO's ability to classify debris. To do so, we generated over 14,000 orbits containing information such as position versus time. With this information, we created a graphical simulation of the debris orbiting Earth. Then, we procedurally took screenshots of the simulation and saved object classes and positions to a file. After slight adjustment, the images were fed into the YOLO program for training, and once it was fully trained we fed the simulation directly into the model to see how it would perform in a real-time environment as opposed to still pictures.

The program finished its training with a very high percentage accuracy of classification. The program had minimal difference between the different types of background in the images. Our results support the conclusion that YOLO can classify debris accurately and can be implemented for the purpose of addressing free floating debris in space. We encourage future researchers to continue this course of study for possible space implantation.

Introduction

The Problem

In the past few decades, society has grown to rely on wireless internet, smart phones, and instantaneous communication networks. Many of these commodities, however, rely on satellite technology. With most of the modern world dependent on satellites in some way, it is important to address problems that threaten their existence. By far the largest one is debris; these pieces of residual payloads, broken satellite parts, and asteroids have begun to litter Earth's orbit on an unprecedented scale. As more satellites are put in space, more debris is created: more payloads are left over, and more old satellites are forgotten and broken down. A piece of debris can reach a speed of 18,000 miles per hour, seven times faster than the speed of a bullet. At that velocity, even a collision between a flake of paint and a satellite could prove catastrophic. The image to the left demonstrates the proportion size between a piece of debris in Low Earth Orbit and its collision impact crater. A single collision could easily decommission a satellite, as well as create more debris to worsen the problem. Repairing damaged satellites costs tens of billions to hundreds of millions of dollars, especially if it requires a human mission. Leaving them vulnerable jeopardizes every function they perform, is a danger to society's way of life, and makes sustainability in space very precarious.



Current methods to detect debris are radar systems which can pinpoint locations of nearby objects as well as contact with a ground-based tracking system with more heavy-duty tools. Neither of these methods can directly determine what exactly the debris is but can collect information such as material. However, in some cases this may not be enough to fully classify the debris, and in the case of ground-based tracking it has a delay between data transfers and can also be very expensive to operate the ground-based tracking systems. Another, newer option is to stream images down from the satellite and perform object detection algorithms on earth. However, streaming such a large amount of data from a satellite would be very expensive and would have a large enough delay for the data to be outdated when it reaches the satellite—remember, debris can move at up to 18,000 miles per hour.

Without information about the debris near a satellite in real time, there is little we can do to reduce collisions. Solutions such as removing the debris, forcing their orbit to burn on reentry,

or having satellites maneuver around debris would all require information about the class and location of the debris that would be necessary in order to determine which approach to use. Clearly, another method is necessary to accurately and completely classify and detect debris with attention to both cost and speed.

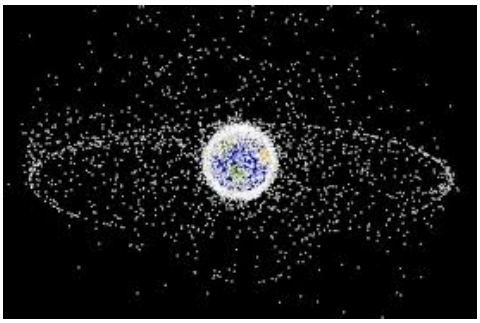
The Objective

Our proposed solution is to find a way of detecting and classifying pieces of debris with a common type of artificial neural network: YOLO. First introduced by Joseph Redmon et al. in 2015, it is a quick and efficient object detection neural network. It has never been implemented in space before but has been proposed for being more effective than current methods of detection according to studies by Mahendrakar et al. and Ahamed et al. However, no previous studies have suggested any ability of YOLO's classification skills in space.

Classification of debris could help us understand the risk of each piece of debris, provide information on the best way of removal, and gauge whether the debris would burn up in the atmosphere during reentry. Providing this information would be valuable for addressing the debris problem and is an area that has not yet been explored in YOLO's abilities, thus addressing a gap in scientific understanding.

YOLO sets itself apart from many other machine learning algorithms because it is the most likely object detection model to work on current satellite hardware because of its speed and relatively low resource requirements. Realistically, a surveying satellite does not have the time to be able to detect and classify every object it sees. Instead, it has to be able to process the required information before the object disappears. Speedy calculations will lead to a more efficient system of surveying and give the satellite a greater amount of time to act and attempt to remove or avoid the debris. Because LEO has the highest concentration of debris, our simulations have been based on receiving its data from this altitude (2,000km or less). At the conclusion of this study, our results should show how effective YOLO is at detecting and classifying debris in 7 classes: Asteroid, Cube Satellite, Envisat, Voyager, Hubble, the ISS, and the SaturnV5 Rocket. These classes include common objects seen in space along with specific, universally recognised objects that would demonstrate YOLO's ability to classify different types of objects such as Envisat, Voyager etc.

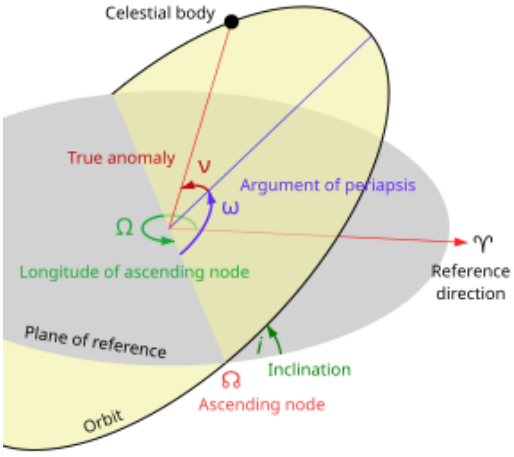
By separating the project into two sections: generating the debris orbits, and creating and training the YOLO program. We generated the debris orbits based on a dataset of real debris orbits, which output individual characteristics of around 14,000 orbits and saved each orbit to a file. These generated files, containing the position versus time of each item of debris, were then converted into a real-time graphical simulation of the debris, as shown in the image to the left. The graphical simulation was adjusted to demonstrate the vantage point of a LEO satellite. Screenshots of the simulation from this vantage point were fed into the YOLO model for training. Afterward, the YOLO was able to receive information live from the generated debris visualization.



Solution

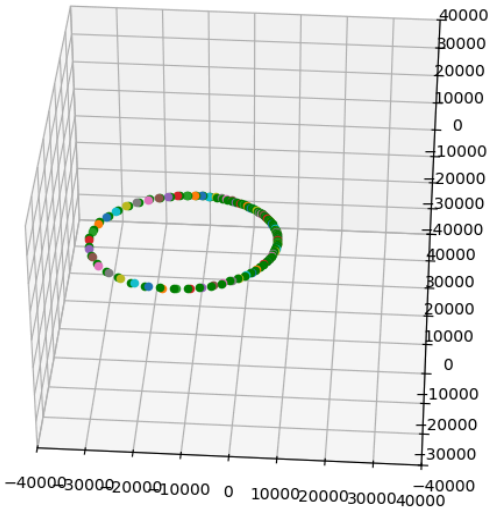
Orbit Generation

The debris orbits were generated from a dataset obtained from Kaggle. The dataset quantified one debris item per line summing to a total of around 14,000 pieces of tracked debris and 14,000 lines of data. To describe each orbit, the dataset gave over 20 Kepler's characteristics along with name, ID, country of origin, and date of creation using a Comma Separated Value(CSV) file. As the information may suggest, the debris information was recorded from real debris in space and does not contain randomly generated numbers.



Kepler characteristics are values that describe an orbit in a way that allows us to derive specific information about its path. For example, a characteristic could include an object's closest and farthest point radius, eccentricity, inclination off the xy plane and ‘w’ rotation away from the zy plane. Using

these values, the radius of the debris from earth are given by $R = \frac{A^*(1-e^2)}{1+e^*cos(V)}$ where 'A' is the semi major axis, 'e' is the eccentricity, and 'V' is the true anomaly (angle from $\theta = 0$). A python program input the required values and calculated the radius based on incrementing θ, calculating one full orbit at



a time. This provided a basic outline of the orbit's position over an orbit. It was then rotated off the xy, yz, and zx axis based on the 'w,' 'i,' and the RA node. The rotation matrix used for adjusting the orbits is given by $R = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$. This translates the Kaggle orbits into the 3D path of the orbit.

After the piece of debris' actual path through space has been found, the time steps need to be calculated. To do this, we used the conservation of momentum, $P = mv$ to find the time between each calculated position. The velocity was calculated by subtracting the potential energy from the total energy and solving for 'v'. The equation for total energy is given by $E_{total} = -\frac{GMm}{2A}$, the potential energy at the time of measurement is $U = -\frac{GMm}{r}$, and the Kinetic energy is given by $K = \frac{1}{2}mv^2$. Using pythagorean theorem, we calculated the distance between the said measurement and used $\Delta t = \frac{distance}{v}$ to find the time. At the completion of finding each time step, the information was saved to a CSV file for further use. It is saved to the format *time since the start of orbit*, *x*, *y*, *z*. The python program repeated the process for the rest of the debris, reading one more line of the dataset and creating another debris CSV file until every debris file was made.

Orbit Visualization

<u>The Setup</u>

Each individual orbit is saved to a CSV file, with each line containing information about the x, y, and z coordinates of the debris along with the time. There are 14,372 debris orbits in total, with two extra files for the orbits of the Earth and moon. The Sun orbits around the Earth for the sake of not having to deal with massive

floating-point numbers causing precision loss on satellite positions. Instead of a full orbit it changes between 8 approximation positions in a circle around the Earth. The Earth also rotates around its axis on a 24-hour time interval. To keep the sun and earth's rotations synced up with the debris' timescale, they use one of the orbit files to determine the timescale being used, and then take reference to the timestep while using their own logic for positioning. Below is what our simulation currently looks like. The sun is not directly seen by the satellite at its current angle, but it helps simulate the day/night cycle.

For convenience, the simulation is sped up significantly, and the xyz coordinates of every orbit are decreased by a scale of 256. This scaling down is necessary because with 3D simulations such as Unity, as a number gets larger the amount of floating-point precision a number can hold decreases. By scaling down the number we get more decimal precision, and it is easier to move around the environment during testing. The earth and moon are scaled down in size to match the coordinate shrinking, so the simulation is an overall 1/256 scale model of the Earth-Moon-Sun system.

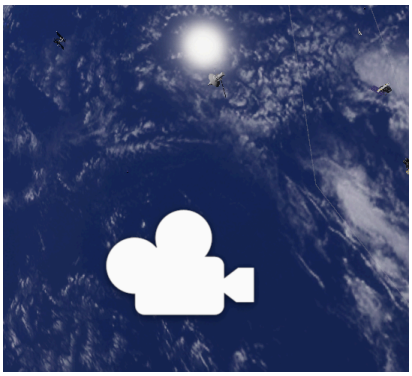
At runtime, a debris generator object instantiates all 14,372 pieces of debris with one of seven random debris models. Each piece of debris is instantiated at a random angle as well. The camera is attached to a survey satellite in the same orbit zone as the debris, and pointed directly towards the Earth. The debris are scaled up to 100 times their real world sizes to make it easier for the low resolution images we are training the neural network with to actually capture the 3D models. This would not be as much of a problem in a real-world situation because the camera would take higher resolution images that could capture important features from further away.

Optimization

14,372 pieces of debris is a lot to simulate individually. Each piece must update its position in almost every frame and has a fairly dense 3D model attached to it which also uses a significant amount of computing power. On a RTX 3060 graphics card and an intel i9 cpu, the simulation runs at under 10 Frames Per Second(FPS) without optimization. The simulation is

intended to run alongside a neural network and must run at or near “real-time” (30-60 FPS) to be viable.

To optimize the simulation without compromising on the amount of debris, everything the camera can’t see doesn’t need to be fully simulated. We can do this by checking if a debris item is past a certain distance from the camera, and if it is we turn its model off. By running this every time when a piece of debris moves, we create a “mask” around the camera where things are visible, and hide everything else.



This saves some computational power, but not enough to get a “real time” frame rate. The main thing causing FPS to drop is the number of times each debris item changes position. Every piece of debris updated its position every few milliseconds or so. This meant the CPU was always swamped with requests, making each frame take longer to process. To fix this, we again use the principle of not spending resources on debris the camera cannot see. To do this, we again need to know how far the debris is from the camera. First, we have a script on the debris that calculates each time it moves how far away it is from the camera. Then, we calculate an “optimization number”-a number that the debris’ delta time and lines skipped will be multiplied by, making its motion choppier (but still following the same path at roughly the same time) and less computationally expensive. After trying a few different methods, we settled on taking ‘e’ (Euler’s number) to the power of the distance minus the radius around the camera that we want the debris moving smoothly in with a minimum value of one and a maximum value of 45, rounded to the nearest integer. Using this function means we get a tight sphere around the camera where everything moves smoothly, but after that point the debris gets optimized quickly (but not so much that the orbit becomes completely unusable). By doing this, the simulation can now run all 14,372 pieces of debris simultaneously with a range of FPS between 30-60. Another thing to note is that the simulation is rendered at a resolution of 256x256, for two reasons: One reason is that it makes the simulation run faster, and the second reason is that it makes the neural network faster to train and run. In a real-world scenario the resolution would

need to be much larger than this, so there would be some necessary loss in performance past what we use in this project in order to make the model work better in space.

Dataset Generation

With the simulation complete, the dataset for the object detection model could be generated. This is relatively easy with Unity's built in GUI functions: existing functions can detect which objects are on screen and where they are on screen (this does not use machine learning, as Unity has access to the 3D simulation and can figure all of this out deterministically). We used more GUI functions to draw the rectangles on screen as a sanity check for the boxes before we screenshot each frame of the simulation and write the important information such as class and bounding box of each object to another CSV file (YOLO uses the format center x, center y, width and height for its bounding boxes). Below is an example of one of these boxes and the sanity check:

For the most part this gives relatively good bounding boxes to use in training the neural network. This code runs until each object class has a certain number of screenshots with it in the picture. We generated 100 images per class as this is the recommended minimum for such a model.

However, the format of the dataset cannot be used with YOLO out of the box: the standard format for YOLO requires a multitude of text files containing object information with one text file per image rather than a CSV file with every image's data in it. We made this transition with a single python script. We also divided each value for the bounding box by the resolution of 256 by 256 since YOLO works off of screen ratio rather than pixels, and changed the definition of the Y value on the image from starting at the bottom, as Unity has it, to starting at the top. After everything was in the proper format, the YOLO object detection model could be trained.

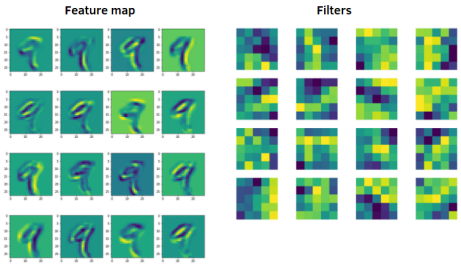
Object Detection

The Structure of YOLO

YOLO stands for You Only Look Once, and it is currently the most popular

neural network for object detection. There are many different versions of YOLO, ranging from the original YOLO to the most recent YOLOv12 model. During this challenge, we focused on YOLOv4 but eventually switched to YOLOv5 after we ran into a critical bug in our v4 code that required us to switch over to v5 just in case we couldn't solve the bug in time.

YOLO's structure consists of three parts: a backbone, a neck, and a head. The backbone is a convolutional neural network for object classification, and is the part that actually determines what objects are in the scene as well as a great deal of where they are. A convolutional neural network classifies objects using convolutional layers, which extract various "features", such as closed circles or even something as complicated as the contours of a face. Below is an example of what these features and feature extractors look like if visualized.



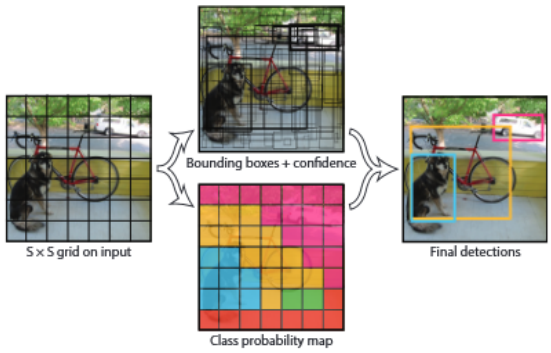
Typically, convolutional neural networks are used to identify a single object in a picture as a standalone neural network, but YOLO modifies the network by essentially splitting the given image into a grid that the backbone classifies individually. This gives the neck and head a starting approximation of where each object is as well as what the objects are. This is the largest part of the neural network, and does most of the heavy lifting.

The neck is relatively small compared to the backbone and head, but serves as an important bridge of information between the two. This section collects the features that were extracted from the image during the backbone section and sends this information to the head. This is commonly done with a Feature Pyramid Network (FPN), which can aggregate features on several layers of the backbone at different scales (convolutional layers shrink the image's dimensions, creating vastly different sizes of features). By feeding accurate data into the head, the head works better and can draw more accurate boxes.

The head is where most of the output of the neural network is put together. The head takes the output from the neck and runs it through a series of "YOLO layers" which generate thousands of boxes. Each box has its own confidence value representing how confident YOLO is that there is an object in the box and class prediction. The class

prediction is a list of confidence values where the index of the highest value in the list is the class it believes is in the box. (Object types during computation are represented by numbers.) YOLO outputs a list of three separate tensors with these boxes for low, medium, and high box size based on the different scales the neck inputs to the head. This ensures that regardless of an object's proximity to the screen or relative size, the neural network can still pick up on it.

However, YOLO's output is completely unfiltered and outputs thousands of boxes that need to be sorted through to find the best ones. This is done by the second step of the algorithm, which is separate from the neural network and deterministically culls the low-confidence boxes so that only the most accurate ones remain. This turns thousands of useless boxes into a select few usable boxes that are much closer to the actual object's position and class.



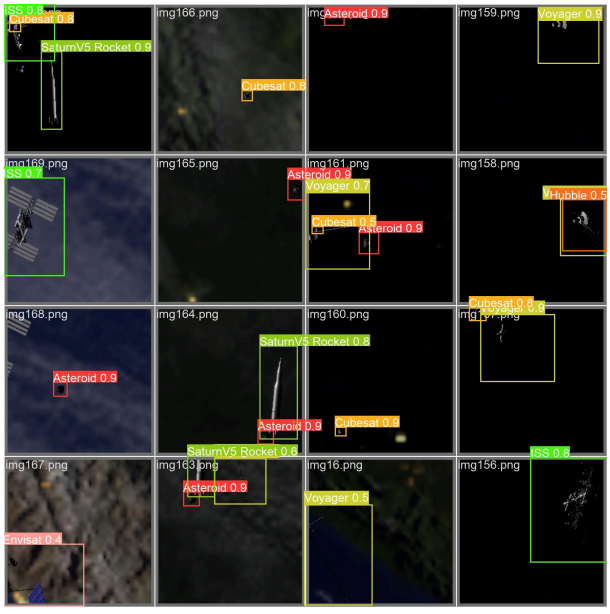
Attempts at Making a Custom Network

Our original idea was to make a custom implementation of YOLO based on researcher Wong Kin-Yiu's Github repository demonstrating YOLOv4's architecture in Pytorch. We coded this implementation with a dataloader that would have taken our original CSV file dataset format, along with a more specialized structure for our specific needs. The bulk of the code was finished in late February, but when we tried to train the model, the loss of box accuracy plateaued at 0.3 (a very high number) and the output was completely unusable. Unfortunately, attempts to fix this issue didn't look promising, and after a month of bug fixing we decided to switch to a professional implementation of YOLO and continue fixing the problem after we had something working. Having to use a version of YOLO we didn't personally program and customize was disappointing, but through making our own implementation at the start we were able to understand the structure of the tool on a deeper level than if we had started out with a premade model.

Training YOLOv5 Instead

Since we already had a large amount of experience getting our YOLOv4 set up, getting YOLOv5 to train on our dataset was

simple. We switched the dataset from the CSV file to a set of txt files so the dataloader could read it, and added a .yaml configuration file that would point towards our dataset and the number and names of our classes. After this, we loaded up a python virtual environment to run 100 epochs of the YOLOv5 nano model, which is the smallest version of YOLO that is typically used. We chose this version because it has the best chance of running on current space technology, while also being accurate enough to pick up on all of the features it needs to pick up.

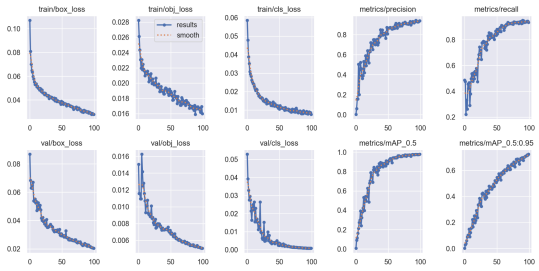


As seen above in one of the output testing images, the model draws boxes over each piece of debris it detects along with the class it believes the object is. After it was trained, we ran the neural network simultaneously with the graphical simulation and visualized the output much like the above image. This worked extremely well, even when the camera was looking at a darker image like when it was nighttime. To improve accuracy at a marginal increase of computational stress, we could increase the resolution and the number of classes. However, we have not given up on fixing YOLOv4. We will continue work on it after the report is finished and try to get it fixed for the expo.

Results

YOLOv5 ended its training with a box loss of 0.278, an object loss of 0.016, and a class loss of 0.007. Its ending precision was slightly under 94%. If necessary or if we wanted to add more classes, we could train it for longer to get an even more accurate model. The “nano” model structure didn’t even need two computers to run it in real-time as we originally planned, and it could run easily alongside the real-time graphical simulation and a communication script that fed the simulation’s window into the neural network. Here is the YouTube video that demonstrates the simulation and the detection side by side: https://www.youtube.com/watch?v=W5LQczcIe_4. While there is a slight delay in

presentation of the detection boxes and the original simulation due to using python as a visualizer of the boxes, the detection algorithm ran at a much higher framerate than the simulation, running at hundreds of FPS and drawing accurate boxes around debris. Additionally the simulation's debris and angle are randomly generated, so there is little overlap between what the neural network explicitly trained on and the material it was tested on here. The model's high performance in this case indicates that the neural network did not overfit while training. The neural network was also able to perform well during the night, demonstrating its ability to accurately discern class and box location based on less clear information. If required we could easily up the image resolution to something more accurate for a satellite, as well as improve the model from YOLOv5 Nano size to small or even medium with a manageable tradeoff of speed in return for more accuracy. Below is the graph of our loss metrics (left) and accuracy (right) during training. For a larger version of this graph along with other important graphs, see the end of this document in the Graphs and Tables section.



Conclusion

YOLO has proven to be an accurate, performant way of analyzing and classifying space debris. However, it is also rather delicate and difficult to get working, and there are many edge cases and direct attacks that can decrease its accuracy significantly, as is the case with many neural networks. With our own implementation, programming around these issues and getting it running in the first place was complicated and required us to learn a great deal about neural networks, convolutional neural networks, and object detection algorithms in general. This makes it our most significant achievement in the project, even though we were unable to get YOLOV4 working in the final product.

YOLO as a solution to detecting and classifying objects in space has some downsides. Primarily, it can only classify objects it has been explicitly trained on. We used only seven classes in our project, but there are hundreds, if not thousands of types of debris currently in orbit and it would be impossible to catalogue them all and train a

network on it. Classification needs more training the more classes you train on, and even though the relatively primitive YOLOv2 could hypothetically guess over 9000 classes relatively accurately (for its time), this creates a larger neural network that requires more memory and power. However, one advantage indicated by research is that a larger neural network would be more "robust" when exposed to radiation than a smaller one, as there would be more redundancies and pathways that the model could still perform relatively well without.

Additionally, YOLO cannot explicitly find the distance to or relative position of the debris in 3D space, just where it is on the screen (Although this can be paired with other algorithms to figure this part out). Finally, we once again run into the issue of performance: current space technology is at least ten years behind modern graphics cards and CPUs, and even in 2017 the most efficient object detection algorithms ran at 6 FPS and were considered extremely performant at the time, as seen in Tsung-Yei Lin's paper on feature pyramid networks. This is due to radiation being hard on modern, fragile graphics cards and frying them easily, along with the amount of power neural networks use. This is a relatively small problem, as even now there is research going into getting neural networks such as this into space and even some commercially available satellite parts that YOLO could reasonably run on without too much trouble.

These weak points are a significant concern, but using other technologies in combination with object detection can cover these weak points and result in a much more accurate debris detection system than we currently have. In particular, a supporting onboard radar system would be able to tell where exactly the debris is (one of the problems with YOLO) as well as provide auxiliary information about objects such as their material or mass. This last point is important because it would let YOLO train on fewer, more general classes of objects (such as satellite, rocket stage, etc.) but still have enough context clues for a deterministic algorithm to piece together the output of both the radar and YOLO for a much more extensive list of detections than either one could give alone.

Acknowledgements

Special thanks to Francisco Viramontes, one of our teammate's internship mentor who has provided helpful insight on both the inner workings of YOLO as well as advice on how to frame our problem to be both accurate to space technology's current ability to handle machine learning algorithms, as well as giving us ideas as to how machine learning could be used effectively in space.

Regina Hunter also helped in proofreading this paper in the final stages, and helped us catch points where we needed to elaborate for clarity of concepts.

We would also like to thank Jeremy Jensen, our team Sponsor for the last three years of competition. Without him we would not have been able to compete, and it would have been much more difficult to communicate with SCC without him.

Finally, Hamilton Link provided us with crucial input on troubleshooting the YOLOv4 network along with Mario Serna who helped with our orbital simulations and calculation.

Works Cited

Bochkovskiy, Alexey, et al. "YOLOv4: Optimal Speed and Accuracy of Object Detection." Institute of Information Science, 23 April 2020, YOLOv4: Optimal Speed and Accuracy of Object Detection. Accessed 15 January 2025.

David, Leonard. "Space Junk Removal Is Not Going Smoothly." 14 April 2021, https://www.scientificamerican.com/article/space-junk-removal-is-not-going-smoothly/. Accessed 20 March 2025.

"Earth Textures | 1.0.0." Github, 29 May 2024, https://github.com/RSS-Reborn/RSS-Earth/releases/tag/V1.0.0. Accessed 6 February 2025.

"Envisat." Sketchfab, 2016, https://sketchfab.com/3d-models/envisat-65b0ec49681a44f68dfc8bd4efe95839. Accessed 30 January 2025.

"Hubble space telescope." Sketchfab, https://sketchfab.com/3d-models/hubble-space-telescope-e22236fab9634c959c0525c7ab9c83d7. Accessed 20 January 2025.

"International Space Station 3D Model." NASA, 22 April 2019, https://science.nasa.gov/resource/international-space-station-3d-model/. Accessed 17 February 2025.

Keeter, William. "NASA 3D Resources." NASA, https://nasa3d.arc.nasa.gov/detail/cubesat-1RU%5C. Accessed 20 February 2025.

KHANDEKA, KANDHAL. “SATELLITES AND DEBRIS IN EARTH'S ORBIT.” Kaggle, 2022, https://www.kaggle.com/datasets/kandhalkhandeka/satellites-and-debris-in-earths-orbit. Accessed 4 November 2024.

Lin, Tsung-Yi, et al. “Feature Pyramid Networks for Object Detection.” Cornell University and Cornell Tech, 19 April 2017, https://arxiv.org/pdf/1612.03144. Accessed 1 February 2025.

Mahendrakar, Trupti, et al. “SpaceYOLO: A Human-Inspired Model for Real-time, On-board Spacecraft Feature Detection.” Florida Institute of Technology, https://arxiv.org/pdf/2302.00824. Accessed 29 October 2024.

“Orbit Orientation.” AI Solution, https://ai-solutions.com/_freeflyeruniversityguide/orbit_orientation.htm. Accessed 24 December 2024.

“Planetary Physics: Kepler's Laws of Planetary Motion.” *Orbits and Kepler’s Laws*, NASA, 26 June 2008, https://science.nasa.gov/resource/orbits-and-keplers-laws/. Accessed 30 October 2024.

Redmon, Joseph, et al. “You Only Look Once: Unified, Real-Time Object Detection.” https://arxiv.org/pdf/1506.02640. Accessed January 2025.

“Space Debris.” *HQ Library Navigation*, NASA, https://www.nasa.gov/headquarters/library/find/bibliographies/space-debris/. Accessed 26 October 2024.

"Space Debris Velocities." *RULES OF THUMB AND DATA FOR SPACE DEBRIS STUDIES*, NASA, https://science.nasa.gov/resource/international-space-station-3d-model/. Accessed 25 October 2024.

"Space Rocket Saturn V 3D Model." Free3D, 23 April 2020, https://free3d.com/3d-model/space-rocket-saturn-v-360313.html. Accessed 23 January 2025.

"Space Rocks." CGtrader, https://www.cgtrader.com/free-3d-models/space/other/space-rocks-9351486c-0cd0-46e7-ab36-62c8a621820d. Accessed February 14 2025.

"Voyager." NASA, 31 March 2025, https://nasa3d.arc.nasa.gov/detail/jpl-vtad-voyager. Accessed 11 February 2025.

Yiu, Wong Kin. "PyTorch implementation of YOLOv4." Github, https://github.com/WongKinYiu/PyTorch_YOLOv4. Accessed 27 January 2025.

"YOLO9000: Better, Faster, Stronger." Allen Institute for AI, 25 December 2016, https://arxiv.org/pdf/1612.08242. Accessed 10 January 2025.

"YOLO, Ultralytics." Github, 22 November 2022, https://github.com/ultralytics/yolov5. Accessed 2 January 2025.

Links To Products

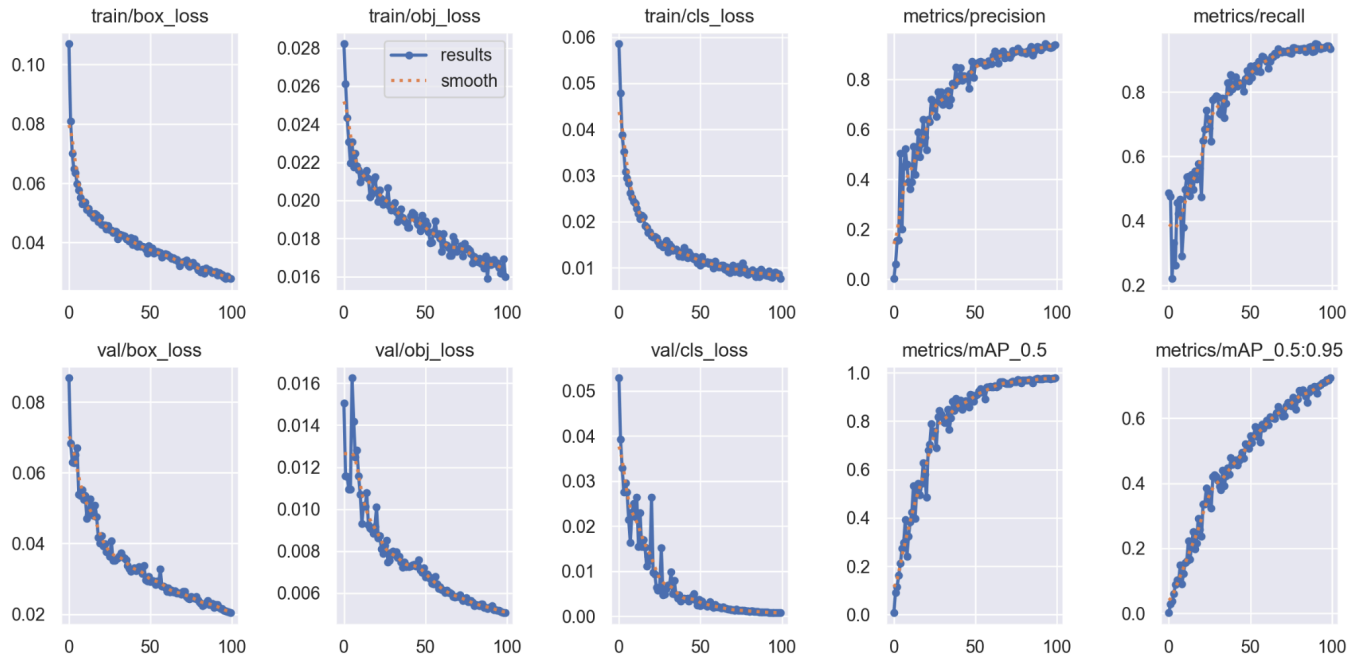
https://github.com/HadwynLink/SCC-2024-2025

(results table can be found in Yolov5/Runs/results.csv)

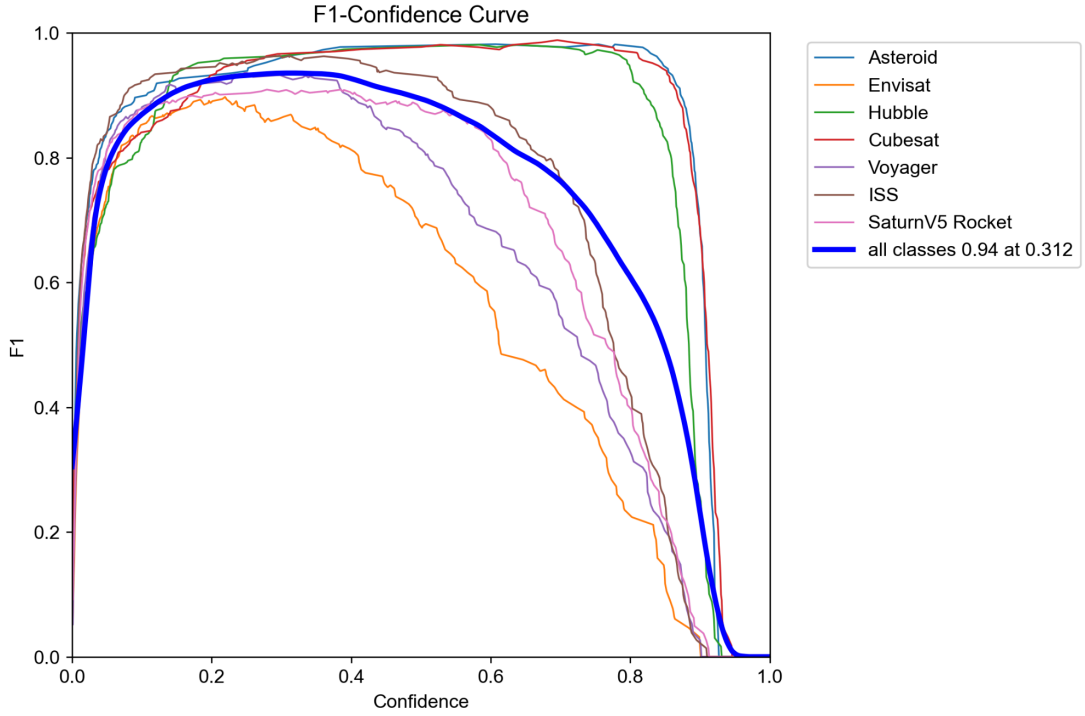
https://www.youtube.com/watch?v=W5LQczcIe_4

Graphs and Tables

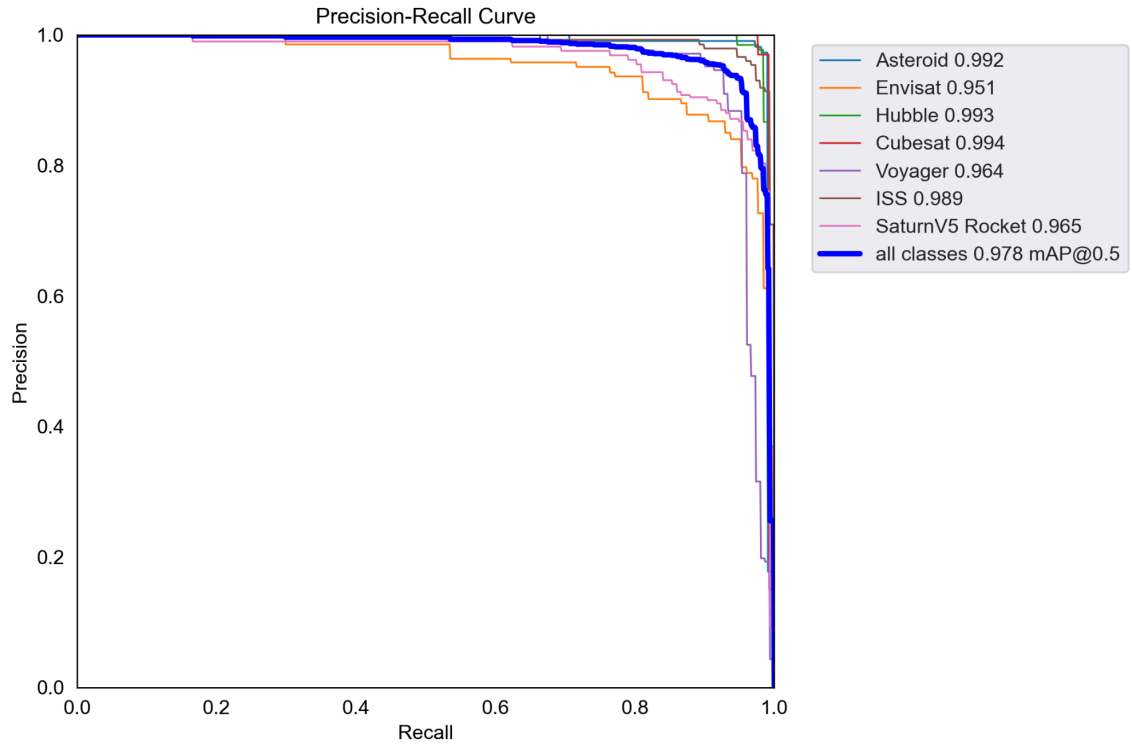
Graph of loss and precision during training



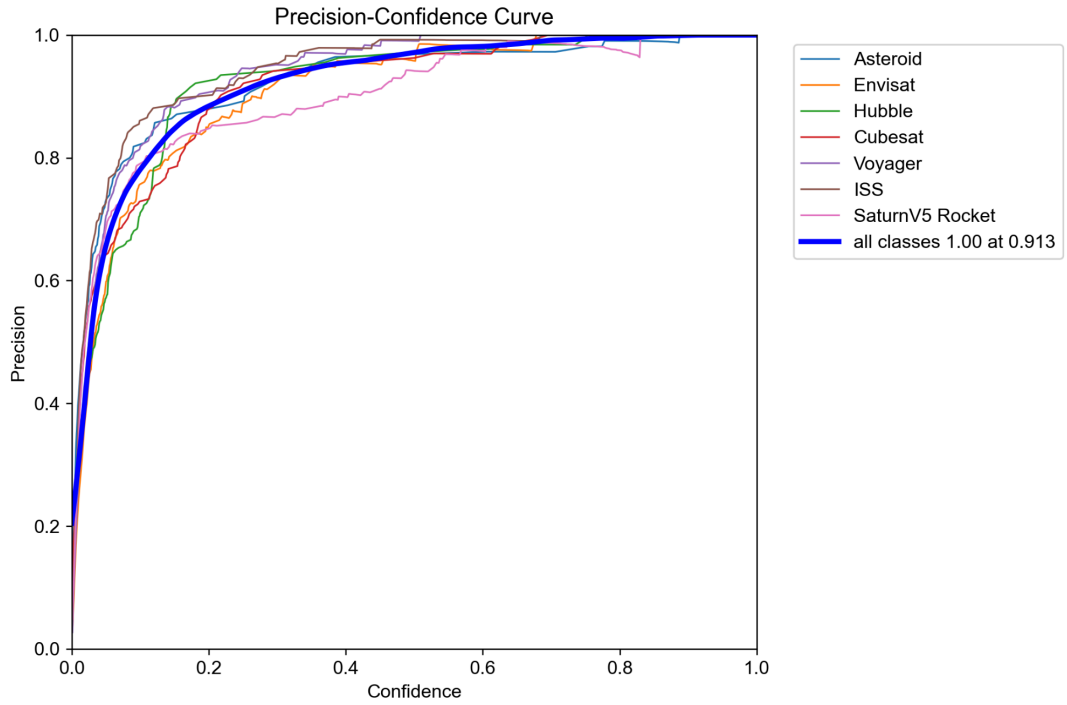
Confidence Curve



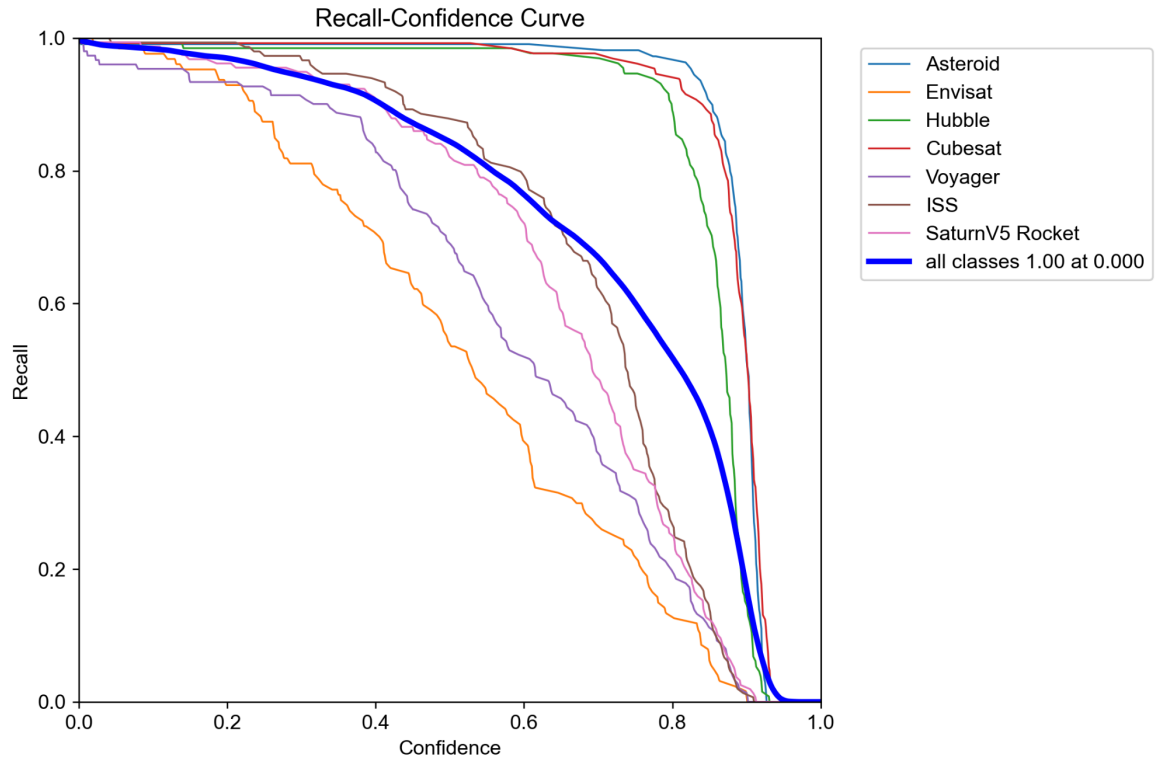
Precision-Recall Curve



Precision-Confidence Curve



Recall-Confidence Curve



Albuquerque Academy Team 1 Additive Manufacture Kinetics and Thermodynamics Model

New Mexico

Supercomputing Challenge

Final Report

April 25, 2025

AA Team 1

Albuquerque Academy

Created by

Harrison Schiek

Teacher Mentors

Jay Garcia

Alex Benedict

Table of Contents ----------

- Executive Summary 2
- Problem 3-4
- Methodology and Algorithms 4-16
- Verification and Validation 16-24
- Results: Analysis and Conclusions 24-26
- Future Use: Development and Testing 26
- Bibliography and Acknowledgements 27-29
- Appendix A 29-30

Executive Summary ----------

Problem

As Additive Manufacturing (AM) and, more specifically, Fused Filament Fabrication (FFF) grow in importance and prevalence, it is essential to understand bonding conditions for materials and resultant material strength. Especially for high-strength applications like aerospace and construction and high-precision applications in nanotech and biomaterials, manipulation and careful characterization of materials is crucial. With its introduction into these fields, AM will lower costs and break down barriers for research and innovation. It is essential to expand support for AM applications through bettering understanding for improved material effectiveness.

Solution and Methodology

The filament was modeled as a set of non-colloidal particles using Molecular Dynamics and other interactions. Two kinds of exchange occur between nodes of the simulation; the first being thermal energy, and the second being kinetic energy. To model the first, the Heat Equation was applied using simple approximations of the second derivative. For kinetic exchanges, Lennard Jones potentials, friction, and torque captured the interaction with high fidelity. Temperature was also included in gauging friction and bond potential energy to increase accuracy. To evaluate bonding strength, measured temperature and strain during solidification contributed to an aggregate measure for characterization thereof.

Validation and Verification

By testing each component of the model in individualized trials, the model is shown to accurately predict logical results and textbook examples. To further verify their effectiveness, material characterizations were compared with literature material properties at temperature.

Results

The model indicates two major changes to improve FFF material effectiveness. The first change is higher plate temperature. This provides for the slow and steady bonding of the material. The second conclusion is that effective high temperature control of the bonding surface increases bonding quality, as maintaining even high temperature ensures key interlayer bonding.

Problem ----------

There are myriad reasons to value AM highly as a target for not only research but also investment. AM isn't a new idea, but many of its extensions and specializations, especially FFF, have grown immensely in recent years and show promise to revolutionize numerous industries that were previously unaffected by this type of manufacture. Lack of understanding now prevents broader applications, as gaps in precision and material strength limit usability. Overall, due to AM's growth, versatility, and environmental importance, it is essential to better understand and characterize bonding and product strength to broaden its impact.

AM technology has grown at an unprecedented rate and is projected to continue this growth. Specifically, growth of the global industry for AM from 2023 to 2030 is estimated to be 432.75% (around 23.3% annually)[1]. Importantly, FFF is the second largest contributory sector of AM[1].This growth makes it essential to support AM and FFF with better understanding and tools to predict performance. By facilitating more effective use of AM, greater uptake and positive effects will be seen across numerous industries. AM is also expanding into numerous new fields, where it promises to remove impediments to research and production. An article summarizing AM and its current uses states, "[AM] is rapidly expanding to a large number of industrial sectors such as aeronautics, automobile and biomedicine, with significant growth in the medical device and wearables markets"[2]. Given the wide spanning reach with an emphasis on precision and strength, predicting and printing for specific characteristics is crucial. Additionally, by improving understanding, problems with bonding and product durability will be reduced in severity. To empower the transition to AM, it is invaluable to improve understanding to reduce waste and cost and develop overall quality.

AM's versatility makes it crucial for large scale implementation. FFF machinery and equipment are almost independent of design, meaning costly specialization by product is nearly nonexistent. This allows flexibility for producers, lowering costs for experimentation and development. In addition, this makes much more possible for scientists. NASA highlights the importance of AM coupled with Thermodynamic modeling software specifically: "Applying these two processes (Thermodynamic Modeling and 3D Printing) has drastically accelerated the rate of our materials development. We can now produce new materials faster and with better performance than before"[3]. As seen here, effective modelling crucially streamlines development

processes and better supports science and industry. Adding on, a report by the US Department of Commerce notes the utility of AM in a broad variety of new uses due to its versatility. It states, "[AM] can facilitate the customized production of strong light-weight products"[4]. The wide spanning uses of AM as a result of its versatility illuminate the importance of utilization with confidence.

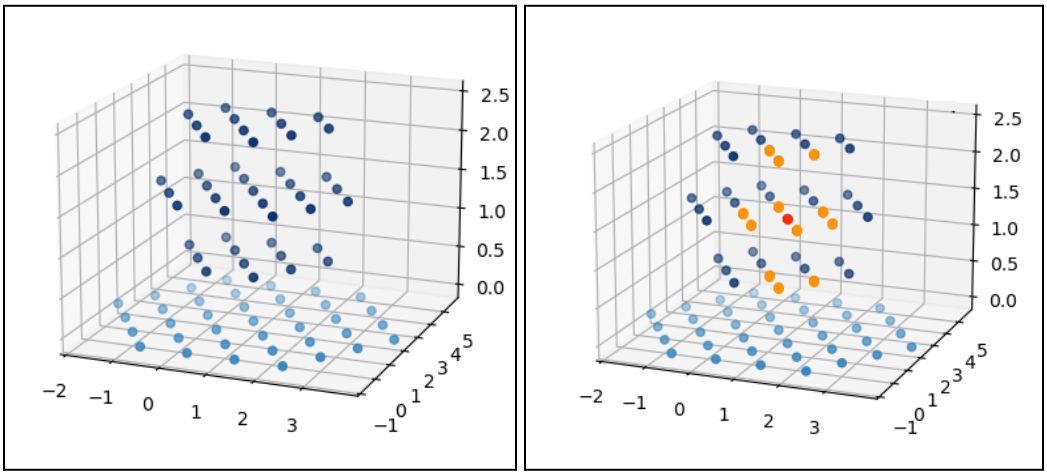
AM results in far less production waste, energy cost, and raw material usage. Its implementation will lead the way to importantly reducing the significant industrial burden on the environment. A paper, published in the journal *Advanced Industrial and Engineering Polymer Research*, focusing on AM as a sustainable manufacturing alternative reveals that AM results in "lesser material waste, energy usage, and machine emissions" as opposed to current manufacturing techniques[5]. Given these profound benefits over currently prevalent methods, the use of AM technologies to reduce overall pollution is promising. In the ongoing fight against Climate Change, the effective implementation and development of AM is a crucial step. The same paper emphasizes AM's importance in long term sustainability and climate efforts[5]. AM technologies play an important role as a more sustainable and environmentally friendly means of manufacture, and it is essential to support broad application and implementation.

Methodology and Algorithms ----------

The model consists of two major methods of energy exchange between nodes: thermal energy exchange and a kinetic energy exchange. Each of these methods will be examined in depth through foundations in established theory to implementation and program calculation. First, though, a description of the model framework and setting will set the stage for algorithmic usage and its computational effectiveness.

The model was developed from scratch exclusively in python using text editor BBEdit. It was run predominantly on a desktop computer with 3.4 GHz 8 Core Processor and a portable computer with 4.05GHz 8 Core Processor. Significant Python libraries Matplotlib and NumPy were used, the first for visualizations and the second for arrays and data organization. The model uses a Modelling by Decomposition approach to model a complicated 3D printed solid. It uses a hexagonal matrix of points to approximate the whole solid, and it manages internodal interactions by defining adjacency by spacing and considering only directly adjacent interactions.

The nodes can be thought of as small pieces of the whole not in any particular shape. As an example of adjacent nodes see Figure 1. The red node gives an example node and the orange nodes give all the adjacent nodes. The definition is just within 1.5 units of distance.



(Fig. 1)

Heat Exchange

The first physics feature of interest is heat exchange. To model this interaction, the Heat Equation was modified and applied to this nodal approach. Since the framework considers specifically particle-on-particle interactions, the complexity of the 3D Heat Equation was somewhat generalized for usage. A single variable *r* can represent distance between particles and replace position variables *x*, *y*, and *z*. *t* gives time. T gives the temperature, which is a function of *x*, *y*, *z*, and *t*. α is the thermal diffusivity. See Equation 1-1:

$$\frac{\delta T}{\delta t} = \alpha \nabla^2 T = \alpha\left(\frac{\delta^2 T}{\delta x^2} + \frac{\delta^2 T}{\delta y^2} + \frac{\delta^2 T}{\delta z^2}\right) = \alpha \frac{\delta^2 T}{\delta r^2} \qquad (1\text{-}1)$$

To adapt the equation further for use in the simulation, gaining an approximation of the second derivative for a point is crucial. To do this, consider (in only one dimension for simplicity) three nodes of temperatures T_1, T_2, and T_3, and positions x_1, x_2, and x_3. Approximating the first

derivative on the two sections, the two approximations are given by equations 1-2. The approximation is simply the equivalent of change in T over change in x for the function.

$$\frac{\delta T}{\delta x}_1 \approx \frac{T_2 - T_1}{x_2 - x_1} \qquad \frac{\delta T}{\delta x}_2 \approx \frac{T_3 - T_2}{x_3 - x_2} \tag{1-2}$$

Again using the same idea to approximate the second derivative at the center of the full interval and making the simple assumption that these particles are at the same distance from the central node (calling this distance r), the simplification and separation of interactions is shown (Eq. 1-3).

$$\frac{\delta^2 T}{\delta x^2} \approx \frac{\frac{T_3 - T_2}{x_3 - x_2} - \frac{T_2 - T_1}{x_2 - x_1}}{\frac{x_3 + x_2}{2} - \frac{(x_2 + x_1)}{2}} = \frac{\frac{T_3 - T_2}{r} - \frac{T_2 - T_1}{r}}{\frac{2r + x_2 + x_1}{2} - \frac{(x_2 + x_1)}{2}} = \frac{T_3 - T_2}{r^2} + \frac{T_1 - T_2}{r^2} \tag{1-3}$$

This shows the final simplification and manipulation of the heat equation for use in one-on-one particle interactions. Equation 1-3 really shows how the change in temperature at a point is given in approximate by the sum of nodes around in their change in temperature divided by the square of the distance, thus making these interactions simply calculable for any given set of nodes. See Equation 1-4 for the final usage of the Heat Equation, with Particle 1 being the particle from who's reference the calculations are made and Particle 2 being a particle adjacent to Particle 1 and thus included in the calculations. (Subscripts giving attributes of the particles and r being distance between particles). This is applied to a collection of nodes in 3D in the program.

$$\frac{\delta T}{\delta t}_{part} = \alpha \frac{\delta^2 T}{\delta r^2} \approx \alpha \frac{T_2 - T_1}{r^2} \qquad \frac{\delta T}{\delta t}_{tot} = \sum_{n=2} \alpha \frac{T_n - T_1}{r_n^2} \tag{1-4}$$

Using this relation, the model calculates heat flow from the difference in temperature and the distance between particles accurately.

Simple Internodal Forces

Next, the Lennard Jones Potential and its application will be discussed. The 12-6 Lennard Jones Potential has long been a precise way to model the intermolecular forces of very small particles. It gives the potential energy as a function of distance with several constants to control the depth of the potential energy well (the strength of the interparticle attraction or bond) and the distance of lowest potential. In this way, the equation can be adapted generally, and overall, inclusion was simple. See Equation 2-1 for constant choice.

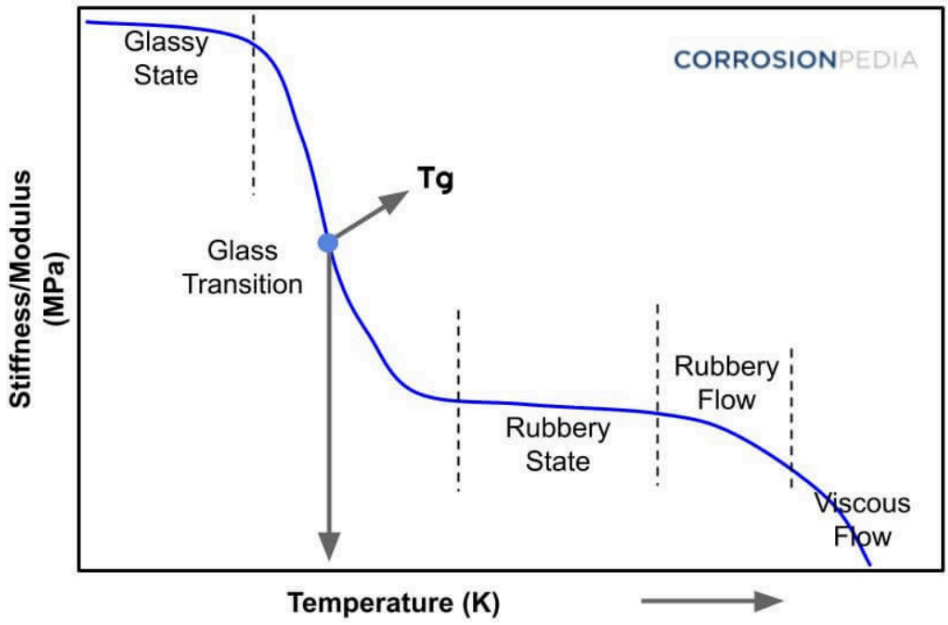
$$V_{LJ} = 4\varepsilon\left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right) = 4\varepsilon\left(\frac{1}{4r^{12}} - \frac{1}{2r^{6}}\right) \qquad (2\text{-}1)$$

Important to note: Despite the *V*, the Lennard Jones is not an electric potential.

This is the substitution of σ for $2^{\frac{-1}{6}}$ and leaving ε (depth of the potential well) variable for implementation as a function of temperature. The idea being that, with lower temperature, there is less (unshown) thermal kinetic energy fighting the attractive force, and thus, the overall observed bond potential energy will increase. A simple approximation of this phenomenon was used, given the glass transition temperature (T_g) (Eq. 2-2). E gives the preset bond energy, which, for the coarse grain of the model, can be approximated from state changes for the material in questions and also experimental results. Specific data and implementation are discussed later.

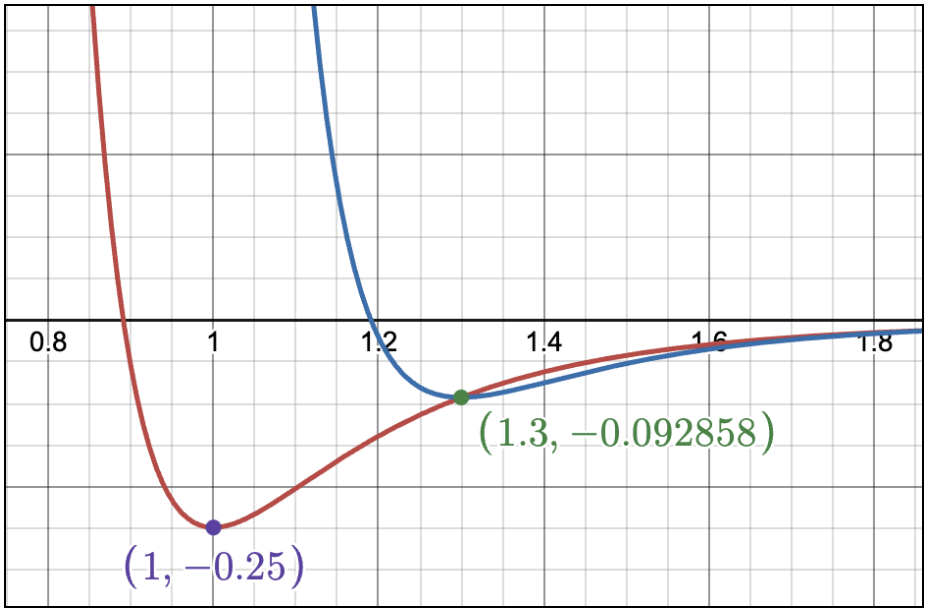
$$\varepsilon = E\left(\frac{4T}{T_g} - 3\right)^{-1} \qquad (2\text{-}2)$$

An important characteristic of this approximation is that it gives simply E for $T=T_g$. I.E. setting E will depend on what properties the material exhibits as it approaches its brittle 'glass' state before any temperature related changes occur to its chemical structure. Another important differentiation is that for the model, this approximation of bond energy is only used when $T>T_g$. Once $T<T_g$, the model diverts calculations to a different method as will be discussed. This function maps to this concept for thermal structural changes of polymeric materials (Figure 2).



(Fig. 2)[6]

When passing the Glass Transition Temperature from hot to cold, the material loses its malleability and becomes stiff. To model this change in properties, when particles in the simulation cross this temperature they solidify their bonds and become resistant to change. The most important means of doing this is in the reevaluation of force relations with adjacent nodes. Whatever the distance between the particles is, a new Lennard Jones potential is made with the new distance as the location of the potential well, and the new potential well depth as the current potential of the node based on the old Lennard Jones. See Figure 3 where the original potential is shown in red and the new potential (of a particle bound too far at a distance of 1.3) is shown in blue. Note the new depth of the potential and the location of its well. This process is a bit abstract, but in total, it is setting forces to zero for a new configuration and weighting the strength of these ‘new’ bonds on how strained they were before solidification. The factoring of current strain into future bond strength is crucial. This measure of strain is also used later for evaluation of bond quality.



(Fig. 3)

This well approximates several characteristics of polymers, specifically storing stress while 'solidified' and becoming resistant to any change once cooled. The math to create this in the model is fairly simple. The potential and distance at the time of 'solidification' are recorded. And used in this form to yield the new Lennard Jones Potential on demand. See Equation 2-3. Where old distance is given as *d*, the old potential is given as $V_{old}(distance)$, and r is the current distance.

$$V_{new} = 4\varepsilon\left(\frac{1}{4(r+1-d)^{12}} - \frac{1}{2(r+1-d)^{6}}\right)\frac{V_{old}(d)}{v(1)}$$

$$V_{new} = 4E\left(\frac{1}{4(r+1-d)^{12}} - \frac{1}{2(r+1-d)^{6}}\right)\frac{V_{old}(d)}{4E}$$

$$V_{new} = V_{old}(d)\left(\frac{1}{4(r+1-d)^{12}} - \frac{1}{2(r+1-d)^{6}}\right) \qquad (2\text{-}3)$$

This process is simply moving the well and changing its depth as seen here. Additionally, there is a small shift in the d to account for thermal warping. Of course, this is just a potential and does not immediately apply to the model. A derivative is necessary to make a relevant force for computation. See Equations 2-4 for the final implementations.

$$F_{old} = \frac{dV}{dr} = 12\varepsilon\left(\frac{1}{r^7} - \frac{1}{r^{13}}\right) \quad (2\text{-}4)$$

This equation gives an accurate force between nodes of the simulation and allows for kinetic exchange. However, this isn't the whole picture.

Complex Internodal Forces

Friction (or really drag) plays an important role in more concretely understanding these interactions at a coarser incrementation of distance. To gain a finer understanding without modelling millions of nodes, these additional calculations account for more properties. The model operates under a simple assumption that drag of a node operates like a Newtonian flow over its surface. Equation 3-1 gives the relevant drag equation (for flow over a surface giving net force on the ground) more clearly with $\Delta\hat{v}$ giving the vector difference in velocities**, $\hat{d}$ giving the distance vector, $\widehat{F_f}$ giving the force vector from friction, η being the viscosity, and A being the contact area. The viscosity is changed with temperature using the same algorithm as the bond energy.

$$\widehat{F_f} = \quad -\frac{\eta\Delta\hat{v}}{|\hat{d}|} \cdot A \quad (3\text{-}1)$$

One question about this assumption is should the difference velocity be considered regardless of the direction of the considered particle. An alternative projects the velocity to the distance and takes the orthogonal component to consider (Eq. 3-2).

$$\widehat{F_f} = \quad -\frac{\eta A}{|\hat{d}|}\left(orth_{\hat{d}}\Delta\hat{v}\right) \quad = \quad -\frac{\eta A}{|\hat{d}|}\left(\Delta\hat{v} - \frac{\hat{d}\bullet\Delta\hat{v}}{|\hat{d}|^2}\hat{d}\right) \quad (3\text{-}2)$$

This supplies a different approach, which separates motion directly toward or away the particle

** Unfortunately, this software doesn't support conventional vector notation, so this will show vectors.

in question from motion in the perpendicular plane. This makes more sense for smaller and smaller objects, but for this coarse grid, it makes more sense to include all directions given that the contact of these particles will not only resist perpendicular motion but also motion in the direction of the other node. Combined with the increased computational weight, this specification was not included. While this force was directly applied to the nodes, it is important to note that this force is applied off-center of the node, so angular momentum is relevant and needed. Using standard formulas for Torque, calculations for angular acceleration can be used. See Equation 3-3 with $\hat{\tau}$ as the torque vector. The distance is divided by two since the interaction occurs in the middle of the two particles.

$$\hat{\tau} = \frac{\hat{d}}{2} \times \widehat{F_f} \tag{3-3}$$

This is converted pretty simply to angular acceleration by using the moment of inertia for a sphere of evenly distributed mass. See Equation 3-4. *I* is the moment of inertia. *M* is the mass of the node. $\hat{\alpha}$ is the acceleration vector.

$$\hat{\alpha} = \frac{\hat{\tau}}{I} = \frac{\frac{\hat{d}}{2} \times \widehat{F_f}}{\frac{2}{5} M \left| \frac{\widehat{d}}{2} \right|^2} \tag{3-4}$$

Now that the angular velocity of the nodes has been established, a look back at the calculation of the net velocity of a nodal interaction for the definition of friction. Since each node is rotating, the observed velocity is different at each radial direction. Thankfully, this is calculated very cleanly using vector algebra. See Equation 3-5 for the definition of surface velocity from angular velocity. ω gives angular velocity (the resultant quantity of the angular acceleration shown in 3-4).

$$\widehat{v_s} = \widehat{\omega} \times \frac{\widehat{d}}{2} \tag{3-5}$$

This makes a net velocity calculation rather simple (Eq. 3-6)

$$\widehat{\Delta v} = \widehat{v_{sA}} - \widehat{v_{sh}} + \widehat{v_A} - \widehat{v_h} \tag{3-6}$$

This wraps up all sophisticated nodal interaction. These methods approximate the real interactions of parts within the system and lead to an accurate prediction.

Evaluation

When a node crosses the aforementioned glass transition temperature, the model grades the quality of the bonds, but to gauge what results in the actual and necessary effective bonding, it is important to define what leads to effective bonding in the first place. For this model, two major contributing factors were identified.

The first is the evenness of cooling. In modern manufacturing, the first issue is preventing uneven cooling. As summarized by industry manufacturer ZhongDe: "[It is essential] to solidify uniformly and minimize internal stresses, preventing issues like warping, shrinkage, and part deformation"[7]. Identifying uneven temperature during bonding is crucial as it leads specifically to material weakness and point failure. Specifically, a study of plastic composite cooling and solidifying highlights, "high cooling rates … cause uneven material shrinkage across different parts and pronounced warping defects"[8]. The model accounts for this process on a large scale by including thermal deformation on bonding, but finer details of temperature-related bonding are also captured. The model on grading inspects adjacent nodes and compares temperatures to the bonding temperature. The scale (S_T) gives high values for adjacent temperatures (T_a) near the bonding temperature (T_g) and lower scores for changes up or down (Eq. 4-1).

$$S_T = 10 - \left|T_g - T_a\right| \tag{4-1}$$

This positively reports temperatures close to the glass transition temperature and successfully identifies bonding situations where the particles are not at ideal temperatures such that warping and diminished strength are problems.

The second factor is residual stress during solidification. Many materials and especially plastics 'store' their stress in their material lattice structure when solidified under stressing conditions. These stresses impact their material properties lastingly. In the ScienceDirect chapter on Tensile Residual Stress, its effects are summarized: "Tensile residual stress often induces environmentally assisted cracking and fatigue crack initiation, resulting in crucial damage"[9]. To include the factoring of stress into the model and its grading of bonding a comparison of bond potential gives the necessary detail. The algorithm calculates the bond potential and compares it to the maximum potential to obtain a measure of its closeness in energy to the optimal bonding scenario (Eq. 4-2). S_S gives the stress score; V_{old} gives the original Lennard Jones Potential, and d gives the distance at which the nodes solidified (crossed the glass transition temperature). In essence this measures the potential of the bond and compares it to the maximum potential it could have, giving 1 for a perfect length bond and numbers less than one for strained bonds depending on the intensity of their strain. This gives the strain on the bond as it shows how much the bond has been forced to change.

$$S_S = \frac{V_{old}(d)}{V_{old}(1)} = \frac{V_{old}(d)}{4E} \tag{4-2}$$

To make a composite score, both scores were normalized and added together so that the maximum score was 100 for readability (Eq. 4-3).

$$Score = 50S_S + 5S_T = 50\frac{V_{old}(d)}{V_{old}(1)} + 5\left(10 - \left|T_g - T_a\right|\right) \tag{4-3}$$

To maximize material strength and minimize the highlighted issues with bonding, maximizing the score gives a good approximation of the necessary steps, and overall, it gives a good characterization of the bonding quality for a given simulation.

That finishes all numerical methods utilized with this model to simulate the FFF deposition process. All of these elements worked in tandem to produce an accurate recreation of the conditions and processes for the material. Next it is important to briefly treat time iteration and stability regions.

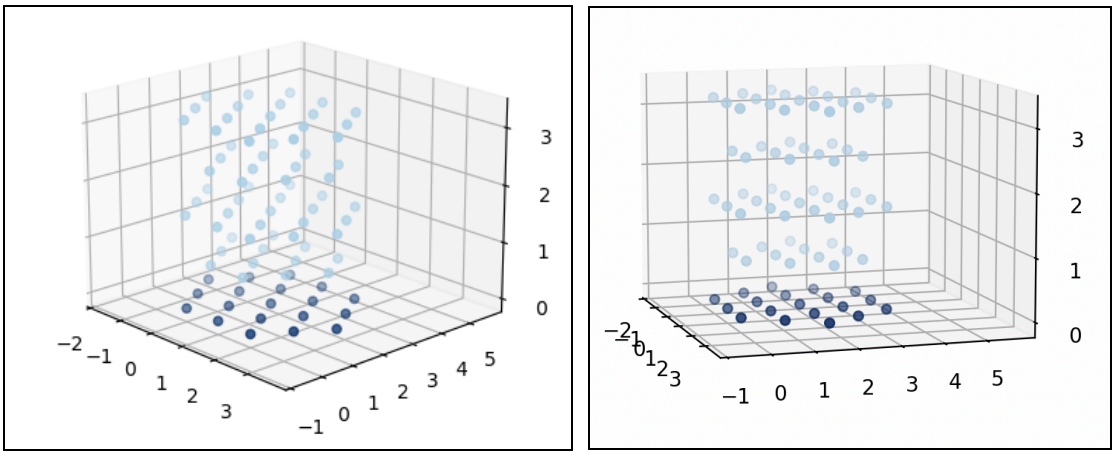
Time Iteration and Stability Regions

The stability of this model proved to be an essential struggle in producing results. The high powers of the Lennard Jones potential are very sharp and can easily send a solution into divergence. Through development of this model, thousands of node sets have exploded (literally) into divergence. Given more time, a complete look at stability regions and implementation of a superior time iteration algorithm (Runge Kutta 4, reverse Euler, etc.) would be immensely beneficial for quality of results and time to compute (another significant roadblock). Currently, due in part to the difficulty of literature on time iteration, stability, and related topics and due more mainly to lack of early consideration for stability as an issue, attempts at implementation have been unsuccessful.

Over the usage of the model, two methods have been explored as potential time iteration algorithms. The first is a simple Euler method, I.E. using a fixed time step. The second is an adaptive time step based on controlling the maximum observed force. It functions by recording the maximum force observed and setting the time step such that the impulse (force multiplied time) is beneath a set level. A similar time step set for velocity was considered but unused. These methods worked to the needs of the project but improving them in the future is a priority. Though a mix was used, curiously the Euler seemed more stable in its applications.

Model Shape

To do any meaningful simulation, the model must construct a matrix of points that are stable given the standard qualitative concepts. The first idea here was the use of a rectilinear grid which turned out to be unstable and prone to slight rotation and collapse, so a hexagonal matrix was implemented as it is much more stable for interactions based on distance. Figure 4 gives an example of the hexagonal grid as implemented into the simulation.



(Fig. 4)

While the setup is fascinating and included some moderately nontrivial geometry, it isn't incredibly pertinent to the actual modeling of the material. The build process is detailed in Appendix A if more information would be beneficial, but the important details are given here. The main principle is that all nodes are equidistant to their neighbors and unlike the rectilinear grid it requires significant force to break. Additionally, a hide feature was included to mask nodes that hadn't been printed yet, so given an extrusion rate, nodes could be gradually introduced into the model, simulating active printing.

Data and Implementation

Finding relevant data and implementing physically realistic material values gave this algorithm the actual realism and applicability needed. The material chosen for specialization and simulation is polylactic acid (PLA), a very common plastic for FFF AM. Although PLA was selected here, the model is flexible for any material for similar use. Here is summarized the data found for PLA and where it was utilized.

First, Thermal Diffusivity of PLA is $5.8244 \cdot 10^{-7} \frac{m^2}{s}$. This is from an industry material information sheet[10] and a formula given in a paper on AM Thermodynamics[11]. This is for implementation in the Heat Exchange calculations.

The Glass Transition Temperature of PLA is 68.5º C. This is from a paper on the general properties of PLA and its uses[12]. A wide spectrum for the values were reported, and they were almost always given as a range of potential values. It seems likely PLA T_g depends on the manufacturer or other sources. This is used for defining phase shifts and property changes.

The energy of the attraction (E) is difficult to pin down given the somewhat ambiguous grid. The value is calculated from the tensile strength. Using it, individual maximum force before separation between particles is found using $F = \sigma \cdot A$ as 70 N[13]. Thus, the F = $\frac{dV}{dr}$ is 70 N. Looking at the formula for F, the max force achieved is $2.689901 \cdot E$. Thus, if the desired max force is 70 N, E would be $\frac{70}{2.689901} J = 26.023 J$.

The viscosity of PLA is a bit difficult to find due to its general usage, so the shear modulus, which is easier to find, can be used. Since viscosity is given by shear modulus *G* times relaxation time λ, viscosity is roughly given by 1.287 MPa[13, 14]. While viscosity does change with temperature. This change was simply considered with the same factor as in the calculation of ε (Eq. 2-2).

The Thermal Expansion of PLA is $4.17 \cdot 10^{-4} K^{-1}$. There is also some flexibility on the specific definition of this constant likely again due to manufacturing differences. This particular value came from a recent paper investigating the thermal expansion of PLA with variable infill[15].

Some additional values used include plate temperature of 50 ºC, extrusion temperature of 110 ºC, and nodal diameter of 1 mm. The first two represent typical and common settings for those two. The last is a computational choice, resulting in a mass per node of 1.24 mg.

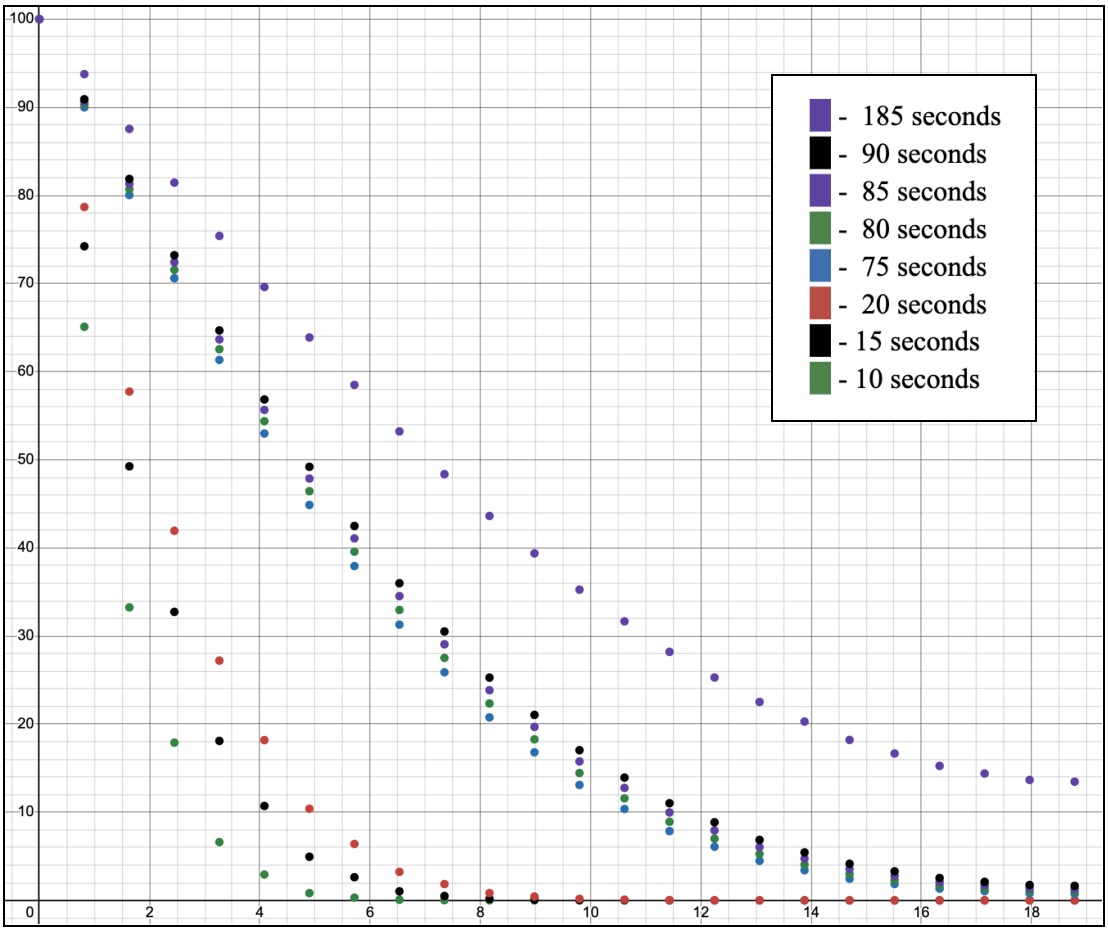
These data values bring this model to life with needed realism. Through the effective use of these values, the model is able to simulate FFF and characterize its bonding.

Verification and Validation ----------

The Verification and Validation of this model is crucial to ensure its effectiveness as a means for prediction and generalization. To verify the components of the simulation highlighted in Methodology and Algorithms, individual test simulations show effectiveness of each component in its isolated form. The simulation for thermal energy exchange is a textbook example of heat conduction. As for validation, two general cases, one at high temperature and one at low will illustrate the effective modelling of these different characteristics.

Heat Transfer

Starting off with heat, the popular problem Heat Conduction into an Semi-Infinite Wall will serve as an example. For this simulation all physical interactions are turned off to standardize distance and reduce computational complexity for faster run times. A hexagon of side length 3 particles with a height of 24 levels served as the basis for the simulation. The first level was fixed at a high temperature (called T_s) and the rest of the material was started at a low temperature (called T_0). The temperatures of the other particles were then recorded at periodic intervals to gain an idea of the temperature flow as modelled. Given these parameters, two limitations may be identified. The first being area in the x and y directions. Limited area in these directions deviates from the theoretical ideal by reducing diagonal heat transfer. Closer to the heating surface however, this difference is less noticeable. Thus, more heed will be paid to nodes close to the plate. The second limitation is the limited height of the model, thus eliminating temperature diffusion to infinity. This means that late times when the heat has penetrated the whole solid will be less accurate. Running the model, the following results are returned. Figure 5 shows the average observed temperatures for each of the 24 levels of the solid (with x being the height of the level with the distance between levels being $\sqrt{\frac{2}{3}}$) at times (from the bottom up) 10, 15, 20, 75, 80, 85, 90, and 185 seconds.



(Fig. 5)

Here can be seen that those limitations did cause the later steps to not exhibit the proper qualitative features. For example, the temperature curve for 185s can be seen to not decay properly to 0 due to the unrealistic stop to the model at 24 levels. This can also be seen (but to a much lesser extent) for the four middle timeframe curves.

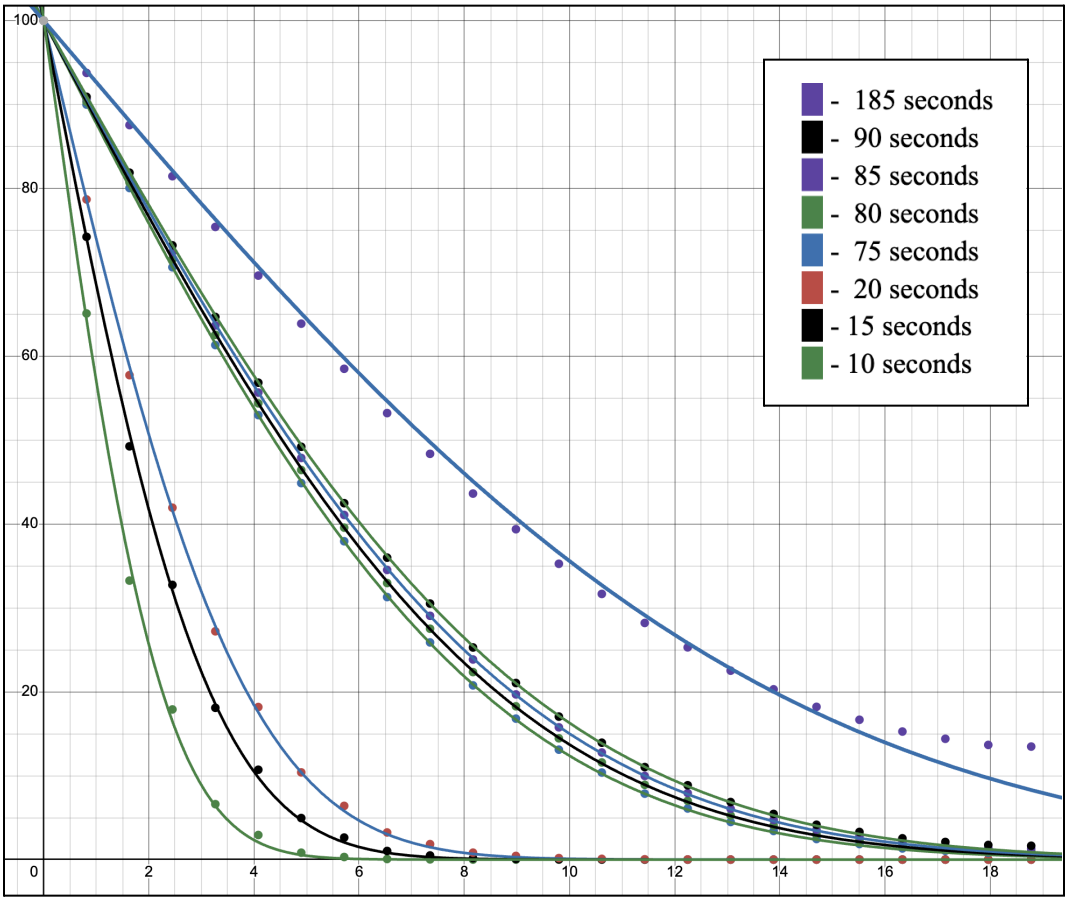
To get an idea of how accurate the model is to the theory, a solution curve was mapped to the data points to minimize error given malleable thermal diffusivity, using desmos. Equation 5-1 gives the solution to Heat Conduction into an Semi-Infinite Wall as set out by the physics textbook *Fundamentals of Momentum, Heat, and Mass Transfer*[16]. x gives distance in. t gives time. α gives thermal diffusivity. T is the temperature being solved for. *erf* is the error function.

$$\frac{T-T_0}{T_s-T_0} = 1 - erf\left(\frac{x}{2\sqrt{\alpha t}}\right) \qquad (5\text{-}1)$$

Solving for T (Eq. 5-2) gives the function that can be mapped to the results data set.

$$T = \left(T_s - T_0\right)\left(1 - erf\left(\frac{x}{2\sqrt{\alpha t}}\right)\right) + T_0 \qquad (5\text{-}2)$$

Using desmos to map this function to the data, the following curves are obtained (Fig. 6).



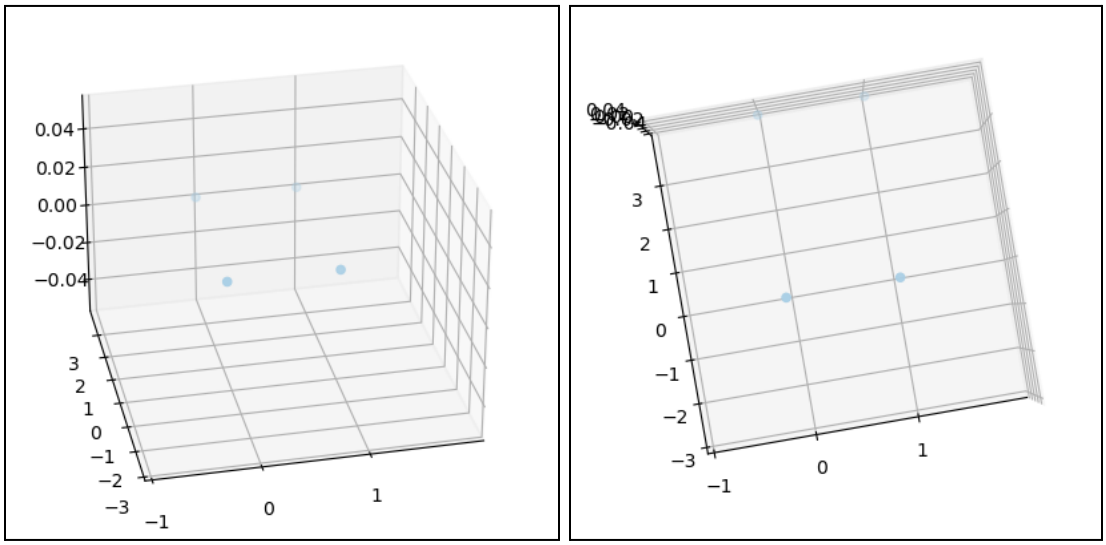
(Fig. 6)

The curves mapped almost all had R^2 values of 1, though curves near the beginning and end of the timeframe had slightly larger errors. Color unfortunately confuses curves of best fit; the legend is still for the points. Curves in the region described by the limitations exhibited excellent adherence to the actual thermal diffusivity number. The black curve for 15 seconds in the lower three curves is within 1.33% of the actual thermal diffusivity. Curves beyond overestimate the thermal diffusivity, and curves before underestimate it. This is likely due mostly to inaccuracy in

the approximation of the second derivative for the implementation of heat transfer; however, previously described limitations on the scenario also affect the error observed. Overall in the broad and generally-unextreme context of the 3D print environment, the heat transfer mechanism and the 2nd derivative approximation are very accurate and provide a precise input for mechanical changes.

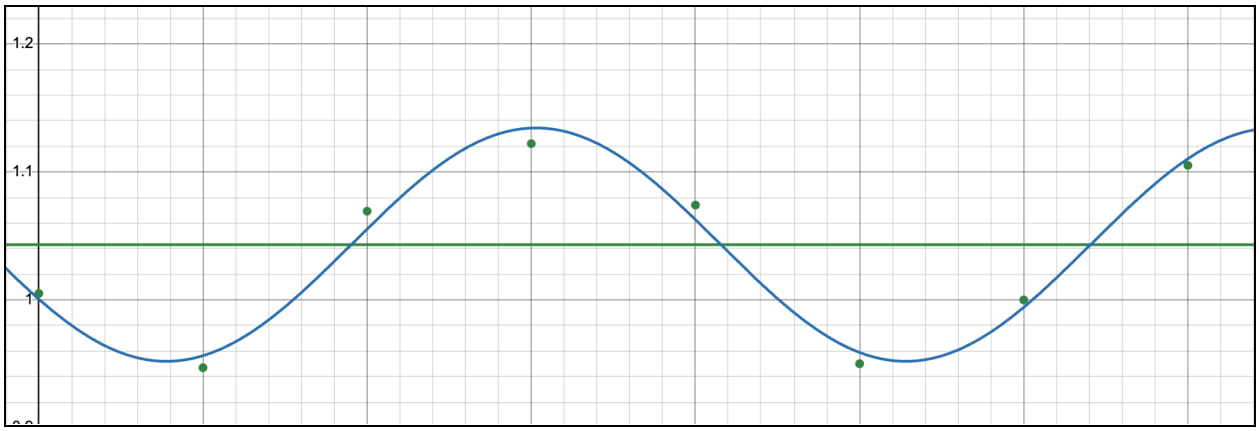
Simple Nodal Interaction

Next, for simple nodal interaction, two particles are placed at 1 mm apart and a separation force is applied to them of 60N, and they should stay together, having their bond stretch accordingly. Running the simulation, the following results are obtained for .5 seconds (Fig. 7). The two particles in the background are for visual distance reference. Saved data was used for accuracy calculations.



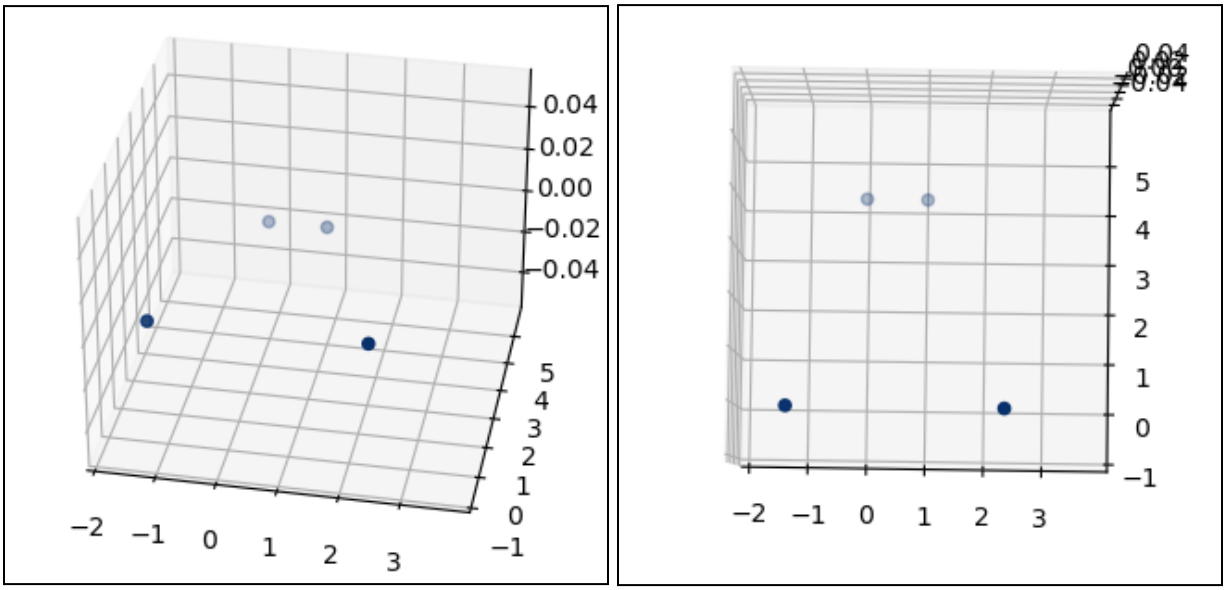
(Fig. 7)

The nodes oscillated in and out due to the initial energy given, but their bond length was easily fit to a sine wave oscillating around 1.04314 mm. That is within 1.3% of the formula ideal. See Figure 8 for the sine wave and plotted points. The x axis is time, and the y axis is bond length.



(Fig. 8)

Overall it is evident that the attractive potential is well implemented and calibrated. To illustrate, here is another simulation where the force between these particles is 80N. This simulation was completed over .5 seconds. See Figure 9.

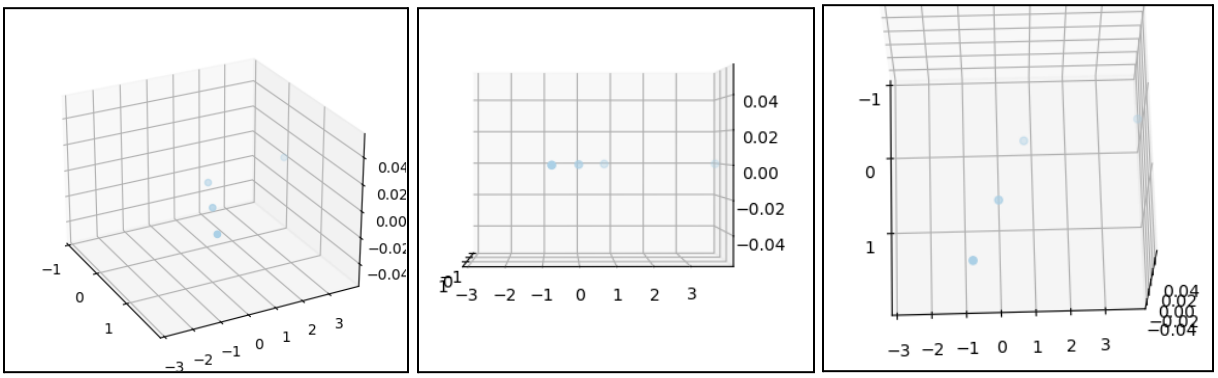


(Fig. 9)

It simulates as intended. The nodes detach and separate.

Complex Nodal Interaction

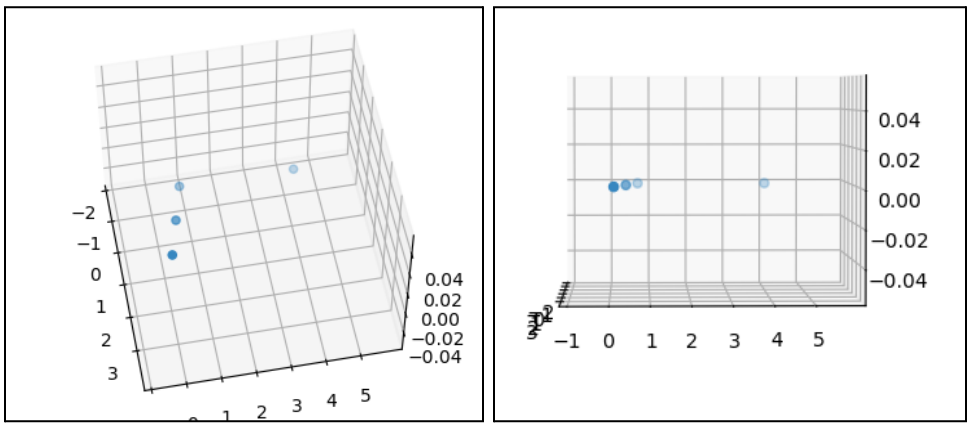
The next important aspect to test is torque and friction. To do this, three particles are created and placed one millimeter apart in a line on x = 0. The middle particle is fixed positionally (i.e. cannot move but can accumulate angular momentum).Then, one of the outer particles is given a velocity of 1 mm/s. Doing such, the following results are obtained for 1s of simulation (Fig 10).



(Fig. 10)

Take note of the backwards motion of the front particle. This is exactly what should happen. This shows the successful effect of the torque and friction within the model and demonstrates the complex interactions under the surface. Despite the simplicity of this simulation, it does a lot to give confidence to the often dense vector algebra calculations that go into its definition.

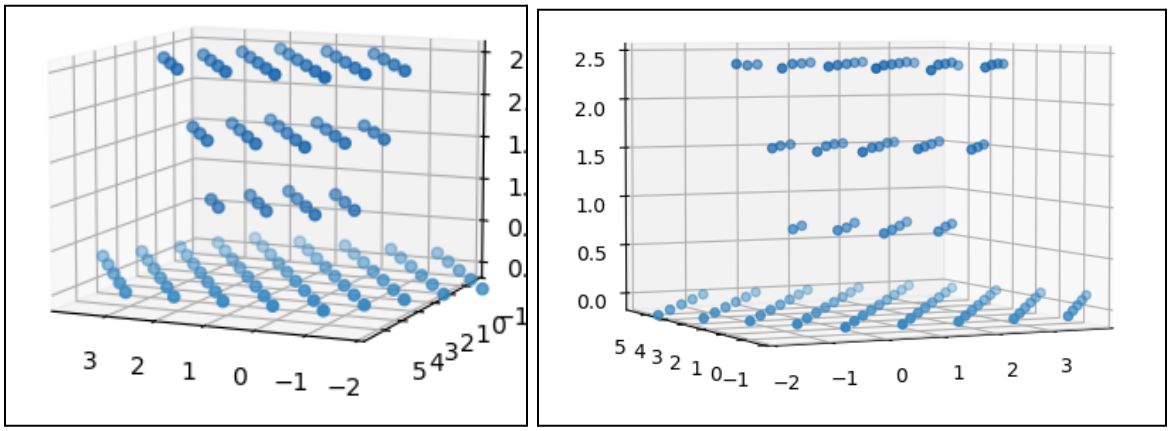
Repeating the test without fixing the position of the middle particle, a full demonstration of friction can be seen. This simulation also had that initial velocity of 1 mm/s and it was run for 1 seconds. Figure 11 shows the results.



(Fig. 11)

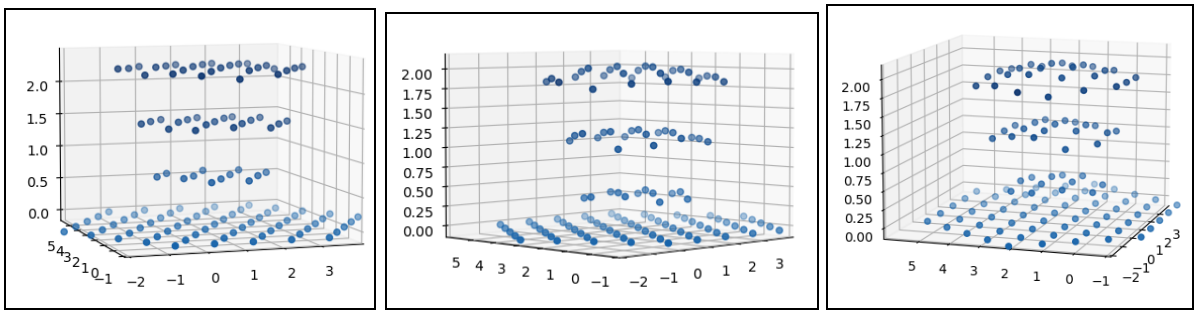
These show how the particles are dragged along as a result of friction. It also shows how the particle doesn't travel the 1mm that it would if it were unimpeded, giving the effectiveness of the friction algorithm.

Now that the effectiveness of the individual components has been explored. Two Validation simulations show how full usage of these algorithms yields correct characteristics. The first of the two simulations is of a low temperature solid. The model shows that it does not move and maintains its shape like a solid. This was run for plastic at 60º C (a bit below the Glass Transition Temperature). Some modifications were needed for this special introduction of solid particles immediately, as calculations for solidification occur after one step of normal consideration (Eq. 2-2 can give extreme values for $T<T_g$). It was run over 2 seconds (small time frame due to computational complexity). Figure 12 gives the results graph on the right and the starting graph (for temperature).



(Fig. 12)

As seen, the model holds its shape successfully. Now, for the second simulation, the material was put at a high temperature and modeled to show liquid characteristics. The temperature was set to 150º C, and it ran over 2 seconds. Figure 13 shows several frames of the results at times 0.75, 1.5, and 2 seconds (left to right). The starting configuration is unshown as it is the same as the previous.



(Fig. 13)

It can be seen clearly that the model successfully models the liquid characteristics of the flow as well. As a qualitative check, a look at the spread of the flow and the actual simulation proportions gives more information about the flow. The actual model is about 6 mm by 6 mm at the start, and it can be seen that the flow spreads from this normally. Given that the surface tension of PLA is low but not infinitesimal at this temperature, it makes sense that the drop (really that's what it is), spreads out a bit but not flowing off the plate yet at 2 seconds in. Longer time wasn't feasible due to long calculation times.

Overall, through these individual verification simulations and these two broader validation simulations, it is clear that the model accurately and effectively models the material in the printing environment.

Results: Analysis and Conclusions ----------

A few experimental simulations were conducted to analyze the effect of various temperature inputs on the quality of bonding and the general conformity to design in the simulation. These simulations showed two major changes to typical practices that would better support model strength and shape.

The first of which is supported by three simulations with varying plate temperatures. Those plate temperatures were 40º C, 60º C, and 65º C. The foremost was to gauge the effects of reducing plate temperature. These simulations yielded significantly better bonding grades for temperatures that are within 10º C of the glass transition temperature. Of course, for plate temperatures too close to the glass transition temperature, slow deformation caused significant structural changes from the design. This, however, was not an issue for temperatures from 5º C

to 10º C lower than the glass transition temperature. Bonding grades were generally 10 to 15 points higher for these high plate temperature simulations. This is supported generally by the concept that actual shifting and movement of the material is very contained and minute. The high viscosity, high tensile strength, and constant solidifying leads the material to be resilient to the small force of gravity on it. This makes focus of proper temperature for bonding significantly more important than keeping overall temperatures low to maintain rigidity. Higher constant temperatures also ensures that the model cools to room temperature uniformly, which, in fact, lowers overall deviance from the design and other physical changes. The usage of higher plate temperatures ultimately supports the effective bonding and the careful adherence to the original shape.

The second major change is more general and related to the laying of material. Due to the often long waits between when filament is applied to an area and when more filament is added to it. Temperature differences can be shocking, leading to poorer bonding and warping and physical changes. To better maintain constant bonding conditions for surfaces to be added to, heating from the top of the model would be beneficial. Sharp temperature changes from very hot added filament to the easily-cooled surroundings nodes causes issues.

In two simulations, changing extrusion temperature gave light to the importance of control for the adding surface. While actually changing the extrusion temperature would be disastrous as the filament would no longer melt and bond properly, this gives light to changing steep temperature changes near that surface. Using lower extrusion temperature, the temperature changes at the adding surface smoothed out and bonding grades went up. To smooth out surface temperatures and provide for better bonding, ventilation at a set temperature would allow finer control. This prevents steep temperature changes, which limit effective bonding and set the stage for unequal thermal contraction. The two simulations were at extrusion temperatures of 95º C and 80º C. By reducing the extremity of the temperature on the bonding side, the bonding grades were observed to go up around 5 to 10 points. It's important to note that this seen change in bonding temperature could reasonably be attributed to a general lowering of the temperature for the model. However, it can be known that this is not the case by considering the previous simulations, showing that raising the overall temperature results in higher bonding grades, so the increase seen is directly from the evening of temperature on the adding side.

These two changes, as seen, benefit the overall strength and adherence to the desired design. Their implementation could improve the effectiveness of FFF processes for myriad applications.

Future Use: Development and Testing -------

The flexibility of this algorithm is key to its importance overall. Using properties from any wide variety of materials, different materials can be modeled without drastic change. This allows this same sort of thermal optimization for any number of applications. Additionally, use on more sophisticated computers with more refined algorithms could yield a significantly more nuanced understanding and, more so, understanding to optimize physical properties.

As for further development of this model specifically, usage of superior time iteration methods, refinement of grid size, and improvement of approximations in second derivatives are priorities for future improvement. Usage of Runge-Kutta or Reverse Euler algorithms for time iteration would significantly improve computation time and likely eliminate bad simulations where a large amount of time is used calculating a simulation only to have it diverge. Refining the grid to smaller particles would make the simulation more accurate, but it would also make the simulation more computationally costly. In the long term, increasing computational cost to increase accuracy is a positive exchange. Lastly, improving approximations in the second derivative by, for example, including more nodes in the approximation or turning the temperature into a field, unbound to nodes, would reduce overall error.

The greatest accomplishment of this process was to model the drag (force from surrounding flow: characterization of friction), torque, and angular momentum of the particles. This utilized difficult implementation of vector algebra and physics, and its inclusion represented an oft unconsidered element of material simulations.

Bibliography and Acknowledgements -------

1 - “Additive Manufacturing Market Size Report, 2030.” n.d. Www.grandviewresearch.com. https://www.grandviewresearch.com/industry-analysis/additive-manufacturing-market.

2 - Pérez, Mercedes, Diego Carou, Eva María Rubio, and Roberto Teti. 2020. “Current Advances in Additive Manufacturing.” *Procedia CIRP* 88 (January): 439–44. https://doi.org/10.1016/j.procir.2020.05.076.

3 - Fitzgerald, Diana. 2022. “NASA’s New Material Built to Withstand Extreme Conditions - NASA.” NASA. April 12, 2022. https://www.nasa.gov/aeronautics/nasas-new-material-built-to-withstand-extreme-conditions/.

4 - Thomas, Douglas S., and Stanley W. Gilbert. “Costs and Cost Effectiveness of Additive Manufacturing.” *Costs and Cost Effectiveness of Additive Manufacturing*, no. 1176, Dec. 2014, nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1176.pdf, https://doi.org/10.6028/nist.sp.1176.

5 - Javaid, Mohd, et al. “Role of Additive Manufacturing Applications towards Environmental Sustainability.” *Advanced Industrial and Engineering Polymer Research*, vol. 4, no. 4, Aug. 2021, pp. 312–322, www.sciencedirect.com/science/article/pii/S254250482100049X, https://doi.org/10.1016/j.aiepr.2021.07.005.

6 - Gopal, Raghvendra. “What Is a Glass Transition Temperature? - Definition from Corrosionpedia.” *Corrosionpedia*, Corrosionpedia, 2019, www.corrosionpedia.com/definition/593/glass-transition-temperature-tg.

7 - “A Comprehensive Overview of Injection Molding Cooling Time - Zhongde.” *Www.zdcpu.com*, 30 June 2023, www.zdcpu.com/knowledge-hub/injection-molding-cooling-time/.

8 - Li, Yue, et al. “Analysis of Warping Defect Formation Mechanisms in Hot Molding of CF/PEEK Thin-Wall Structures and Their Influence on Mechanical Properties.” *Thin-Walled Structures*, vol. 207, 24 Nov. 2024, p. 112740, www.sciencedirect.com/science/article/abs/pii/S0263823124011807, https://doi.org/10.1016/j.tws.2024.112740.

9 - S. Fyfitch. *Corrosion and Stress Corrosion Cracking of Ni-Base Alloys*. 1 Jan. 2012, pp. 69–92, https://doi.org/10.1016/b978-0-08-056033-5.00079-3.

10 - *PLA Technical Data Sheet (Polylactic Acid)*. https://www.seas3d.com/MaterialTDS-PLA.pdf

11 - Blanco, Ignazio, et al. "Specific Heat Capacity and Thermal Conductivity Measurements of PLA-Based 3D-Printed Parts with Milled Carbon Fiber Reinforcement." *Entropy*, vol. 24, no. 5, 6 May 2022, p. 654, https://doi.org/10.3390/e24050654.

12 - Khouri, Nadia G, et al. "Polylactic Acid (PLA): Properties, Synthesis, and Biomedical Applications - a Review of the Literature." *Journal of Molecular Structure (Print)*, vol. 1309, 1 Apr. 2024, pp. 138243–138243, https://doi.org/10.1016/j.molstruc.2024.138243.

13 - Shivraj Yeole. "Tensile Testing and Evaluation of 3D Printed PLA Specimens as per ASTM D638 Type-IV Standard." *3rd International Conference on Innovative Design and Development Practices in Aerospace and Automotive Engineering (IDAD 2018)*, 23 Feb. 2018, www.researchgate.net/publication/323726339_Tensile_Testing_and_Evaluation_of_3D_Printed_PLA_Specimens_as_per_ASTM_D638_Type-IV_Standard.

14 - Yi-Sheng Jhao, et al. "A Mechanical Model for Stress Relaxation of Polylactic Acid/Thermoplastic Polyurethane Blends." *Journal of Composites Science*, vol. 8, no. 5, 1 May 2024, pp. 169–169, www.mdpi.com/2504-477X/8/5/169, https://doi.org/10.3390/jcs8050169.

15 - Botean, Adrian - Ioan. "Thermal Expansion Coefficient Determination of Polylactic Acid Using Digital Image Correlation." *E3S Web of Conferences*, vol. 32, 2018, p. 01007, https://doi.org/10.1051/e3sconf/20183201007.

16 - Welty, James R, et al. *Fundamentals of Momentum, Heat, and Mass Transfer*. John Wiley & Sons, 1976.

Benenson, Walter, et al. *Handbook of Physics*. New York, Springer, 2006.

Hesthaven, Jan S, et al. *Spectral Methods for Time-Dependent Problems*. Cambridge University Press, 11 Jan. 2007.

John Charles Butcher. *Numerical Methods for Ordinary Differential Equations*. Chichester, West Sussex, Wiley, 2016.

Acknowledgements

This project would not have been possible without the help and support of my parents and teachers. My mom, a chemical engineer and fluid dynamics researcher, helped me immensely to figure out the relevant material properties and general concepts. She also helped me revise this paper. My dad, also a chemical engineer with a background in computer science, helped me a lot to think through and consider the problem from a computational perspective. My math teacher Dr. David Metzler helped me figure out my errors when some of the more obscure vector algebra wasn't going right. Mr. Jay Garcia was quite helpful in planning out the process and checking in with me periodically. Mr. Alex Benedict, computer science teacher, also helped me quite a bit and recommended some books to help me understand the more complicated computational theory. I would like to thank all of these amazing people for their help and guidance.

Appendix A - Building the Hexagonal Grid -

To build the stable and effective grid for this simulation a hexagonal grid of many equilateral triangles was used. These equilateral triangles had side length one, and the nodes were placed at each vertex. This was pretty simple for the first hexagon. Given some side length n, the first row has n particles of spacing one, the next row has n+1 particles with an offset of ($\frac{-1}{2}, \frac{\sqrt{3}}{2}$) times rows minus one. That offset simply comes from looking at the height and half the base of an equilateral triangle, literally taking the side and turning it into a vector. This holds until there are the same number of rows as there are nodes per side. Then the offset is given as

(1-n+row/2, $\frac{\sqrt{3}}{2}$ row). This comes from carefully considering the definition of this 'line,' as it continues the other. Et voila, the first level is done.

Using some more geometry to make an equilateral triangular prism, the z change between levels is $\sqrt{\frac{3}{2}}$. There is also a new offset which is the vector to the center of the triangle: ($\frac{1}{2}$,$\frac{1}{2\sqrt{3}}$) or (0,$\frac{1}{\sqrt{3}}$) depending on the orientation of your triangle. The initial choice gives two rotationally equivalent second levels. Alternating between these gives proper progression inward or outward (adding or subtracting them). The node distribution on these layers is a little more complicated since it's not a perfect hexagon, skewing definitions on when to start moving the X offset inward and when to end. These two formulas give the number of nodes per level by hexagonal and in between respectively: $3n^2 - 3n + 1,\ 3n^2 - 6n + 3$. n changes from a in between section moving inward. Those formulas can be derived from simple series sum ideas.

The hexagonal matrix overall made the model more stable and more effective at predicting the motion and structure of the solid.

Investigating Intersubjective Realities From Novel NLP and Chaos Theory Approach

Camila Carreon

Santa Fe Preparatory School

Santa Fe, NM

April 2025

Contents

1 Executive Summary **2**

2 Statement of the Problem **3**

3 Introduction **3**

4 Background **5**

4.1 Intersubjective Realities 5

4.2 Chaos Theory as a Novel Way to Understand at Linguistic Intersubjective Realities in the time of Social Media 8

4.3 Natural Language Processing and State of the Field 10

4.3.1 State of the Field 11

4.3.2 Topic Modeling and Latent Dirichlet Allocation (LDA) 11

4.3.3 Preprocessing Techniques 13

5 Methodology **14**

5.1 Preprocessing and Data Collection 14

5.1.1 Data 14

5.2 Natural Language Processing 17

5.3 Engagement Analysis over Time 18

5.4 Recurrence Network Analysis 18

5.5 Symbolic Dynamic Analysis 20

5.6 Social Network Analysis 21

5.6.1 Network 21

5.6.2 Network Analysis 21

5.7 Veracity Analysis 22

6 Discussion **22**

6.1 Conclusion 22

6.2 Qualitative Contextualization 24

7 Achievements **24**

8 Acknowledgments **25**

9 Data Availability **25**

1 Executive Summary

When does information become important, and how do sentiments gain traction and turn into beliefs, collective and revisionist histories, principles, and ideologies? As wars of human rights and contrasting beliefs and values rage across the Earth, such questions are incredibly important. While the scope of my research is incapable of ending these global conflicts, I start at a smaller scale, analyzing social media data and discourse during the Covid-19 Pandemic taking language as the currency of information to analyze stability and structure. Specifically, I use Natural Language Processing and techniques borrowed from the mathematical field of Chaos Theory to explore what, as historian and scholar Yuval Noah Harari defines as "intersubjective realities," or a "shared, mutual understanding between individuals."[Har24] Ultimately, these realities are the hotbed of shared beliefs, stories, and, at an enlarged scale, ideology, and thus important. Furthermore, this research acts as a form of counterterrorism, looking for patterns in language and structure within these intersubjective realities to assess their potential for becoming influential and dangerous (becoming conspiracy theories).

2 Statement of the Problem

The goals of my research are twofold, first to investigate an issue that is often left out of the quantitative limelight: understanding narrative development and structure, and second addressing rising concerns of misinformed rumors and conspiracy theories on social media platforms like X (Twitter) and Reddit. A time period particularly ripe for such exploration is the Covid-19 pandemic, which in an immense time of uncertainty, as Francesco Farinelli of the Radicalisation Awareness Network (RAN) reports, "Conspiratorial narratives flourish[ed] in such a context and extremist groups exploited the spread of the coronavirus to disseminate fake news and to incite violence."[Far21] We define conspiratorial and uncharged yet popular narratives around the Covid-19 pandemic to be intersubjective realities, which become real once many people believe in it or begin to act in compliance with these narratives, or as the quotation from the RAN suggests, extremist violence in anti-government, anti-establishment, anti-lockdown and anti-restriction protests or AGAAVE—Anti-Government, Anti-Authority Violent Extremism. By identifying the linguistic and structural patterns of conspiratorial narratives we may be able to predict when and how they spiral into harmful ideologies. In other words, this work is not just about studying misinformation—it's about recognizing when belief manipulation turns into a societal threat. The Counter-Terrorism Committee Executive Directorate (CTED) in a 2020 survey reports that, "69% of respondents stated that countering terrorism has become more challenging as a result of the pandemic"[Dir21]. Thus to address such concerns, my work aligns with modern counterterrorism efforts, which in the US as shown here "promote US National security by developing coordinated strategies and approaches to defeat terrorism abroad and secure the counterterrorism cooperation of international partners." My work will be one such strategy and approach.

3 Introduction

The COVID-19 pandemic was not just a biological crisis but also an informational epidemic, where rumors and conflicting narratives shaped public perception. These narratives are part of what scholar Yuval Noah Harari calls intersubjective realities—shared beliefs that exist only in the human mind but are given power through collective belief, or recognition by at least 2 or more people. Yet the

propagation of such narratives reached unprecedented heights, when millions stuck at home resorted to screens and leveraged digital technologies and platforms like social media to stay connected to the rest of the world while quarantined. For example, the amount of Facebook users went up to about 1.9 billion worldwide by the end of 2020, marking an 8.7% increase over 2019[Wil]. And other social media platforms saw similar surges in popularity. Thus to understand the nature of pandemic-related discourse I turn to social media data (Twitter or X), and specifically use the COVID-19 Rumors dataset [Che+21], focusing on how misinformation and competing narratives form and evolve over time. Using Natural Language Processing (NLP), we track how rumors behave within the framework of chaos theory. Unlike static ideological structures, digital discourse is inherently nonlinear, meaning that small fluctuations—such as a single viral tweet—can unpredictably alter the trajectory of public perception. Online discourse, especially on platforms like Twitter, exhibits complex, nonlinear dynamics, making it an ideal system to analyze through chaos theory. The chaotic spread of misinformation, the convergence and divergence of narratives, and their unpredictable shifts mirror the properties of dynamical systems, where small changes in initial conditions can lead to vastly different outcomes. By mapping pandemic-related sentiments in a phase space, we analyze how belief-driven discussions shift, stabilize, or fragment into new patterns—much like turbulent flows in physical systems. These patterns of belief formation are a manifestation of intersubjective realities, where shared ideas constructed through collective engagement shape the social fabric of digital communication. To quantify these shifts, we construct recurrence networks and use symbolic dynamics to assess how belief transitions (support, deny, neutral) emerge, stabilize, or spiral into chaos. Measuring these transitions through entropy helps us understand the unpredictability of discourse and the moments when narratives undergo sudden, irreversible transformations. This study is grounded in the Three V's framework (Lukić), which defines the chaotic nature of online information networks: These properties make digital discourse highly chaotic and sensitive to initial conditions, meaning that small perturbations can lead to significant, unpredictable shifts in dominant narratives. By bridging chaos theory with computational discourse analysis, this research provides a novel framework for understanding the spread of misinformation, uncovering the underlying patterns that govern belief formation in the digital age. Through time series modeling and network-based methods, we map the evolution of online narratives, shedding light on the turbulent

Table 1: Three V's

V	Description
Volume	The vast scale of pandemic-related tweets, reflecting the sheer magnitude of discourse.
Velocity	The rapid rate at which narratives emerge, spread, and transform in real time.
Variety	The diversity of content and sentiment, encompassing conflicting perspectives, emotions, and misinformation.

flow of information and its impact on public perception during the pandemic.

4 Background

4.1 Intersubjective Realities

The swift entrance of Gutenberg's Printing Press in 1440 galvanized the rest of the Afro-Eurasian world into an unprecedented era where information and knowledge became public currency. Information, as scholars often term it, was democratized. Quickly the masses embraced Gutenberg's 42-line Bibles and even flocked to other printed works. One such text that took the medieval world by storm in the 15th century was Malleus Maleficarum, or Hammer of Witches, written by Heinrich Kramer in 1485, and a witch-hunting guide that warned witches were part of a satanic-led campaign to destroy humanity. The books popularity reflected in its immediate impact, as the narrative sparked outrage and violent responses for many in fear of their safety. Preaching to enraptured Alpine peasants, the book extends from Exodus 22:18 that, "You shall not permit a sorceress to live."[Enc]

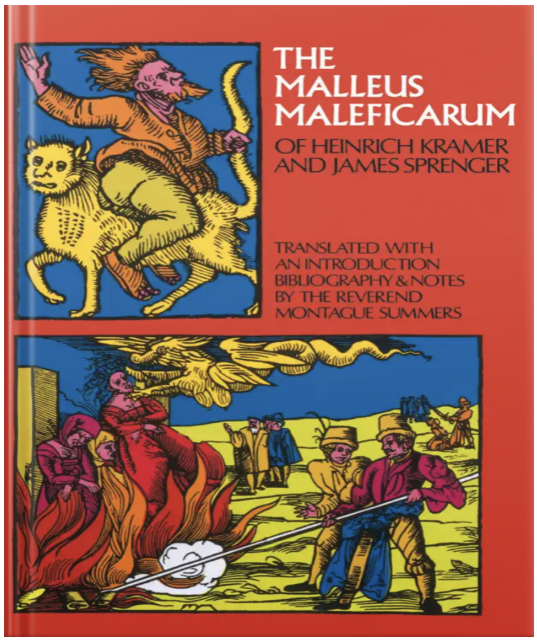


Figure 1: Malleus Maleficarum

The books dual biblical nature and call to legal action, turned credence to violence and mass hysteria. It is estimated that between 40,000 to 60,000 were killed after being tried and accused of witchcraft as a result. Yuval Noah Harari in his book Nexus: A Brief History of Information Networks from the Stone Age to AI considers the European witch craze an intersubjective reality, writing, "But witches became an intersubjective reality. Like money, witches were made real by exchanging information about witches."[Har24] Harari borrows from the philosophical lexicon of Edmund Husserl (1859 - 1938) who describes intersubjectivity as "the interchange of thoughts and feelings, both conscious and unconscious, between two persons or 'subjects,' as facilitated by empathy." [**knuthwebsite**]Husserl introduces intersubjectivity as part of a conceptual hierarchy of realities (objective, subjective and intersubjective) shown below:

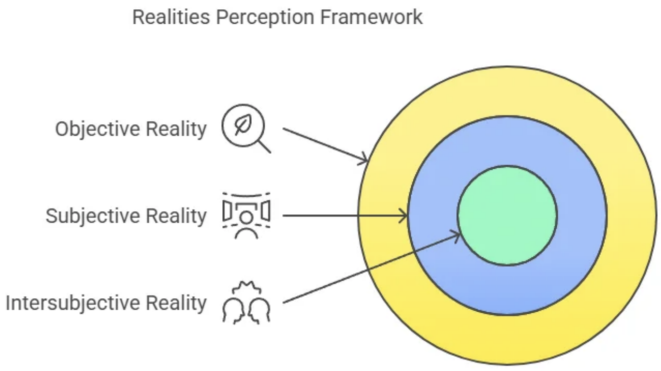


Figure 2: Realities Perception Framework

Harari injects this definition into a modern analysis of society, where he views intersubjectivity as shared fictional realities and social constructs upon which society depends like money, nations, and human rights, which only become real when enough people believe in them and have considerable influence. The European Witch Craze of the late 15th century becomes an intersubjective reality, when mass belief and hysteria overtakes much of Europe and results in extensive corporal loss.

Harari's thesis, as professor of law at Northeastern University Beth Simone Noveck puts it "is profound, albeit obvious: technology is inherently political, shaped by those who wield it, and often reinforces existing power structures."[Nov] In other words, the influence of intersubjective realities is also determined by who and how it is propagated, which as Harari illustrates have become increasingly less human. From the social media algorithms that largely dictate the flow of information in social media platforms and more recently AI, which "increasingly determines what we read about, think about, and talk about"[Nov], studying intersubjective realities in the digital age or age of information introduces a new layer into the research-algorithmic radicalization or the idea that as the Observer Research Foundation expands "Algorithms usually promote emotionally provocative or controversial material by focusing on metrics such as likes and shares, creating feedback loops that amplify polarising narratives."[Awa] For this project this means the structures of discourse and emergence of narratives we discover will be more pronounced than ever, and thus riper for analysis.

4.2 Chaos Theory as a Novel Way to Understand at Linguistic Intersubjective Realities in the time of Social Media

Chaos theory, often associated with physical systems, is a mathematical framework used to describe complex, dynamic systems that are highly sensitive to initial conditions. This sensitivity posits that small changes in the starting conditions of a system can lead to vastly different outcomes over time. The classic paradigm of these sensitivities is the culturally famous "Butterfly Effect" which imagines how the flap of a butterflies wings can cause a hurricane on the other side of the world. In the context of social media and online discourse, this concept is particularly relevant, as seemingly minor posts or viral tweets can trigger widespread shifts in public perception and belief systems.

In the digital age, platforms like Twitter, Facebook, and Reddit serve as a breeding ground for the formation of intersubjective realities—shared beliefs and narratives created through collective human engagement. As Yuval Noah Harari suggests, these intersubjective realities only become "real" when enough people collectively believe in them, regardless of their factual accuracy. The chaotic nature of online discourse, where rumors, misinformation, and competing narratives circulate rapidly, mirrors the principles of chaos theory.

The propagation of misinformation on social media is an inherently nonlinear process. A single viral post or tweet can escalate rapidly, becoming a global phenomenon, while other topics or narratives, despite similar initial traction, may dissipate into obscurity. Additionally, this process is buttressed by the advent of social media algorithms. Twitter's algorithm works by first "learn[ing] about users based on their clicks, likes, and responses. Then, it takes this information and turns it into outputs. In this situation, that information helps create the main 'For You' feed on the Twitter platform." [Tea]On March 31, 2023, Twitter became the first social media platform to release it's engagement formula which is shown in the figure below:

Type of engagement	Weight
Probability the user will like the tweet	0.5
Probability the user will retweet the tweet	1.0
Probability the user replies to the tweet	13.5
Probability the user opens the tweet author profile and likes or replies to a tweet	12.0
Probability (for a video tweet) that the user will watch at least half of the video	0.005
Probability the user replies to the tweet and this reply is engaged by the tweet author	75.0
Probability the user will click into the conversation of this tweet and reply or like a tweet	11.0
Probability the user will click into the conversation and stay there for at least 2 minutes	10.0
Probability the user will react negatively (requesting "show less often" on the tweet or author, block or mute the tweet author)	-74.0
Probability the user will click report tweet	-369.0

Figure 3: Twitter Engagement Formula

Chaotic systems, while they exhibit seemingly random and unpredictable behavior, are governed by deterministic laws. The butterfly and the hurricane both exist in a world governed by the same physical laws, and similarly the social networks on social media platforms are governed by the determinstic rules like the engagement formula above, but still characterized by unpredictability and randomness given it's sheer variety, velocity and volume as per Lukic et al.'s 3 V's Framework as shown in Table 1. This unpredictability in the evolution of online discussions can be modeled using chaotic systems, where the smallest changes can lead to divergent or turbulent outcomes.

Furthermore, by applying chaos theory to linguistic data, we aim to uncover patterns in the way rumors and misinformation spread through online platforms. Similar to the behavior of dynamical systems, these patterns of discourse exhibit critical points where narratives transition from stability to instability, where once solid beliefs splinter into competing factions or where a seemingly benign topic may spiral into a full-blown conspiracy theory. Using techniques from chaos theory, such as recurrence analysis and symbolic dynamics, this study explores how shifts in linguistic patterns and sentiment lead to the formation, consolidation, or collapse of intersubjective realities within the

digital realm.

In summary, chaos theory provides a valuable lens through which to understand the unpredictable and highly sensitive nature of belief systems on social media. It allows us to explore how small fluctuations in discourse—such as the introduction of a new rumor or a change in sentiment—can lead to large-scale shifts in the collective psyche, highlighting the fragile and volatile nature of digital ideologies.

4.3 Natural Language Processing and State of the Field

Natural Language Processing (NLP) is a subfield of artificial intelligence (AI) that focuses on the interaction between computers and human language. NLP involves designing algorithms and models that enable machines to understand, interpret, and generate human language in a meaningful way. The complexity of human language, with its nuances, slang, and context-dependent meanings, makes NLP a challenging area of AI. However, NLP techniques have made significant advancements in recent years, allowing computers to process and analyze vast amounts of textual data effectively.

In the context of my project, NLP provides the tools necessary to handle the large-scale textual data generated on social media platforms during the COVID-19 pandemic. Social media platforms like Twitter contain millions of posts, tweets, and comments that express varying beliefs, opinions, and narratives about the pandemic. NLP techniques allow for the extraction of relevant insights from this vast and unstructured data, enabling a deeper understanding of the complex dynamics of belief formation and the spread of misinformation.

The primary objective of this research is to analyze the evolution of beliefs, rumors, and competing narratives surrounding the COVID-19 pandemic. Given the massive volume of data, as suggested in the 3 V's framework, involved and the unstructured nature of textual content on social media, manual analysis would be impractical. Thus, by using NLP, we can automatically process, categorize, and analyze textual data to extract meaningful insights that would otherwise be difficult to uncover.

For example, with the help of NLP techniques, we can detect topics that are being discussed, track how those topics evolve over time, and observe how beliefs and ideologies form, stabilize, or fragment in response to external events. The ability to process large datasets with NLP makes it

possible to study the dynamics of public discourse at scale, providing insights into the spread of misinformation, the formation of intersubjective realities, and the relationship between language and belief systems.

4.3.1 State of the Field

Collectively these abilities of NLP, have brought it substantial popularity in research on conspiratorial narratives or other types of discourse. For instance Albladi et. al in their paper *Detection of Conspiracy vs. Critical Narratives and Their Elements using NLP*, use the BERT (Bidirectional Encoder Representations from Transformers) NLP model developed by Google, and the RoBERTa model which proposed in RoBERTa: A Robustly Optimized BERT Pretraining Approach by Yinhan Liu, Myle Ott to identify conspiracy theories[Ais24]. On the more analytical side of this kind of research, Haupt et.al in their paper, *Detecting nuance in conspiracy discourse: Advancing methods in infodemiology and communication science with machine learning and qualitative content coding*, use NLP and qualitative content coding to explore the 5G conspiracy popular during the Covid-19 Pandemic[**Haupt**]. While this is just a small snapshot into the breadth of scholarship on NLP in Thus I was given a wealth of avenues to use NLP in studying Twitter discourse and choose the following techniques and process.

4.3.2 Topic Modeling and Latent Dirichlet Allocation (LDA)

Topic modeling is an NLP technique used to discover the hidden thematic structure in a large collection of texts. The goal of topic modeling is to automatically identify the underlying topics that are present in a corpus of documents, without prior knowledge of what those topics might be. This is crucial for understanding the wide array of subjects being discussed on social media platforms, especially in a context as complex as the COVID-19 pandemic. Below is a graphic explaining the process of Topic Modeling:

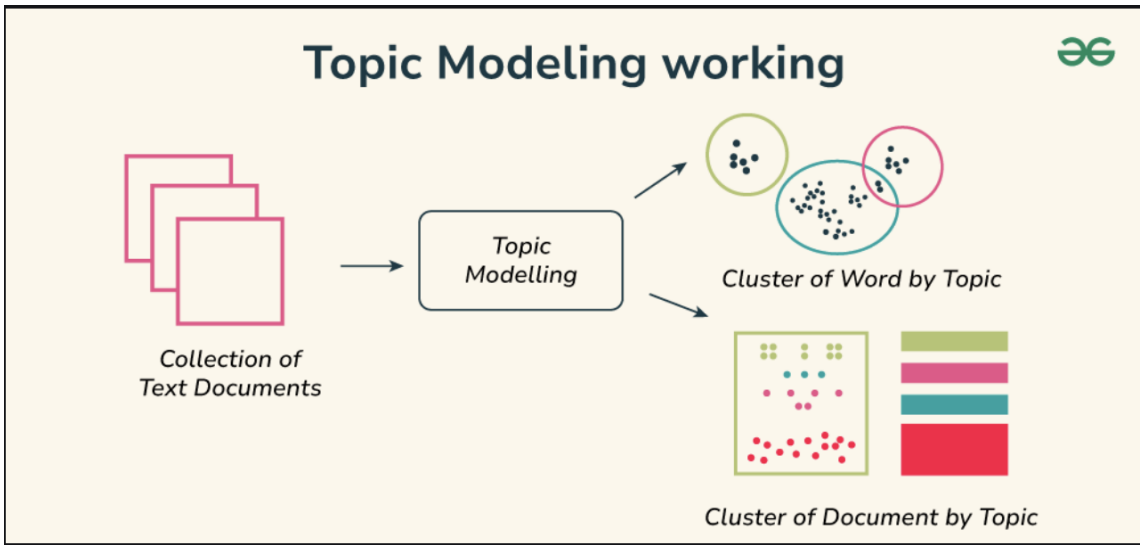


Figure 4: Topic Modeling

One of the most popular algorithms for topic modeling is Latent Dirichlet Allocation (LDA). LDA is a probabilistic model that assumes each document (in this case, a tweet or social media post) is a mixture of topics, and each topic is represented by a distribution over words. Furthermore, LDA generates topics by tracking frequency of co-occurrence (how likely is a word to appear with another word) and individual word frequency, using Gibbs equation to assign topics to words. Gibbs equation is shown below,

$$P(z_i = t \mid z^{-i}, w) = \frac{n_{m,t}^{-i} + \alpha}{\sum_{t'=1}^{T}(n_{m,t'}^{-i} + \alpha)} \times \frac{n_{t,w_i}^{-i} + \beta}{\sum_{v'=1}^{V}(n_{t,v'}^{-i} + \beta)}$$

Figure 5: Gibbs Equation

Where the first ratio is the probability of topic t in some corpus of text d, and the second ratio is the probability of the word w belonging to topic t. The model uses these distributions to identify

latent (hidden) topics in the corpus based on the co-occurrence of words. LDA assumes that there is a fixed number of topics in the collection and seeks to uncover the mixture of topics that best explains the observed word distributions in the data. [Jac]

In my research, LDA plays a critical role in identifying the various themes and narratives being discussed throughout the pandemic. By applying LDA to the COVID-19 Rumors dataset, I can extract a set of topics that represent key areas of discourse, such as "Government Response," "Health Measures," "Vaccines," and "Conspiracy Theories." Understanding these topics allows for a more structured analysis of how public perception evolves over time and how misinformation or competing narratives form and spread.

4.3.3 Preprocessing Techniques

Preprocessing is a critical step in any NLP pipeline. The raw text data collected from social media platforms is often noisy, inconsistent, and unstructured. Therefore, preprocessing techniques are applied to clean and prepare the text for further analysis. In the context of this project, the following preprocessing techniques were employed:

Tokenization: This process splits the text into individual words or smaller units (tokens), which are the basic building blocks for any subsequent analysis. For example, the sentence "Masks save lives" would be tokenized into ["Masks", "save", "lives"].

Lowercasing: To avoid distinguishing between words due to case sensitivity, all text was converted to lowercase. This ensures that words like "Mask" and "mask" are treated as the same word.

Removing Stopwords: Stopwords are common words (such as "the", "and", "is") that carry little meaning on their own and are often removed to reduce noise in the dataset. Removing stopwords helps to focus on the more meaningful words in the text.

Lemmatization: Lemmatization reduces words to their base or root form. For example, "running" is lemmatized to "run". This ensures that variations of a word are treated as the same entity, improving the consistency of the analysis.

Removing Non-Alphanumeric Characters: Social media posts often include special characters like punctuation, hashtags, and URLs. These were removed unless they were relevant to the topic modeling process (e.g., hashtags might indicate the topic of a tweet).

Stemming (if necessary): Although not applied in this project, stemming can also be used to reduce words to their stem form (e.g., "running" becomes "run"). However, lemmatization is generally preferred for its ability to handle words more intelligently.

The cleaned and preprocessed text data is then ready for analysis using NLP models, such as topic modeling or clustering algorithms. Effective preprocessing ensures that the data is structured in a way that allows for accurate insights into the dynamics of online discourse during the pandemic.

5 Methodology

5.1 Preprocessing and Data Collection

5.1.1 Data

Dataset Selection: The primary data source used in this study is the COVID-19 Rumors dataset [Che+21], which contains a rich collection of social media posts, primarily from Twitter, that discuss rumors and narratives related to the pandemic. The dataset includes both original posts and responses, accompanied by engagement metrics (likes, retweets, replies), making it ideal for analyzing social media discourse. There were 2,705 identified tweets and 34,847 retweets/comments associated with the posts. Additionally as shown in the images below the rumor dataset, twitter replies and posts had individual datasets and different accompanying metadata :

Twitter ID	release date	comment	time	replies	retweets	likes	timestamp	stance
1002962201143611	Sun Mar 29 2020	@nypost @JoshMan	Sun Mar 29 03:54:46	0	1	1	11:10.3	comment
1002962201143611	Sun Mar 29 2020	@nypost remember	Sun Mar 29 01:44:43	0	2	2	11:10.3	comment
1002962201143611	Sun Mar 29 2020	@nypost Hats off	Sun Mar 29 02:10:24	0	1	1	11:10.3	comment
1002962201143611	Sun Mar 29 2020	@nypost That's the	Sun Mar 29 01:43:38	0	1	1	11:10.3	comment
1002962201143611	Sun Mar 29 2020	@nypost lol, where a	Sun Mar 29 04:30:06	0	1	1	11:10.3	comment

Figure 6: Twitter Comments

label	content
F	If you can hold your breath without coughing, discomfort, stiffness, or tightness, your lungs do not suffer from fibrosis and therefore you have no COVID-19 infection
F	A homemade hand sanitizer made with Tito's Vodka can be used to fight the new coronavirus
F	Gargling with salt water or Vinegar 'eliminate' the COVID-19 coronavirus from the throat of an infected person's system
U	Patients should avoid taking ibuprofen to relieve pain and fever associated with COVID-19 infections
F	Chinese officials are seeking approval to start the mass killing of 20,000 people in order to stop the spread of new coronavirus

sentiment	reply numbers	retweet numbers	likes numbers
3	2	61	95
3	3	41	67
3	6	39	73
3	0	0	0
4	0	0	0
3	10	256	454

Figure 7: Twitter Posts

The column of metadata most important to me was the stance column. Stance as Cheng et.al define is, "The attitude of the author or editor of the rumor source. We follow classical rumor stance classification and define four classes of stance: support, deny, comment, and query. The stances are labeled and cross-validated manually by going through the context of each website."[Che+21] Other important pieces of metadata to my study are veracity, or labeling the posts/rumors as true(T) or false(F) or unverified(U), which were all manually labeled and cross-validated by referencing authoritative websites like Snopes , Politifact and Boomlive, and the reply, retweet and likes numbers for posts and comments. In their discussion of their dataset, Cheng et al. write,

> "We envision the downstream applications or usage cases of this dataset to include but not restricted to (i) the identification, prediction, classification of rumor, misinformation, disinformation, and fake news; (ii) the study of rumor spread trend and rumor/misinformation/disinformation/fake news combating and/or control; (iii) social network and complex network-related studies in terms of information flow and transition; and (iv) the natural language processingrelated studies of rumor sentiment and semantic."

[Che+21]

The scope of my research speaks to (ii) and (iii), as I am studying the development of rumors and narratives on twitter as a form of risk analysis.

Data Cleaning: In order to ensure the integrity of the analysis, several preprocessing steps were conducted:

Numeric Conversion: Engagement metrics, such as the number of likes, retweets, and replies, were converted into numeric values for easy analysis.

Text Cleaning: The text of each post was standardized by removing stopwords, correcting typographical errors, and eliminating irrelevant content such as URLs, special characters, and repeated symbols.

Topic Modeling: LDA was used to extract topics and classify each tweet into 10 different topics. Additionally I visualized topics with pyLDAvis. Below are the images of my python code for topic modeling using the Gensim library and the visualization outputted for Topic 0:

```
# Train LDA model
lda_model = gensim.models.LdaModel(corpus=corpus, id2word=id2word, num_topics=10,
    random_state=100)

# Extract topics
def get_lda_topics(model, num_topics, top_n_words):
    word_dict = {}
   for i in range(num_topics):
      word_dict[f"Topic # {i+1:02d}"] = [word[0] for word in model.show_topic(i, topn
    =top_n_words)]
  return pd.DataFrame(word_dict)

topics_df = get_lda_topics(lda_model, 10, 10)
print(topics_df)
# Function to assign the most likely topic to each tweet
def get_dominant_topic(text, id2word, lda_model):
  bow = id2word.doc2bow(text)  # Convert text to bag-of-words
  topic_probs = lda_model.get_document_topics(bow)  # Get topic probabilities
  return max(topic_probs, key=lambda x: x[1])[0] if topic_probs else 'Unknown'  #
    Return highest probability topic

# Assign topics to each tweet
tweets_df["topic"] = tweets_df["tokens"].apply(lambda x: get_dominant_topic(x,
    id2word, lda_model))
```

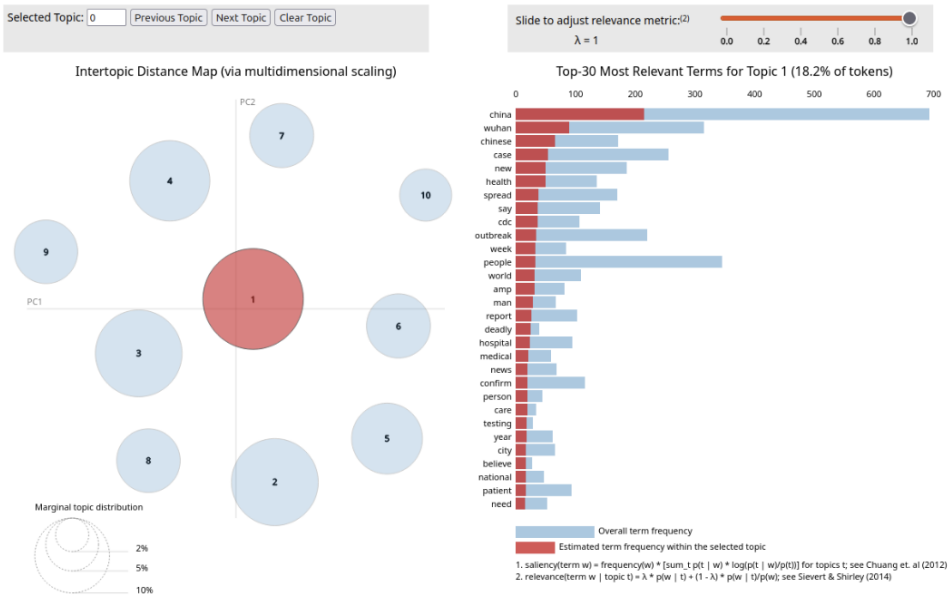


Figure 8: LDA Visualization for Topic 0

Topic Assignment: Topics for each post were assigned using TopicGPT from Pham et.al's *TopicGPT: A Prompt-based Topic Modeling Framework*. Using the generate topic lvl1 function I was able to generate high-level and generalizable topics. This allowed me to analyze how specific rumors and narratives evolved within certain thematic domains, such as government response, health measures, or public perceptions of the pandemic.[Pha+23]

Handling Missing Data: Missing or incomplete engagement data was handled by replacing NaN (Not a Number) values with zero, assuming no engagement occurred. This approach is standard for ensuring that incomplete posts don't disrupt the continuity of engagement analysis.

5.2 Natural Language Processing

Text Vectorization: To facilitate the computational analysis of text, I utilized NLP techniques such as tokenization, lemmatization, and vectorization. Each tweet was converted into a vector of word embeddings, using a pretrained word2vec model that encodes semantic relationships between words.

Stance Detection: The stance of each post was classified into one of three categories: Support, Deny, or Neutral. This classification was accomplished through a supervised machine learning model trained on labeled data. The model uses a variety of linguistic features, including sentiment polarity, word frequency, and topic-specific keywords, to assign a stance to each post.

5.3 Engagement Analysis over Time

Timestamp Processing: To analyze how discussions and engagement evolved over time, the timestamp of each tweet was extracted and standardized. Invalid timestamps were removed, and valid entries were aggregated by day and by topic, allowing for a temporal analysis of engagement metrics (likes, retweets, replies) for each topic.

Engagement Aggregation: Engagement metrics were aggregated at both the post and topic levels, considering interactions across the entire dataset. The number of retweets, replies, and likes for each post was summed daily, providing a time series of engagement for each topic. This step is critical for tracking shifts in public interest and engagement with different narratives over the course of the pandemic.

Stance Ratio Calculation: To examine the spread of conflicting beliefs, I calculated the ratio of support to denial stances for each topic. This ratio was smoothed using a 5-point rolling window to reduce the effect of outliers and to capture long-term trends in belief evolution. This metric gives insights into how narratives shift between support for and opposition to particular topics over time.

5.4 Recurrence Network Analysis

Distance Calculation: To construct a recurrence network, pairwise distances between smoothed stance ratios were computed using the pdist function from the scipy library. These distances were based on the Euclidean distance metric, which measures the similarity between two time points based on their stance ratios.

Recurrence Matrix Construction: A binary recurrence matrix was generated by applying a 10th percentile threshold to the distance values. This matrix indicates the recurrence of similar stance ratios, i.e., whether the discourse at a given time point is similar to that at another time point. This recurrence matrix serves as the foundation for network analysis.

Network Graph Creation: The recurrence matrix was converted into a network graph, where nodes represent time points, and edges represent the similarity (recurrence) between them. This network allows for the visualization of how discourse evolves over time and which time points are most similar to each other. Below is an image of the Network produced for topic 3:

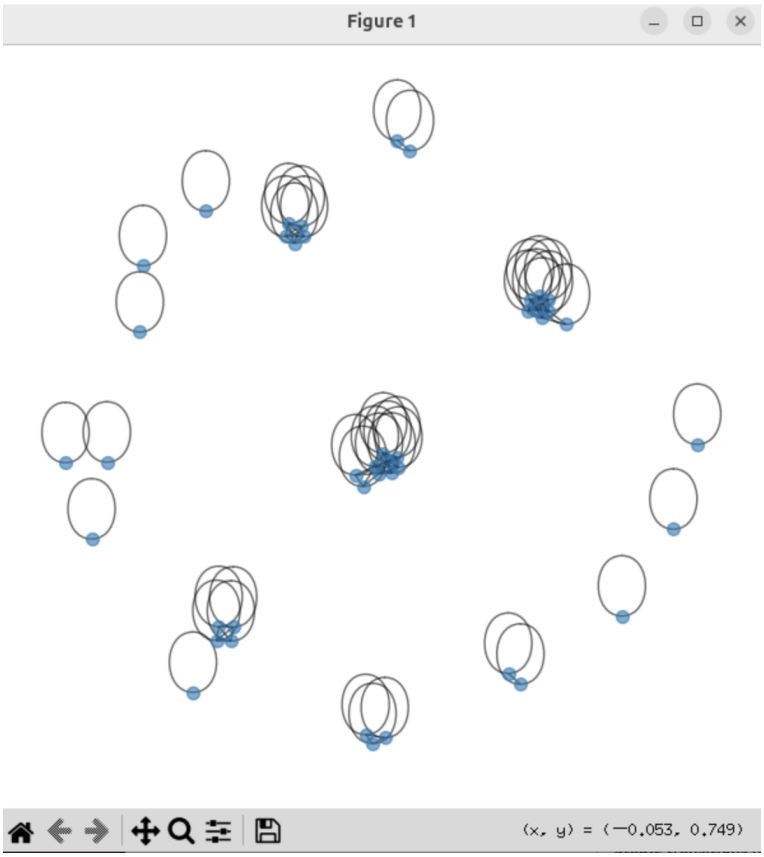


Figure 9: Network produced for Topic 3

Network Metrics Calculation: Several network metrics were calculated to assess the structure of the recurrence network. Specifically, the average clustering coefficient was computed to evaluate how interconnected neighboring nodes (time points) are. A higher clustering coefficient indicates a more tightly-knit network, which suggests that certain beliefs or narratives are more likely to spread within closed groups, fostering echo chambers.

5.5 Symbolic Dynamic Analysis

Discretization of Stance Data: To analyze stance transitions over time, the smoothed stance ratio was discretized into three symbolic categories: Deny, Neutral, and Support. This discretization was achieved by dividing the stance ratio data into quantiles, which allowed us to represent continuous stance changes as discrete states.

Transition Matrix Construction: The transitions between the symbolic states were encoded in a transition matrix, where each element represents the probability of transitioning from one state to another (e.g., from Deny to Support). The transition matrix was normalized to sum to 1 across each row, ensuring that the probabilities are valid. Below is an image of the transition matrix produced for topic 3:

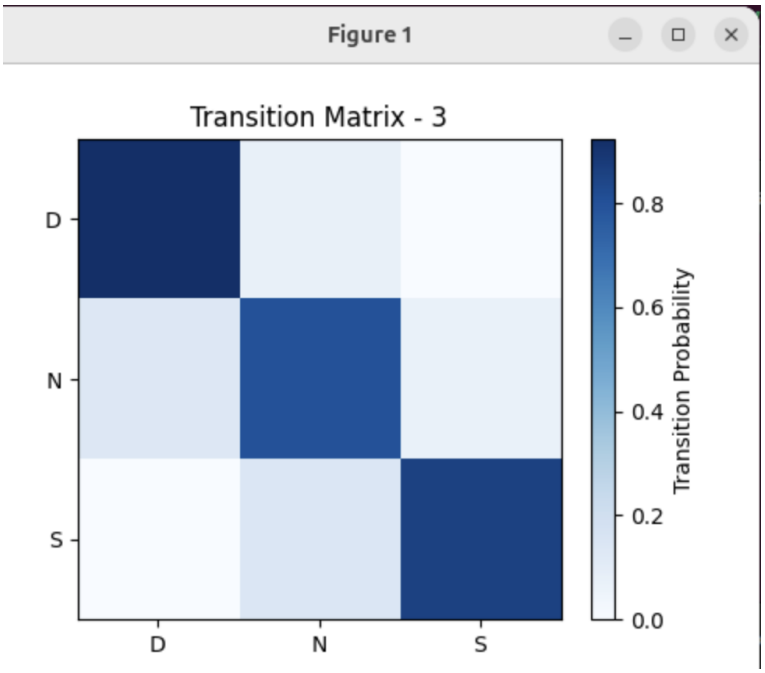


Figure 10: Transition Matrix for Topic 3

Entropy Calculation: To quantify the unpredictability of discourse transitions, I computed the entropy of the transition matrix. Entropy is a measure of randomness or disorder, and higher entropy values indicate more erratic and unpredictable shifts between stances over time. This measure

provides insight into how stable or unstable certain narratives are as they evolve. Below is an image of the calculations for entropy and clustering coefficient of transitions for topic 6:

```
Building recurrence network...
Average Clustering Coefficient: 0.7622
Performing symbolic dynamics analysis...
Transition Matrix:
          D         N         S
D  0.800000  0.133333  0.066667
N  0.217391  0.652174  0.130435
S  0.037037  0.185185  0.777778
Symbolic Dynamics Entropy: 3.0783
```

Figure 11: Calculations on Entropy and Clustering Coefficient for Topic 6

5.6 Social Network Analysis

5.6.1 Network

Network Construction For each of the ten topics, a directed network is constructed using pythons NetworkX library where nodes represent tweets (posts and comments), edges represent interactions between tweets, such as comments and retweets. Additionally, weights on edges correspond to engagement levels (sum of likes, replies, and retweets). Finally posts are assigned a node attribute (label) indicating their veracity, with values mapped as:

1 (True)

0 (Uncertain)

-1 (False)

5.6.2 Network Analysis

Furthermore, for each topic analysis is done using NetworkX's built in functions to calculate betweenness centrality, closeness centrality and degree centrality. The top 5 nodes (labeled with Twitter ID and respective measure of centrality) with the highest of each respective measure of centrality are shown. A description of each form of centrality analysis is featured in the table below and the result for all three forms of centrality analysis for topic 3 is pictured below:

Table 2: Three V's

Centrality Analysis	Description
Degree Centrality	Measures the number of direct connections a node has.
Closeness Centrality	Assesses how close a node is to all other nodes in the network.
Betweenness Centrality	Evaluates the extent to which a node lies on shortest paths between other nodes.

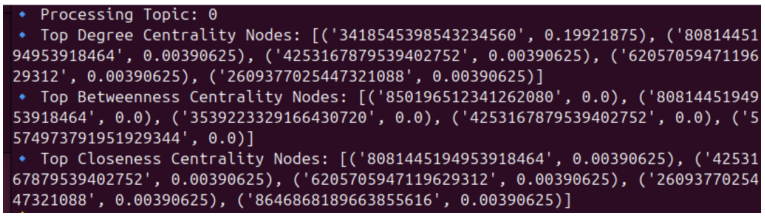


Figure 12: Calculations on Centrality for Topic 0

5.7 Veracity Analysis

Finally I associated each topic with a veracity score shown in the equation below:

$$V = \frac{T}{T + F} \tag{1}$$

Where V is the veracity score and T is the number of posts related to a topic with veracity labels of True (T) and F is the number of posts related to a topic with veracity labels of False (F).

6 Discussion

6.1 Conclusion

My analysis of conspiracy theory discussions on Twitter, leveraging recurrence networks and symbolic dynamics, reveals distinct structural and dynamical patterns in the spread of misinformation. I observed that topics with stronger local connectivity, such as Topic 8 (clustering coefficient = 0.7500) and Topic 6 (clustering coefficient = 0.7622), likely represent tightly-knit echo chambers

where individuals engage primarily within a closed community. These echo chambers may perpetuate misinformation by reinforcing existing beliefs. On the other hand, topics with more diffuse interactions, such as Topic 2 (clustering coefficient = 0.5474) and Topic 4 (clustering coefficient = 0.5714), may foster more open yet still biased discussions, facilitating the spread of conspiracy theories to a broader audience. The symbolic dynamics analysis, highlighting stance transitions, shows that denial (D) and support (S) are highly stable in most topics, contributing to the persistence of conspiratorial narratives. These states reinforce each other, while neutral (N) states exhibit more variability, suggesting that neutral stances play an intermediary role in discourse evolution. Topics with high entropy, like Topic 8 (entropy = 3.1357) and Topic 6 (entropy = 3.0783), show greater unpredictability and dynamic shifts in conversation, potentially fostering more fluid, yet still misinformed, discussions. Furthermore, the integration of veracity scores indicates a correlation between the structural properties of the network and the truthfulness of the discourse. Topics like Topic 3, which show the lowest veracity (0.1069) and high clustering, suggest that misinformation thrives in more rigid, tightly connected communities. Conversely, topics with higher veracity scores, such as Topic 5 (0.2238) and Topic 4 (0.2086), exhibit more openness in their discourse, indicating a greater potential for corrective discourse or less virulent misinformation spread. Additionally, in looking at centrality measures of the network of posts and comments, weighted by retweets and likes, I also looked at veracity. I calculated mean veracity for the top degree centrality, betweenness centrality and closeness centrality nodes. Topic 3 seems to show a neutral trend for degree centrality (mean veracity = 0.0), but betweenness and closeness centrality are slightly more negative, indicating that while many of the most connected nodes might not be spreading misinformation, those controlling the information flow or with closer connections are more likely to be involved in less reliable content.Topic 0 and Topic 9 show that higher degree centrality doesn't necessarily mean more reliable content, with negative mean veracity. Topic 6, on the other hand, shows higher mean veracity for closeness centrality nodes, which suggests that for this topic, nodes that are more central in the network tend to be more truthful.

Table 3: Labels of Topics

Topic	Manual Label
0	Public Health and Data
1	China's Response to Covid-19
2	Government and Official Response
3	Health Measures and Masks
4	Citizens, Infections, and Spread
5	Confirmed Cases and Deaths
6	Testing and Positive Cases
7	China's Fight Against the Outbreak
8	Medical Professionals and Time
9	Hygiene and Cure Efforts

6.2 Qualitative Contextualization

Contextualizing my conclusions within the manual labels assigned to each topic, particular attention should be given to topics 3 (Health Measures and Masks), 4 (Citizens, Infections, and Spread), 5 (Confirmed Cases and Deaths), and 6 (Testing and Positive Cases). My research suggests that Topic 3, with its low veracity and high clustering, is especially vulnerable to misinformation propagation. This aligns with the fact that mask-wearing became a highly polarizing issue during the Covid-19 pandemic, fueling divided beliefs. Conversely, topics related to pandemic statistics, such as Topics 4, 5, and 6, exhibited higher veracity, entropy, and more open structural dynamics, suggesting that discussions in these areas were more fluid and corrective, as factual claims and evidence are harder to dispute. Together, these findings highlight the complex dynamics of misinformation and its spread, demonstrating the chaotic nature of belief systems in the digital age. This research emphasizes the importance of understanding and addressing the forces shaping public discourse, with potential implications for improving communication strategies and combating misinformation in future crises

7 Achievements

My greatest achievement was learning how to perform NLP in python as well as be able to produce a great number of visualizations that succinctly and effectively presented my analysis.

8 Acknowledgments

I'd like to thank my sponsoring teachers Ms.Jocelyne Comstock for providing the space for me to work and help from Dr. Mark Galassi on this project.

9 Data Availability

The Covid-19 Rumors Dataset I used in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found below: https://github.com/MickeysClubhouse/COVII 19-rumor-dataset.

Works Cited

[Che+21] Mingxi Cheng et al. "A COVID-19 Rumor Dataset". In: *Frontiers in Psychology* 12 (2021), p. 1566.

[Dir21] Counter Terrorism Committee Executive Directorate. "Update on the impact of the COVID-19 pandemic on terrorism, counter-terrorism and countering violent extremism". In: United Nations Security Council, 2021.

[Far21] Francesco Farinelli. "Conspiracy theories and right-wing extremism – Insights and recommendations for P/CVE". In: *EU publications* (2021).

[Pha+23] Chau Minh Pham et al. "TopicGPT: A Prompt-based Topic Modeling Framework". In: *arXiv* (2023). eprint: `2311.01449` (cs.CL).

[Ais24] Albladi Cheryl D. Seals Aish Albladi. "Detection of Conspiracy vs. Critical Narratives and Their Elements using NLP". In: *Notebook for the Lab at CLEF 2024* (2024).

[Har24] Yuval Noah Harari. *Nexus: Nexus: A Brief History of Information Networks from the Stone Age to AI*. Penguin Random House, 2024. ISBN: 9783328603757.

[Awa] Soumya Awasthi. *From clicks to chaos: How social media algorithms amplify extremism.* URL: `https://www.orfonline.org/expert-speak/from-clicks-to-chaos-how-social-media-algorithms-amplify-extremism`.

[Enc] The Editors of Encyclopaedia Britannica. *Malleus maleficarum work by Kraemer and Sprenger*. URL: https://www.britannica.com/topic/Malleus-maleficarum.

[Jac] Eda Kavlakoglu Jacob Murel Ph.D. *What is Latent Dirichlet allocation ?* URL: https://www.ibm.com/think/topics/latent-dirichlet-allocation.

[Nov] Beth Simone Noveck. *The Dark Side of Progress: Harari's Grim AI Predictions in Nexus*. URL: https://rebootdemocracy.ai/blog/nexus.

[Tea] The QuickFrame Team. *How Does the Twitter (X) Algorithm Work in 2025*. URL: https://quickframe.com/blog/the-twitter-algorithm/#:~:text=Twitter's%20algorithm%20learns%20about%20users,feed%20on%20the%20Twitter%20platform..

[Wil] Debra Aho Williamson. *Global Facebook Users 2020: The Pandemic Brought Back Momentum in Lagging Regions and Led to Even Higher Growth in Others*. URL: https://www.emarketer.com/content/global-facebook-users-2020.

Understanding and Predicting Trail Maintenance Needs Using Machine Learning Techniques

Data Science, Machine Learning

Luke Rand, Isaac Olson

Final Report

New Mexico Supercomputing Challenge 2024-2025
Santa Fe Preparatory School
Santa Fe, NM, US
March 30 2025

Contents

1 Introduction **3**
- 1.1 Executive Summary . . . 3
- 1.2 Problem Statement . . . 4
- 1.3 Tools Used . . . 4
- 1.4 Current State of the Field . . . 4

2 Methodology **5**

3 Research and Interviews **5**

4 Data **6**
- 4.1 Database . . . 6
- 4.2 Primary Datasets . . . 6
  - 4.2.1 Projects Dataset . . . 7
  - 4.2.2 Lines Dataset . . . 7
- 4.3 Creating Working Table . . . 9
  - 4.3.1 Determining Mileage . . . 10
- 4.4 Secondary Data . . . 12
  - 4.4.1 Precipitation . . . 12
  - 4.4.2 Slope . . . 12
  - 4.4.3 Traffic . . . 12
- 4.5 Multithreading . . . 13

5 Sub-segmenting Dataset **14**

6 Machine Learning **14**
- 6.1 Linear Regression . . . 14
- 6.2 Multivariate Linear Regression . . . 14
- 6.3 Complete Model . . . 16

7 Analysis **17**

8 Results of the Model **17**
- 8.1 Data Insights . . . 17
- 8.2 Model Application . . . 18

9 Conclusion **20**
- 9.1 Conclusions . . . 20
- 9.2 Limitations . . . 20
  - 9.2.1 Data Limitations . . . 20
  - 9.2.2 Financial Limitations . . . 21
- 9.3 Solutions . . . 21
- 9.4 Foundation and Government Applicability . . . 22
- 9.5 Acknowledgments . . . 22
- 9.6 Generative AI Use . . . 23

9.7 Additional Links 23

10 Works Cited **24**

1 Introduction

1.1 Executive Summary

Countless Miles of Trail have been built across the United States. With limited funds and man hours to maintain those trails, improper allocation of resources harms the overall health of our trail systems and in turn limits community access to the outdoors, which is important to mental, physical, and emotional health. Our project achieves a method to better understand and standardize trail maintenance needs, allowing both large and small trail maintenance organizations to allocate funds and resources in a way that better maintains the health of the entire system.

This study used supervised machine learning to train a model to predict trail maintenance hours from environmental factors by providing a dataset of human determined maintenance. Geographic and maintenance hour data was obtained as a dependent variable from the Pacific Crest Trail Association, deemed the most efficient and leading group in trail maintenance due to the size of their organization and detailed reporting. Independent variables such as traffic, slope, and precipitation data were then correlated with maintenance hours in order to create the training data for a machine learning model. Before generating the model, multiple graphs were created to analyze to determine the model's accuracy. Lastly, the model was tested on a local trail.

This process determined that these environmental variables do influence trail maintenance needs, or at least how often the Pacific Crest Trail Association maintains the trail. Increased slope and traffic both increase maintenance, while increased precipitation decreases maintenance, likely because both slope and traffic encourage erosion, while precipitation discourages it by encouraging the fostering of healthy plant root systems. Still, the correlation between these variables and maintenance is existent, but somewhat weak. Thus, the model provides insight as a starting point, but is imperfect. However, the ability to estimated maintenance hours of a trail using a function that interfaces with our model provides an invaluable starting point for government and local trail organizations.

The model's limitations highlight that the Pacific Crest Trail Association does not determine maintenance data purely using the above variables, despite citing them as the most influential. Thus, the model could be improved with more perfect data, perhaps by analyzing a smaller acreage of trail more carefully and performing maintenance truly as needed, discovering additional independent variables, or working with the Pacific Crest Organization to improve both their methods and our model. However, despite the limitations of the model, the project is import framework which contributes to producing more efficient trail maintenance practices. Additionally, the ability to create a dataset for each of

these environmental variables at a latitude-longitude point is invaluable on its own to many of these organizations.

1.2 Problem Statement

Routine trail maintenance is necessary to maintain the health of our trail systems. Due to limited funding, limited resources are available to complete this maintenance and sustain or trail systems for the community. At the same time, physical activity improves mental and bodily health and fosters relationships between people and with the environment, and trails provide learning spaces for children, improving emotional, physical, and mental health. The importance of trails to a healthy community and world is undeniable. As such, it is important to properly allocate available funding and resources to maintain these systems. Currently, resource allocation decisions are made by humans, leaving some areas of trails prone to being under maintained or receiving more maintenance than necessary at the determent of other trails. Inefficient trail maintenance results in the United States Forest Service spending $80 Million yearly while still having a 157,000 mile maintenance backlog.[1] Our model seeks to streamline the trail maintenance process and ensure that all trails receive the necessary maintenance to be a sustainable resource to the community.

1.3 Tools Used

The model was developed using Python, a programming language known for its numerous libraries and readability. DynamoDB was chosen as the database software for its scalability and price. Git and GitHub streamlined the process of working in a team by allowing for version control and a remote repository.[2]

Essential libraries included `boto3`, a tool for interacting with the DynamoDB remote database, `threading`, a library for implementing multithreading, `scikit-learn`, a machine learning library, `matplotlib` for graphic representations, `PIL` for image handling in webscraping, and `pyppeteer` for webscraping. A full list of libraries used can be found on GitHub.

1.4 Current State of the Field

Currently both government and private trail maintenance organizations make decisions about trail maintenance allocation based on trail assessments by users and agency personnel. The United States Forests Service has a program, Trail Assessment and Condition Surveys (TRACS) that keeps track of trail condition based on surveys completed by personnel and trail crews. Although this program does help allocate trail maintenance, our research yields no organizations that use machine learning models to predict trail maintenance need. Our model is a first step in automating trail maintenance planning and has the potential

[1] Data from 2018
[2] GitHub linked in 9.7

to standardize and revolutionize the field in a way never done before. The application of machine learning in this field has the potential to democratize the trail maintenance process and ensure that all trails are maintained to the community standard at drastically decreased annual spending numbers.

2 Methodology

- First, extensive research was conducted in order to determine the factors that influence trail maintenance. Slope, weather, and traffic were determined to be the primary independent variables.
- Next, datasets from the Pacific Crest Trail Association containing information on trail maintenance projects with geographic data were used to generate a table of geographic points containing the hours of maintenance per mile per year at the location.
- Independent variables were taken from API's or web scraped at the location of each point and appended to the items of the table.
- Two dimensional graphs were constructed relating each independent variable to the maintenance value, and then linear regression was performed.
- Three dimensional graphs relating two independent variables to the dependent variable were constructed with multivariate linear regression.
- All five graphs were analyzed to determine whether patterns were present in the data and to understand the efficacy and accuracy of the linear regressions performed.
- A four dimensional model relating all three independent variables with the work on the trail was constructed to allow smaller or less efficient organizations to emulate the process of the Pacific Crest Trail Association to more effectively carry out maintenance.
- The model was further tested and verified using local trails in Santa Fe.

3 Research and Interviews

Initial research was conducted using Internet resources and reaching out to local and national trail maintenance organizations. Multiple organizations shared informative information about how they conduct trail maintenance and what factors they take into account when allocating their resources. These resources were valuable and were the basis for choosing trail slope, traffic, and precipitation as independent variables for the model.

The Pacific Crest Trail Association (PCTA) responded to our email and was kind enough to provide us with a large amount of trail maintenance data and allow us to conduct an interview. An extensive interview was conducted with Galen Keily, the Geographic Information System (GIS) Specialist at the PCTA. Mr. Keily explained how the PCTA trail maintenance data set was collected and laid out. He then outlined how the PCTA conducts trail maintenance with the goal of maintaining its trails in accordance with the PCTA Comprehensive Management Plan. Further insights provided by Mr. Keily and in this document around how the PCTA makes decisions around trail maintenance proved invaluable in understanding the outcomes of the model, and aided in determining the PCTA to be the leading organization in maintenance efficiency, and thus a strong provider of training data.[3]

4 Data

4.1 Database

Due to the relatively large size of the datasets that we use for this project, setting up a database was necessary to implement persistent and flexible storage. Additionally, a remote database reduced the size of the git repository and local storage requirements. With large amounts of data, this was a necessity. The DynamoDB NoSQL cloud database solution was elected due to being flexible, free, fast, and lending itself well to the project schema. A NoSQL database was selected for its scalability and dynamic nature, a requirement for a project where it was not certain from the beginning how many entries would be required and which additional fields may have became necessary. DynamoDB, like many other NoSQL databases, operates using keys which query a particular entry in a table and allows for reading of values from the entry.

Each initial primary dataset was uploaded to the database, and then operations were performed on these datasets to create new working tables.[4] In order to efficiently access entries in the database, a JavaScript Object Notation (JSON) file containing all the keys in an array was saved. This allows for local iteration through objects in order to make requests to the database. While saving these files required storage space, it was significantly less than locally storing the entire contents of the database.

4.2 Primary Datasets

The primary datasets used, which provided dependent variables and means to collect independent variables, were roughly 4 years of digital logs from the Pacific Crest Trail Association (PCTA). The PCTA manages 4265 kilometers

[3] Elaborated in 8
[4] Elaborated in 4.2

of nearly continuous trail along the western coast of the United States, covering varying environment and human conditions, including rain, snow, wildfire, mountainsides, bike traffic, and hikers. This data was provided by Galen Keily, the Geographic Information System (GIS) Specialist at the PCTA. Two datasets were made available.

4.2.1 Projects Dataset

The first dataset is a list of trail maintenance projects performed by the PCTA. Entries contain a unique ID and numerous fields. The fields applicable to the project are those that contain the number of total hours spent on the project, across all volunteers and staff members, titled "hours", and the date range for the project, titled "date". The entries begin in the middle of 2021 and extend through early 2025. This dataset contains 11742 items, although many of the entries do not contain corresponding geographic data, and thus were not used. The dataset was provided in ShapeFile format and was converted to JSON before iterating through the object and uploading entries to the remote table *(fig. 1)*.[5]

ID *(String)*	date	hours
a1i7V000004Dna4	7/13/2022 - 7/20/2022	405
a1iUS000000AdZZ	4/5/2024 - 4/5/2024	19.5
a1i5w000007HvJq	5/1/2021 - 5/5/2021	372

Figure 1: Selected items in the projects table.

4.2.2 Lines Dataset

The second dataset contains 1621 lines on a map of the world, denoted as a list of points in Mercator Web Projection (EPSG:3857),[6] and the project ID which each of these lines is associated with *(fig. 2)*. To upload to the database, the ID was selected as the primary key, while the list of points (line) was uploaded in json format as a string *(fig. 3)*. However, since multiple lines can share an ID for a project that spans multiple areas, each item contains an array of lines, and IDs with only one line contain an array with a single line.[7]

[5]"Table" will be used to refer to data on the remote database, while "dataset" will be used for other forms.

[6]Mercator Web Projection is a coordinate system that uses the distance in meters from the latitude-longitude coordinate of (0,0)

[7]Note that each line is additionally a two dimensional array, holding points with a latitude and longitude coordinate.

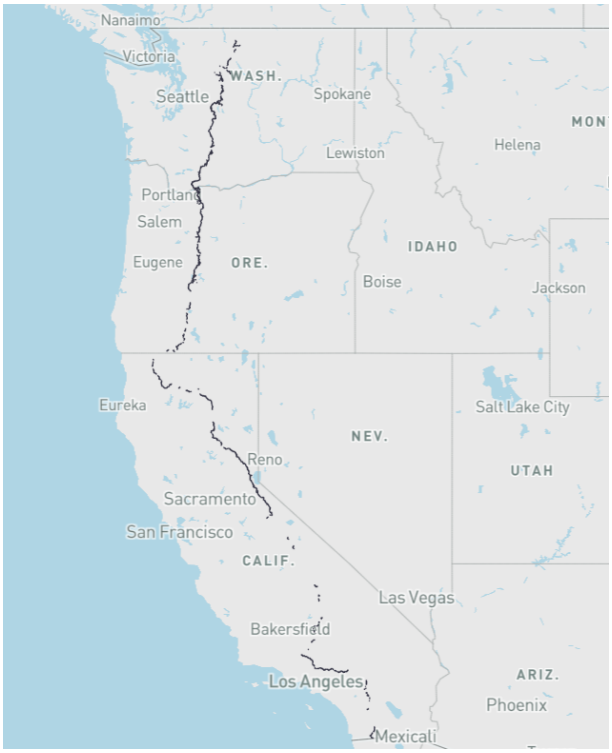


Figure 2: Project lines from the PCTA overlaid on a map.

ID - *Partition key*

a1i5w000007IwP0

points

[[[-13569194.9313, 5726321.7051], [-13569224.431, 5726319.634400003], [-13569230.2196, 5726323.138599999], [-13569237.121399999, 5726357.543200001], [-13569235.006299999, 5726405.964599997]], [[-13569207.955699999, 5726320.749399997], [-13569224.431, 5726319.634400003], [-13569230.2196, 5726323.138599999], [-13569234.0045, 5726342.0929000005]]]

Figure 3: Selected item in the lines table.

4.3 Creating Working Table

Using the data for any machine learning techniques or other analysis requires correlating the data with additional variables and creating a set of uniform items with fields to be correlated. The solution used was to create an additional table that used geographic points located on the PCTA as the primary key, and contained as values the hours spent per mile at the point for each year. Next, an average was calculated from the three years with complete data, 2022, 2023, and 2024, and appended as a value to the point. Secondary data was later added as additional values. Since the value for work done is in hours per mile per year, it is not necessary for the points to be equidistant, as the hours per mile in an area is intrinsic to a point.

This was accomplished by iterating through the list of ids and determining the hours spent on each project and the dates of work from the `projects` table. Next, the coordinates of the corresponding trail were queried from the `lines` table, and the length of trail worked on for the project was determined.[8] Next, each point from the line was assigned a value for hours per mile, calculated as $project\_hours/trail\_length$. The point was uploaded to the database, and the hours per mile value was appended to the point in a field titled as the year the project began, determined by parsing the date value. If a previous project had already updated the year field, the value was added to the existing value. This process was repeated for each project with associated lines, creating a table of 97,825 points. Next, an average was calculated using the three complete years. Finally, a field was added containing the latitude and longitude values in standard form for each point, converted from the previous Web Mercator Projection, to ease computation when correlating secondary data. Thus, the `points` table was fully prepared for independent variables *(fig. 4)*.

```
# Function to create the points table using the lines and projects table.
def create_points(ids): # pass in a list of project ids for indexing
    #iterate through each project
    for id in ids:
        #get project from database
        project = dynamodb.get_item(id) #function simplified for readability

        #get time (total hours by all people) and date
        time = project['hours']
        date = project['date']

        #get line from database
        line = dynamodb.get_item(id)
```

[8] Elaborated in 4.3.1

```
14
15          # get points
16          point_data = json.loads(line['points']) # needs to be
            ↪  loaded from a string
17
18          # lines is for distance calculation, points is just all
            ↪  the points involved
19          lines, points = combine_lines(point_data)
20
21          # get trail mileage
22          length = get_distance(lines)
23
24          # determine the year
25          date1,date2 = date.split(" - ")
26          date2=datetime.strptime(date2, date_format)
27          year=date2.year
28
29          # calculate hours per mile
30          hourspermile = time/length
31
32          # create or modify an entry for each point in the line
33          for point in points:
34              roundedpoint = [int(round(var, 0)) for var in point]
                ↪  #round the point so small changes in gps don't
                ↪  duplicate points
35
36              update_point(roundedpoint,hourspermile,year) # create
                ↪  or update the point in the database
```

point *(String)*	2021	2022	2023	2024	2025	avg	latlon
[-13236900, 4498584]				22.08983…		7.363277…	[37.42550625585259, -118.909095843…
[-13512701, 5989554]		10.75016…	4.757511…			5.169225…	[47.29009391971521, -121.386658380…
[-13557800, 5471932]	16.26259…			9.954065…		3.318021…	[44.041921980728425, -121.79178959…

Figure 4: Selected item in the points table before correlating independent variables. Hours represent the hours per mile at the location of the point.

4.3.1 Determining Mileage

In order to construct the points table and understand maintenance needs, trail length data was required. Calculating mileage of the lines dataset proved difficult, as multiple lines contributed to each project and often had overlap, yet did not share exact point values. This was remedied by combining lines where points were within one meter of each other using the following code, and finally calculating the entire distance of the lines associated with a project

using a simple multi-point distance algorithm. The unit of miles was chosen over kilometers because the outputs of the prediction model will likely be used primarily by United States residents outside of academia.

```
# Function to combine lines with tolerance
def combine_lines(lines):
    #familiar_points contains points seen before, and is used to generate the working table, while combined is the combined lines
    familiar_points = []
    combined = []

    # iterate through each of the lines in order to remove duplicated areas
    for line in lines:
        newline = []
        lastpoint = None
        for point in line:
            is_familiar = False #used to track familiarity of the point

            # if points are familiar, cut the line segment off, and restart when they stop being familiar
            for checkpoint in familiar_points:
                if close(point, checkpoint):
                    is_familiar = True
                    lastpoint = checkpoint
            if is_familiar:
                newline.append(point)
                if len(newline) > 1:
                    combined.append(newline)
                newline = []
            else:
                if len(newline) == 0 and lastpoint != None:
                    newline.append(lastpoint)
                newline.append(point)
                familiar_points.append(point)
                lastpoint = point

        # add the line to the combined set of lines
        if len(newline) > 1:
            combined.append(newline)

    # the returned values will be used to calculate mileage and create the working table
    return (combined,familiar_points)
```

4.4 Secondary Data

4.4.1 Precipitation

Precipitation data was collected using the Open Meteo Weather API, which provides historical weather data for the continental United States. Due to pricing barriers, weather data was collected using the last 90 days of data and yearly predictions were calculated, as earlier data is exclusive to a paid API key.[9] The API was accessed through the Python "requests" library. A python function, `get_average_rainfall(lat, lon)`, takes latitude and longitude as parameters and returns an estimated annual rainfall in millimeters. Another function calls `get_average_rainfall()` for each point in the `points` table, adding annual rainfall as an independent variable.

4.4.2 Slope

Slope data was obtained using the open Meteo Elevation API. This API provides elevation at a specific point with 90-meter resolution. Higher resolution elevation data is available from other APIs with a paid API key.[10] A python function, `find_slope(lat, lon)` takes 8 points in a circle with a radius of 100 meters around the given latitude and longitude and finds their elevation using the Python "requests" library. The function then takes the difference between the maximum and minimum elevations and the distance between the two points to calculate the slope of the ground surrounding the trail. Finally, the slope is converted to degrees using an inverse tangent formula.

4.4.3 Traffic

A relative unit of trail traffic is obtained via web scraping from an open source map project, freemap.sk. This map has a trail use heatmap layer that is scraped from trail use data on a popular athlete social media site called Strava. Trail traffic data is scraped from freemap.sk instead directly from the Strava heatmap due to a friendlier web scraping environment and background map layers that area easier to isolate from the heatmap data.[11] A Python function `trail_use_scraper` opens freemap.sk on chrome using the Python library "pyppeteer", a Python port to the Node.js library puppeteer. The function then calls another function `enter_coordinate()`. This function navigates to the entered coordinate and desired zoom level to take a screenshot *(fig. 5)*. It then isolates all pixels of the heatmap and calculates the average luminosity of the pixels, determining an indicative value for traffic. The returned value is representative of trail traffic because the method used effectively reverses the process used in the heatmap to visually represent trail use.

[9] Elaborated in 9.2.2
[10] Elaborated in 9.2.2
[11] Elaborated in 9.2.1

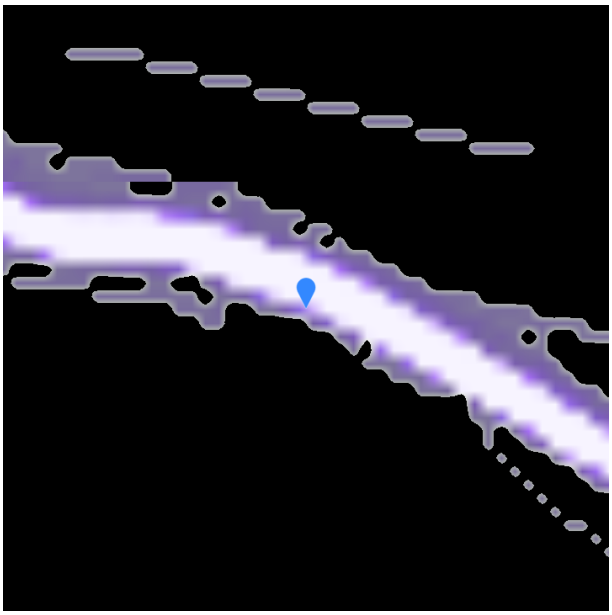


Figure 5: Example manipulated screenshot from webscraping for traffic determination.

Thus, the `points` table was finalized with independent variables and ready for analysis and constructing a model, with each point containing average hours per mile per year, latitude-longitude pairs in both Mercator Web Projection and standard form, annual precipitation, slope, and cumulative traffic *(fig. 6)*.

point *(String)*	avg	latlon	slope	traffic	weather
[-13236900, 4498584]	7.363277...	[37.425506...	16.172159...	12.681646...	1321.4759036144578
[-13512701, 5989554]	5.169225...	[47.290093...	20.052082...	11.760891...	1578.734939759036
[-13557800, 5471932]	3.318021...	[44.041921...	1.1457628...	12.994540...	2441.981927710843

Figure 6: Selected item in the points table after correlating independent variables. Yearly maintenance values are hidden for readability

4.5 Multithreading

Many of the operations performed in the generation of the `points` table involved a significant number of input-output requests to external databases or APIs. Additionally, other aspects of the project such as generating graphs and constructing the model involved many queries to the remote database. Thus, the runtime of many operations that were performed in relation to entire tables of multiple thousands of entries were limited not by compute speed, but rather

by internet latency. Thus, implementing multithreading across many parts of the code-base, including integration of all secondary variables, querying from the database, uploading to the database, and interfacing with APIs, reduced runtime drastically. Multithreading, in contrast to multiprocessing, does not utilize multiple processors, but rather rotates through many threads in quick succession on a single processor, creating the illusion of concurrency. This allows many input-output requests to be sent in quick succession without waiting for the previous request to complete.

5 Sub-segmenting Dataset

Many of the API's used had limitations on requests for the free tier of access.[12] Thus, we were forced to use only a sub-segment of the table to allow for integration of independent variables on all points of the dataset. 1000 points were randomly selected from the points dataset and used for correlation of secondary data, analysis, and construction of the model.

6 Machine Learning

Our data is visualized and the model is constructed using single variable and multivariate linear regression, a simple supervised machine learning technique. Supervised machine learning uses input data and correlated output data to develop a model that fits the provided data. Thus, if the input data follows a specific pattern, the machine can create predicted output data when given unfamiliar input data. Linear regression techniques develop a linear model that is designed to fit the data provided, and thus can predict outputs. Our dataset contains inputs of multiple variables and an output of maintenance hours per mile per year, making it a perfect candidate for machine learning to create a predictive model that can determine maintenance needed on unfamiliar data.

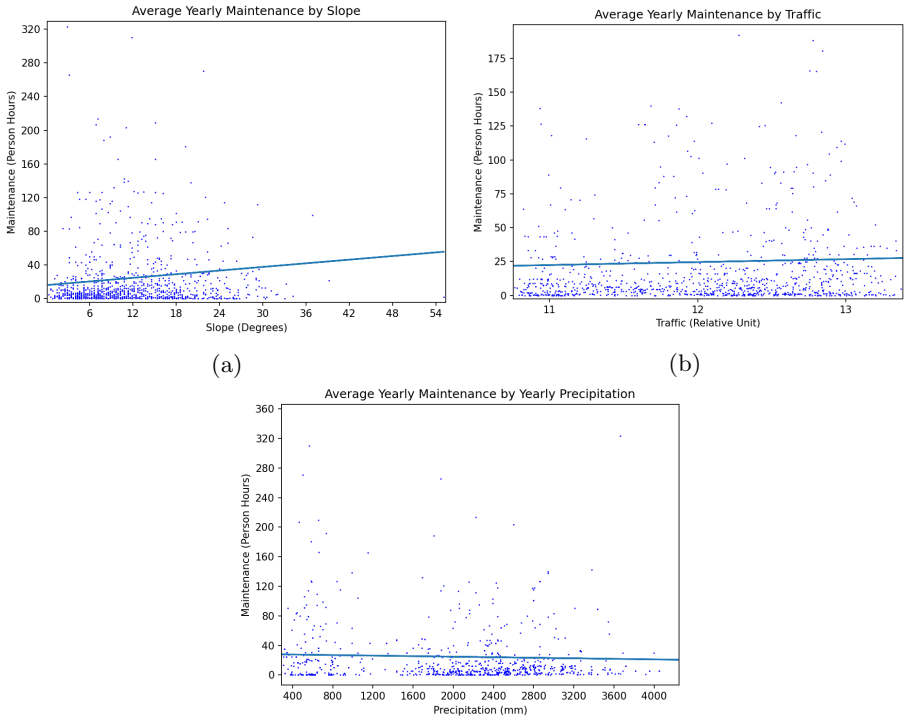
6.1 Linear Regression

To initially understand our data, three two dimensional graphs were produced, and linear regression was performed on the relationship between each of our secondary variables and the dependent variable of work hours per mile per year. This generated 3 graphs for analysis *(fig. 7)*.

6.2 Multivariate Linear Regression

To improve understanding of our data, graphs correlating two independent variables with work hours were produced. Thus, three graphs correlating each pair of independent variables with the dependent variable provide further insight into the working dataset.

[12] Elaborated in 9.2.2



(a) (b)

(c)

Figure 7: Graphs correlating independent variables with yearly maintenance requirements, with linear regression.

Multivariate linear regression was performed on each of these. In contrast with the single variable regressions, these models were developed using only 800 points of the 1000 point dataset, reserving the remaining 200 to understand the validity of the model. The graphs below show the prediction model overlaid with the 200 test points *(fig. 8)*.

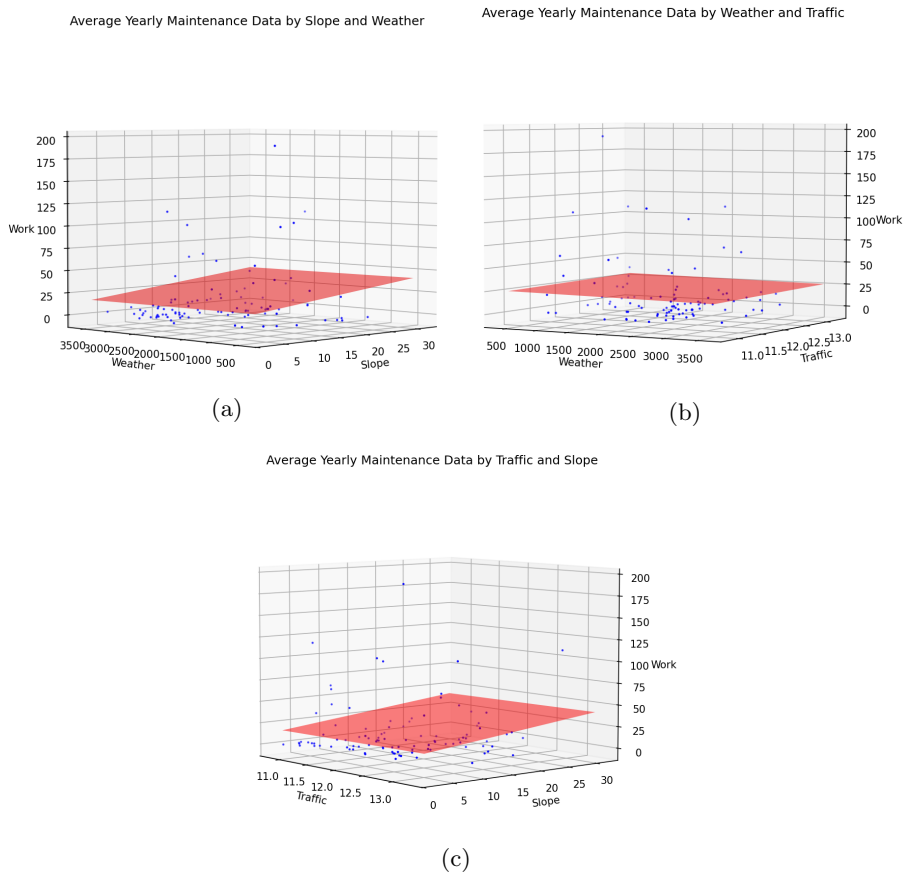


Figure 8: Graphs correlating two independent variables with yearly maintenance requirements, with multivariate linear regression.

6.3 Complete Model

Additionally, a linear regression model taking all three independent variables into account was produced, although it cannot be visually displayed and analyzed as a graph would occupy 4 dimensions. We can, however, understand its efficacy and reliability using the other graphs.

7 Analysis

Studying the first three graphs *(fig. 7)*, it is evident that both traffic and precipitation hold less significance than slope in determining the amount of maintenance performed by the PCTA.[13] Across the entire span of traffic data, from 10.82 to 13.33,[14] the projected change in yearly hours per mile is 5.444. Thus, traffic has some effect, with increased traffic implying increased maintenance. Across the spread of yearly precipitation, the projected hours decrease by 7.418, implying that precipitation causes slightly decreased trail maintenance hours. Slope appears the most significant influence on trail maintenance needs, with a positive difference of 27.52 hours per mile per year across the range of slopes. It is worth noting, however that slopes above roughly the halfway point are rare, so a spread of around 18 can be understood as more significant. It is additionally clear that none of the lines represent a strong fit, as many of the points are far from the line of best fit. This implies that the data does not hold a strong pattern.

The three dimensional graphs add little understanding, and share many of the same traits as the two dimensional graphs, including their lack of certainty due to poor fitting of the data. They do help us to understand that the highest projected maintenance occurs at low weather and high slope, low weather and high traffic, and high traffic and high slope, respectively, a fact that can be inferred from the two dimensional graphs.

The imprecision in the two dimensional and three dimensional regressions casts doubt on the validity of a predictive model constructed using this data. Thus, we can, at best, understand the four dimensional model constructed as a best guess or starting point that attempts to emulate the work of the PCTA.[15]

8 Results of the Model

8.1 Data Insights

Although the model is built on somewhat weak data, the model can still offer some interesting insights into trail maintenance necessity. Our model shows that more maintenance is needed at high slope, high traffic, and low weather area. An area with these characteristics is also one that is most likely to be susceptible to erosion. Large amounts of trail use can degrade trails overtime. Along with this, larger slopes create more damage from water runoff and, although it is counterintuitive, lower amounts of precipitation will likely increase erosion. Low

[13] It is important to note that this does not necessarily reflect optimal work hours, as will be elaborated in 9, 9.2.1 and 9.3

[14] These are relative values calculated from luminosity of heatmap pixels, and has no standalone meaning

[15] This uncertainty is a result of data, not procedure, as will be elaborated in 9

precipitation generally indicates lower vegetation density and thus less organic matter such as roots is present to hold together the topsoil. With no structure holding together the soil, major weather events will cause much more erosion.

Although we originally expected to see a larger correlation between trail traffic and maintenance need, the more uniform results offer an insight into the PCTA trail maintenance strategy. The slight upward trend in trail maintenance when traffic increases indicates that the PCTA spends slightly more time maintaining higher traffic areas but it also full fills its commitment to manage all 4265 kilometers of trail in its purview. A disproportionally large amount of work in only the high traffic areas could negatively affect the heath of the trail system in the long run.

8.2 Model Application

The model was applied to a trail in our local area, the Atalaya mountain trail *(fig. 9)*.

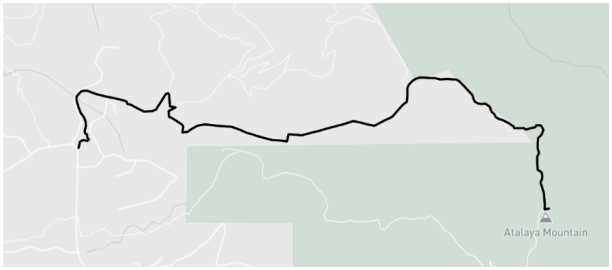


Figure 9: Atalaya trail Shapefile line overlaid on map.

The model was then applied to various points along the trail, which generated values of work hours per mile per year at each point. These values from along the length of the trail were averaged to get a final value of 27.31 hours per mile per year. Given a trail length of 2.2 miles, our model recommends that an organization maintaining this stretch of trail dedicates 60.09 work hours per year across all workers in order to maintain the trail in accordance with the trail maintenance standards outlined in the PCT comprehensive management plan. This comes out to roughly a standard day of trail work for a crew of 7 people.

```
#Return value for trail maintenance need in hours per mile per
↪  year at a specific point
def run_point(lat,lon):
    # download the presaved model
    with open('model.pkl', 'rb') as file:
        model = pickle.load(file)

```

```
7       # get the average precipitation at the point
8       weatherval = weather.get_average_rainfall(lat,lon)
9
10      #convert latlon to meters. This is necessary because the
        ↪  slope and traffic functions takes coordinates in Mercator
        ↪  format.
11      transformer = Transformer.from_crs("WGS84", "EPSG:3857")
12      lat,lon = transformer.transform(lat,lon)
13
14      #get slope at the point
15      slopeval = slope.find_slope(lat,lon)
16
17      # get the relative traffic value at the point. Called
        ↪  asyncronously
18      trafficval = asyncio.run(trailuse2.scrape_one(lat,lon))
19
20      # initialize the input data
21      ind = {"weather":[weatherval], "slope":[slopeval],
        ↪  "traffic":[trafficval]}
22      X = pd.DataFrame(ind)
23
24      #Run model using the values found for weather, slope, and
        ↪  traffic.
25      return model.predict(X)[0] #returns a single item array, so
        ↪  we take the first value
26
27  #This function takes an array of points along a trail and the
    ↪  length of trail in order to return suggested maintenance
    ↪  amount in # of work hours per year
28  def run_trail(trail_points, trail_miles):
29      #add together the values of hours per mile per year for each
        ↪  point
30      trail_vals = []
31      for point in trail_points:
32          #find value at each point
33          trail_vals.append(run_point(point))
34
35      #average values and return total number of hours
36      hours_per_mile_per_year = statistics.mean(trail_vals)
37      return(hours_per_mile_per_year*trail_miles)
```

9 Conclusion

9.1 Conclusions

Our model uses supervised machine learning to predict trail maintenance need based on training data from the Pacific Crest Trail Association and a variety of independent variables recommended by trail maintenance experts. The model provides a way for trail maintenance organizations to allocate their limited funding in a way that ensures all trails are up kept to a certain standard. This functionality offers exciting possibilities in trail maintenance and ensuring our trail remain a valuable resource for the community.

The data set for the model provides interesting insights into understanding the science of trial maintenance. The data collected demonstrates how the environmental variables that were recommended to us by experts in the field, do affect trail maintenance necessity. The correlations in the data between these environmental variables and trail maintenance need aligns with how experts predicted it would. Although the correlation between these environmental variables is undeniable, it is somewhat weak. This suggests that the PCTA does not only base its trail maintenance decisions on these variables. This limitation of the model is not based in its methodology, rather, it represents data set error. The same procedure could be expanded on with a stronger dataset to obtain more accurate results.[16]

Although the model is imperfect due the data it is built upon. It still offers exciting prospects for the field of trail maintenance. As demonstrated in 8.2 the model can easily and effectively be applied to trail systems around the country. If the model is applied to a complete trail system by an organization, it will completely revolutionize trail management. The ability to accurately predict trail maintenance need across an entire system will allow for easier resource budgeting and more effective resource allocation.

9.2 Limitations

9.2.1 Data Limitations

Data availability limitations somewhat handicapped the effectiveness of the model. The trail maintenance data provided by the Pacific Crest Trail Association (PCTA) is an incredible data set and invaluable to the functioning of the model. However, the PCTA has only been tracking their trail maintenance in this form for about 4 years so the data set is still relatively young. As data for more years becomes available the model will be able to produce more accurate insights into trail maintenance necessity.

[16] Expanded in 9.3

Our model also ran into limitations surrounding trail use data. As mentioned in 3.4.3, the trail use data originates from the athlete social media site Strava. Strava offers access to its raw trail use data through its program Strava Metro to urban planners, trail networks, and city governments to help improve mobility infrastructure. We reached out to Strava Metro in order to include their data in the model, however, Strava Metro denied our partnership request citing a large volume of requests and therefore only providing data to groups directly involved in active transportation infrastructure planning. Due to this limitation our model was forced to use a relative unit of trail use derived from heatmap web scraping. If a large trail maintenance organization such as the PCTA were to use our model in trail maintenance planning it is likely that Strava Metro would approve the partnership application and provide more accurate trail use data.

9.2.2 Financial Limitations

The model was forced to use less accurate approximations for both weather and slope data due to API paywalls. We were unwilling to pay for this data but a larger government or non profit institution using the model for trail maintenance planning would have the financial bandwidth to pay these relatively small fees that are used for API upkeep. An API key to the more accurate weather and slope data APIs only costs a total of 99 dollars per month putting the data well within reach of any organization planning on implementing the model for trail maintenance planning on a larger scale.

9.3 Solutions

As outlined above in 9, the correlation between the environmental variables and trail maintenance necessity is present but not perfect. This manifests in a less accurate model, however, the problem is not one of methodology and could be fixed with a stronger data set.

The weaker than expected correlation suggests a limit in the accuracy of the PCTA trail maintenance data. Although the PCTA is one of the leading trail maintenance organizations in the nation it is still probable that much of their trail maintenance resources are not allocated in the ideal way. One possible solution would be to work with the PCTA or a similar organization and segment off smaller sections of their trail to perform trail maintenance in that area truly as needed. This new data would serve as the ideal training data for our model and eliminate problems with the current trail maintenance dataset.

Strengthening the independent variable data would also improve the accuracy of the model. As outlined in 9.2.1, insufficiency with the current data set could be easily resolved by a larger organization using the model for trail maintenance planning. It is also possible that another interdependent variable needs to be

added to the model to improve its efficacy, which further research would aluminate. If the PCTA is making trail maintenance decisions based on an additional variable, this could throw off the model if is not included.

9.4 Foundation and Government Applicability

As mentioned in section 1.4, the United States Forest Service currently makes decisions around trail maintenance resource allocation using the Trail Assessment and Condition Surveys program (TRACS). The TRACS program currently is an organized framework for collecting data on trail conditions by agency personnel in order to better allocate resources towards trails in the worst state of disrepair. Although this program is has a large amount of value in ensuring trail health, it is purely a reactionary strategy that only responds to trail degradation when trail functionality is already limited. Our model has the potential to further develop this program by optimizing maintenance scheduling. The model will prioritize maintenance spending by predicting areas that are most likely to deteriorate soon, effectively switching the entire trail maintenance strategy from reactive to proactive.

The implementation of this model also has the potential to reduce the overall cost of trail maintenance. A more uniform and proactive approach to trail maintenance will allow personnel to identify and fix smaller problems with the trail before there are large problems that are costly to repair. This will also lead to a greater ability to predict trail maintenance costs giving managers a clearer picture of upcoming budgetary needs. This allows for better financial planning and reduces problems with overspending.

Additionally, an understanding of how the PCTA actually runs maintenance and how that interacts with environmental factors will allow them to improve their pipeline. Addressing these factors would not only improve PCTA maintenance, but allow for the training of a stronger model in future years, creating a feedback loop of improvement and benefiting the entire industry.

9.5 Acknowledgments

The Pacific Crest Trail Association made this project possible by kindly sharing the trail maintenance data that is the foundation of our project. We would like to specifically thank Galen Keily, the GIS specialist for the PCTA, for generously donating his time to explain the data set as well as allowing us to conduct an interview to better understand the trail maintenance process. Additionally, we would like to acknowledge our club sponsor Ms. Comstock for providing valuable guidance and a framework for achieving success. Lastly, we would like to thank the New Mexico Supercomputing Challenge for their feedback and aid across all stages of the project.

9.6 Generative AI Use

In the interest of using industry standards and utilizing available tools, the generative artificial intelligence Chat GPT was used in select cases to streamline the coding process and improve efficiency. Due to the limitations of the technology itself, it was used to write small snippets of code (database queries and API requests), or given pseudo code to transform for more simple functions whose work was heavily syntactical as opposed to logical (querying the weather API and calculating line distance). Lastly, in the interest of education and because Chat GPT is incapable of correctly programming complex functions, and often makes mistakes on simpler ones, the technology was utilized only for simple functions (those that could easily be logically planned and understood within seconds), and the human programmers carried out appropriate research and learning beforehand to fully understand the outputted code in order to debug. Essentially, Chat GPT was used to generate only code that the programmers could have written themselves with minimal cognitive stress, and was purely used to increase efficiency.

9.7 Additional Links

`https://github.com/lukerand/trails`

10 Works Cited

References

[1] "Benefits of Hiking." *National Park Service*, www.nps.gov/subjects/trails/benefits-of-hiking.htm. Accessed 19 Dec. 2024.

[2] Bevans, Rebecca. "Multiple Linear Regression — a Quick Guide (Examples)." *Scribbr*, 22 June 2023, www.scribbr.com/statistics/multiple-linear-regression.

[3] "Docs Open-Meteo.com." open-meteo.com/en/docs.

[4] "Elevation API Open-Meteo.com." open-meteo.com/en/docs/elevation-api.

[5] "Freemap Slovakia, Digital Map." *Freemap Slovakia*, www.freemap.sk.

[6] Kirvan, Paul. "Multithreading." *WhatIs*, 26 May 2022, www.techtarget.com/whatis/definition/multithreading.

[7] "Machine Learning, Explained — MIT Sloan." *MIT Sloan*, 21 Apr. 2021, mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained.

[8] Pacific Crest Trail Association. "Pacific Crest Trail Association - Preserving, Protecting and Promoting." *Pacific Crest Trail Association*, 20 Feb. 2018, www.pcta.org.

[9] Ray, Tyler. "Congress Passes Trail Inclusive Spending Bill." *American Hiking Society*, 6 Aug. 2021, americanhiking.org/congress-passes-trail-inclusive-spending-bill.

[10] Smallcombe, Mark. "SQL Vs NoSQL: 5 Critical Differences." *Integrate.io*, 15 Feb. 2024, www.integrate.io/blog/the-sql-vs-nosql-difference.

[11] "Trail Assessment and Condition Surveys (TRACS)." *US Forest Service* www.fs.usda.gov/managing-land/trails/trail-management-tools/tracs. Accessed 1 Apr. 2025.

[12] "COMPREHENSIVE MANAGEMENT PLAN for the PACIFIC CREST NATIONAL SCENIC TRAIL" *US Forest Service* www.fs.usda.gov/Internet/FSE_DOCUMENTS/stelprdb5311111.pdf. Accessed 2 Apr. 2025.

[13] "Why Trails Matter: Outdoor Learning." *American Trails*, www.americantrails.org/resources/why-trails-matter-outdoor-learning. Accessed 19 Dec. 2024.

Analyzing Pre-Indo-European Theory of Etruscan Language Origins Using Topological Data Analysis

New Mexico

Supercomputing Challenge

Final Report

April 2, 2025

Welch Homeschool

Team Members:

Helena Welch

Teacher:

Cindy Welch

Project Mentor:

Paul Welch

Executive Summary

High-dimensional data is often difficult to analyze because of the exponential growth of the size of the space in which the data lives as the dimension increases. [16, 3] One example of high-dimensional data comes from language, which contains many different characteristics (dimensions) with which it can be quantified, but not always sufficient data to detect patterns in it. This is especially true for ancient languages, as there is a sparsity of texts from which to draw. [6]

The ancient Etruscan language is currently classified as a non-Indo-European isolate. However, the Etruscans lived in an Indo-European-speaking region and appear to be genetically related to Indo-Europeans. [12, 28] This study aims to bring a quantitative measure, topological data analysis (TDA), to ongoing investigations of Etruscan to more concretely determine Etruscan's similarity to different Indo-European languages. Phonetic patterns in a specific word list translated into different languages by large-language models are encoded, and the distance between two given phonemes based on this encoding is calculated. Results indicate that Sanskrit has the highest correlation to Etruscan. Etruscan appears similar to older Indo-European languages and thus may be older than neighboring languages, explaining its uniqueness compared to Indo-European languages that developed later in time.

Introduction

The Questions Behind Etruscan as a Non-Indo-European Language

Etruscan was spoken in the Etrurian region of Italy around 800 to 100 BC (see Figure 1). [12] With more than 13,000 examples of Etruscan text, we have enough data to understand a fair amount of vocabulary but not to concretely determine the origins of the language. [12] In addition, while much about the language is unknown, linguists have been able to reconstruct in part the sounds of the language because the Etruscan alphabet was borrowed from that of the Euboean Greeks and subsequently passed on to the Romans. [4] Thus, the phonology of the language is fortunately available for use in this study.

The current reigning theory posits that Etruscan was one of the few Pre-Indo-European languages not displaced by the arrival of the Indo-European language family. The only languages that seem most certainly to be related to Etruscan are the obscure Rhaetic and Lemnian languages, of which only a handful of texts are extant. [37] Despite this, however, a recent genetic analysis indicates that the Etruscans were indigenous to Italy and had very similar genes to those of the Romans. [28] If this is the case, then why is their language believed to be so different from that of their Indo-European-speaking relatives?

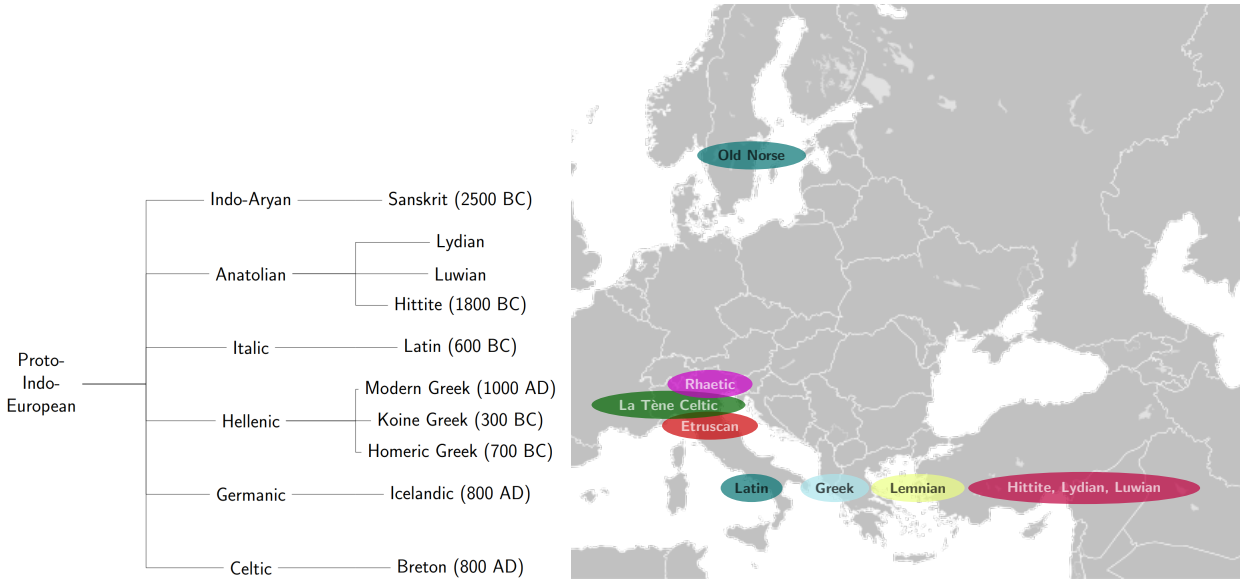


Figure 1: Etruria compared to regions where other ancient languages were spoken. [18]

Other Theories

While the Pre-Indo-European theory is currently most popular, other studies have suggested alternatives. The Anatolian theory originates from writings of the ancient historian Herodotus, which claim that the Etruscans were Lydians who migrated from Anatolia (modern-day Turkey) to Italy. [30] If this is true, Etruscan should exhibit similar linguistic properties to Hittite, a better-recorded relative of Lydian. Others suggest Etruscan was a Greek creole language made up of a mixture of the Greek dialects. The enigma of Etruscan's origin runs so deep that even hypothetical links between Old Norse, the Celtic languages, and Aryan languages such as Sanskrit have been suggested. [32, 13, 7, 15]

Purpose of Project

This project compares the phonology of ancient and contemporary Indo-European languages to that of Etruscan using topological data analysis (TDA), with the aim of discovering which Indo-European languages are most closely related to Etruscan. While statistics aims to fit data points to lines or other geometries, TDA quantifies the distance between data points in a high-dimensional space using topological structures such as n-dimensional holes. [31, 35, 33] This means that it captures higher-level information about the inherent structure of data rather than looking for specific patterns in it that might not exist. [31, 38] Because of this, previous studies have used TDA to detect similarities between languages based on their phonological and grammatical structures. [26, 38] It is known to perform well in comparison to other techniques because of its insensitivity to sparsity of data and noise, which could make it particularly useful in analyzing poorly-documented ancient languages. [38, 35, 33]

Prior Works

While TDA appears to be a novel approach to implement comparative linguistics, two studies that develop a methodology for doing so for different aspects of linguistics are cited here. The first uses TDA to compare grammatic parameters between modern languages [26], while the second proposes an algorithm for comparing the phonetics of languages [38]. The methodology of this study is largely based on the latter's work.

Wolfram's study (2017) performed TDA for large families of modern languages using a list of 200 words and, although laying an extensive groundwork for future work, ultimately determined that a larger data set was needed to prove more conclusive results. In contrast, this study chooses a set of 1700 words and performs Wolfram's (2017) methodology on an ancient language whose ancestry is uncertain. In order to obtain a larger dataset than Wolfram (2017) could acquire, this study uses large-language models (LLMs) to translate a single word list into different languages.

Methods

Data Collection and Preparation

A word list of 1700 Etruscan words and their English translations is drawn from McCallister [1999], and each phoneme in an Etruscan word is then converted to its corresponding International Phonetic Alphabet (IPA) character using conversions provided by Rix [2008] and Ager [Omniglot. 2023]. IPA is a system for encoding phonemes spoken in different languages such that the list of sounds is universal across all languages.

Having acquired an Etruscan word list with its corresponding English translation and converted it to IPA, translations of the word list into other languages are then obtained through the use of large-language models (LLMs). LLMs are chosen to overcome the roadblock of limited available translation data that previous studies met. [38] Each of five LLMs is run from the Ollama server [24] on one of two desktop computers (see Table 1), depending on the size of the models. In addition, a relatively larger model, ChatGPT4o, is run from the OpenAI server. [25] Its running time and cleanliness of the output are notably better.

Each LLM is passed a message communicating its role as an assistant, as well as the prompt, "I will give you a number of words to translate into [a language]. Provide the International Phonetic Alphabet. Do not provide any notes or commentary. Use the format: English: [language]: /IPA/." The list of 1700 words in English is then read from a file and passed in batches of five words as input to the model. Output is then written to a file and parsed to obtain a list of the translation in IPA. While all LLMs are fairly consistent in providing output in the correct format, manual parsing of certain sections in models mistral-small and nous-hermes is required. In addition, all

model	parameters	vocabulary size
ChatGPT4o [34]	200 billion	175k
Mistral-large [20]	123 billion	128k
Mistral-small [22]	22 billion	32k
Command-r [10]	35 billion	128k
Nous-hermes [36]	34 billion	64k
Mistral-nemo [21]	12 billion	128k

Table 1: Characteristics of the large-language models used to translate the Etruscan word list into other languages.

IPA characters outputted across all LLMs are researched, and outdated or nonstandard symbols are replaced to ensure that all translations of the word list conform to the same IPA encoding. Ultimately, this results in an IPA character list of 34 symbols. Etruscan phonology consists of 21 of those characters. [29]

Following the procedure designed by Wolfram 2017, the contextual relations between different IPA characters within a given language are quantified by creating a list of the IPA characters that come before and after each IPA character in a word. This is known as the context list. For example, the context list of *ū* for the English IPA word list *Sūt, St, sūn, rn* would consist of (*S,t*),(*s,n*), and the context list of would be (*S,t*),(*r,n*). The cosine similarity C between two such characters for a given language is then calculated, where S is number of contexts two characters share, N_1 and N_2 are the length of each character's context list, and C is defined as:

$$C = \frac{S}{\sqrt{N_1} * \sqrt{N_2}} \tag{1}$$

This quantifies how irreplaceable one phoneme is with another within a given language. [38] One would expect, for example, that two vowels would have a higher cosine similarity than a vowel and a consonant.

This information can be encoded in a matrix, where each axis is the set of IPA characters and

each value is the cosine similarity between any two IPA characters in a given language. [38] As such, the matrix should be a symmetric, square matrix with values of 1 along the diagonal, as the cosine similarity between two identical IPA characters is 1. A visualization for Etruscan's cosine similarity matrix can be seen in Figure 2, where characters are arranged by vowels (top left corner) and consonants (bottom right). The lighter the color, the more two characters have in common. Dark rows and columns indicate that a given character does not appear in a language.

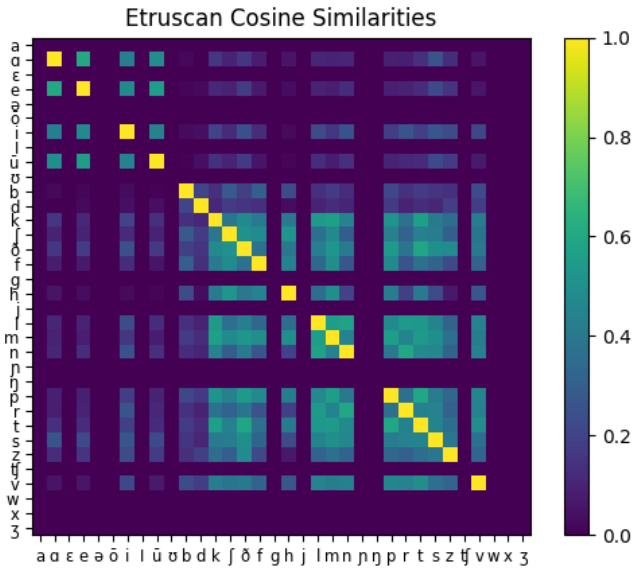


Figure 2: Colorplot for Etruscan's cosine similarity matrix created in matplotlib. [14]

Having quantified the relation between every two IPA characters in a given language, each character is embedded in a coordinate system such that its coordinate axis is the set of cosine similarities between it and each other IPA character. Thus, this point cloud lives in a space with the same number of dimensions as there are IPA characters, which, in this case, is a 34-dimensional space. If an IPA character does not occur in the word list translated into a given language, its cosine similarity with each other character is set to 0. The particular TDA used in this study, persistent homology, is then performed on the point cloud.

Persistent Homology

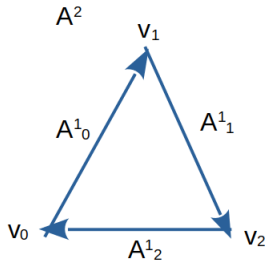


Figure 3: Example 2-simplex.

Persistent homology is performed using the Scikit-TDA python library Ripser. [31] Persistent homology is a type of topological data analysis that draws nth-dimensional balls, n-balls, around a point cloud in an nth-dimensional space. [38] At a set radius of the n-ball, all n-balls that overlap are connected by an n-simplex A^n, where an A^n is the simplest geometry determined by $(n+1)$ connected points in the Euclidean space R^n. [38] For example, 0-simplex A^0 is 1 point, 1-simplex A^1 is 2 connected points (a line segment), and 2-simplex A^2 is three connected points (a triangle).

An n-chain is a sum of n-simplices. It can be described by its boundary δ:

$$\delta(\mathrm{A}^n) = \mathrm{A}_0^{(n-1)} + \mathrm{A}_1^{(n-1)} + \ldots + \mathrm{A}_n^{(n-1)} \tag{2}$$

for n-simplices A^n.

For example, the boundary of an n-chain that contains one simplex, a 2-simplex shown in Figure 3, is

$$\delta(\mathrm{A}^2) = \mathrm{A}_0^{(1)} + \mathrm{A}_1^{(1)} + \mathrm{A}_2^{(1)} \tag{3}$$

For this 2-simplex, the boundary of the boundary is defined as:

$$\delta(\delta(\mathrm{A}^2)) = \delta(\mathrm{A}_0^1 + \mathrm{A}_1^1 + \mathrm{A}_2^1)$$

$$= \delta(\mathrm{A}_0^1) + \delta(\mathrm{A}_1^1) + \delta(\mathrm{A}_2^1)$$
$$= [(\mathrm{v}_1) - (\mathrm{v}_0)] + [(\mathrm{v}_2) - (\mathrm{v}_1)] + [(\mathrm{v}_0) - (\mathrm{v}_2)] = 0$$

(4)

Because of the 2-simplex's orientation, the boundary of each edge (1-simplex) is defined as the difference of its endpoints (the vertices of the 2-simplex). Summing the edges causes vertices to cancel, leaving $\delta(\delta(\mathrm{A}^2)) = 0$. This tells us that the vertices of the 2-simplex are connected. This generalizes to higher-dimensional simplices.

The homology group of the nth dimension, then, is the set of all n-chains that do not cause vertices to cancel. Thus, it describes groups of simplices that are unconnected and have $\delta(\delta(\mathrm{A}^n)) \neq 0$:

$$H_n = \frac{Z_n}{B_n} \tag{5}$$

where Z_n is the set of all n-chains and $B_n \in Z_n$ that have $\delta(\delta(\mathrm{A}^n)) = 0$. [23, 8]

Essentially, H_n measures which points are not connected at a certain radius of the n-balls through calculation of nth dimension holes, where n is the dimension of the simplex. [31] Since a 0-dimensional hole describes the disconnected parts of a 0-simplex, the H_0 homology group is the set of components that are not connected to each other. Each given component, then, contains all points in the 34-dimensional space that are connected by n-balls at a given radius. Each component is known as a cluster.

H_1 measures 2-dimensional holes, and H_2 measures cavitites, known as voids. Note that in the case of language studies, topological structures of a higher order than clusters and holes are often noise and, thus, will not be considered in this study. [38, 27]

Ripser encodes relations between simplices of different dimensions in a boundary matrix,

where each row is an n-simplex and each column represents an $(n+1)$ simplex. If the n-simplex is part of the $(n+1)$-simplex's boundary, it is encoded as a non-zero value in the matrix. [31]

Persistence Diagrams

The Persim library within Scikit-TDA [31] is then used to generate persistence diagrams describing results of the TDA. Figure 4 gives two such graphs, one for German and one for Dutch. These diagrams display the radius of the n-balls at which each cluster or hole is "born" (when it first appears) or "dies" (when it is consumed by another topological structure). Each axis describes the birth or death radius of a topological structure. All clusters (H_0) should fall on a vertical line, since all are born at radius $r = 0$, and it follows that only one will remain as r approaches ∞.

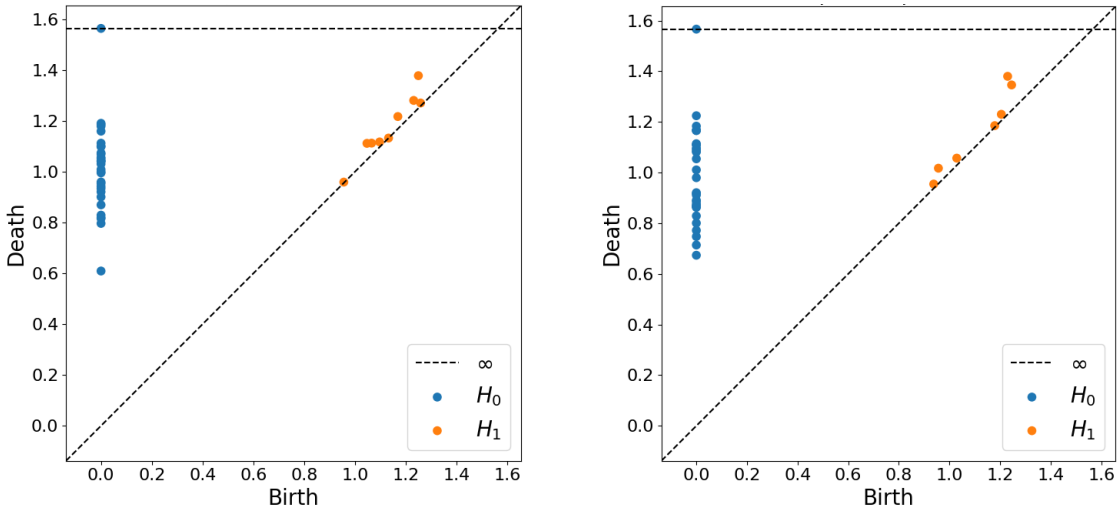


Figure 4: Persistence diagrams for German (left) and Dutch (right); input data for TDA were drawn from mistral-small translations.

Holes (H_1, shown in orange) tend to be near the diagonal, or when birth radius is equal to death radius. The farther a hole is from the diagonal, the larger the radius must become before clusters are joined and the hole disappears. Linguistically, this indicates how similar various IPA characters are to each other, based on how often they appear in the same context in a given language's word list.

Comparison Methods for Persistence Diagrams

After generating persistence diagrams, Persim compares those for Etruscan and eight other lan-

guages. Thus far, these languages consist of Latin, Breton, Koine Greek, Homeric Greek, Modern Greek, Icelandic, Hittite, and Sanskrit. Languages were chosen based on the time and proximity to Etruscan (see Figure 1), as well as availability of LLMs to accurately provide words from their vocabulary. (Icelandic, for example, is known to be linguistically similar to the less well-documented Old Norse.)

For each persistence graph representing a language, the Betti number β_n gives the number of nth dimensional holes across all radii of the n-balls. Thus, β_0 represents the total number of clusters, and β_1 represents the total number of holes. [8] As n increases, β_n decreases; thus, there will be fewer holes than clusters for each language. Since not all IPA characters are used in a given language, the number of clusters and holes for different languages' persistence diagrams will vary.

In this study, the Bottleneck distance B is used to compare persistence diagrams:

$$\mathrm{B}_\infty(X, Y) = \inf_{\eta: X \to Y} \sup_{x \in X} ||x - \eta(x)|| \tag{6}$$

where X and Y are persistence diagrams showing a certain homology group and $\inf \sup$, or the infimum of the supremum, is the largest minimum distance between a point x in X and its bijection in Y, $\eta(x)$. [1]

Conceptually, the Bottleneck distance minimizes the maximum distance between two neighboring clusters or holes of two different diagrams, and B itself is that maximum distance. In the event that one language has more of one topological feature than the other, extra points are paired with the diagonal. [31]

A similar method of comparing persistence diagrams, sliced Wasserstein distance W, approximates distances between birth-death pairs by slicing them N number of ways and projecting them onto one-dimensional lines. Mathematically, it is defined as:

$$\mathrm{W}_\infty(X, Y) = \frac{1}{N} \sum_{i=1}^{N} ||\mu_i - \nu_i|| \tag{7}$$

where μ and ν are the distribution of points for X and Y, respectively, projected onto one-dimensional lines. [5] This metric is similar to the Bottleneck distance, but rather than taking

only the largest distance between two nearest-neighbors into account, it averages across all pairs of clusters or holes.

Validation

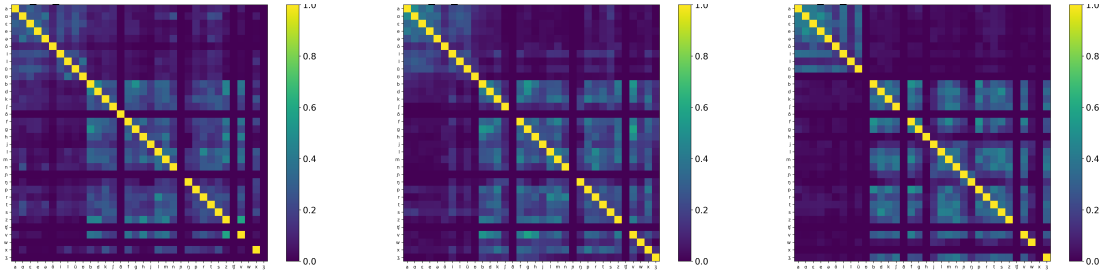


Figure 5: Visualization of cosine similarity matrices for German (left), Dutch (middle), and French (right) before TDA is performed on them.

To determine accuracy of the method detailed above, this study performed TDA on a set of thirteen modern Indo-European languages. Visually, the mistral-small translations for German and Dutch appear more similar through their cosine similarity matrices than German and French (see Figure 5). They can be compared quantitatively using the Frobenius norm metric F, which is defined as:

$$F = ||A - B||_F = \sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij} - b_{ij}|^2 \tag{8}$$

where A and B are cosine similarity matrices and (m,n) describes their dimensions. [9] Quantitatively, the Frobenius metric agrees with the visual colorplots, with F for German and Dutch being much smaller than F for German and French (see Figure 6).

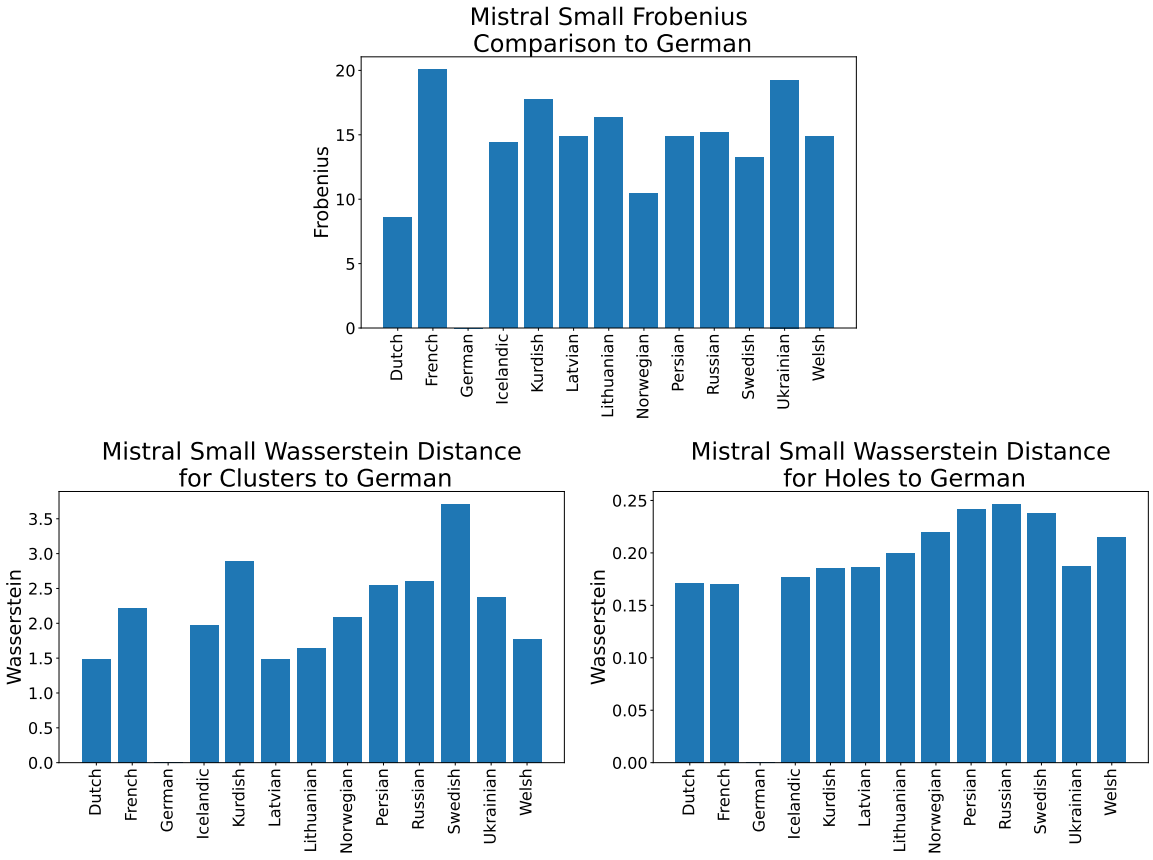


Figure 6: Frobenius norm between cosine similarity matrices compared to sliced Wasserstein distances after TDA.

Now we perform TDA on each cosine similarity matrix and compare persistence diagrams for the above languages using the sliced Wasserstein metric for clusters and holes (see Figure 6). While the more sensitive H_0 sliced Wasserstein distance agrees that German is more similar to Dutch than French, there are some deviations from the patterns in the Frobenius metric. However, this is to be expected, as the purpose of TDA is to detect higher-level information about the language's structure than other metrics (like the Frobenius metric) provide. Thus, validation confirmed that results make sense given general knowledge of modern languages, while also demonstrating the utility of TDA in analyzing data from a different perspective.

Because of biases introduced by variability in LLM translations and using one specific word list, the study must also investigate whether results can be distinguished from arbitrary results. If persistence diagrams do not differ greatly from randomly generated matrices, then results in this

study carry no meaning due to the arbitrary nature of obtaining data. Thus, this study follows Wolfram's (2017) approach in comparing TDA performed on randomly generated matrices with the same symmetry and diagonal as the input cosine similarity matrices. Figure 7 shows three such randomly generated matrices, and Figure 8 gives their corresponding persistence diagrams.

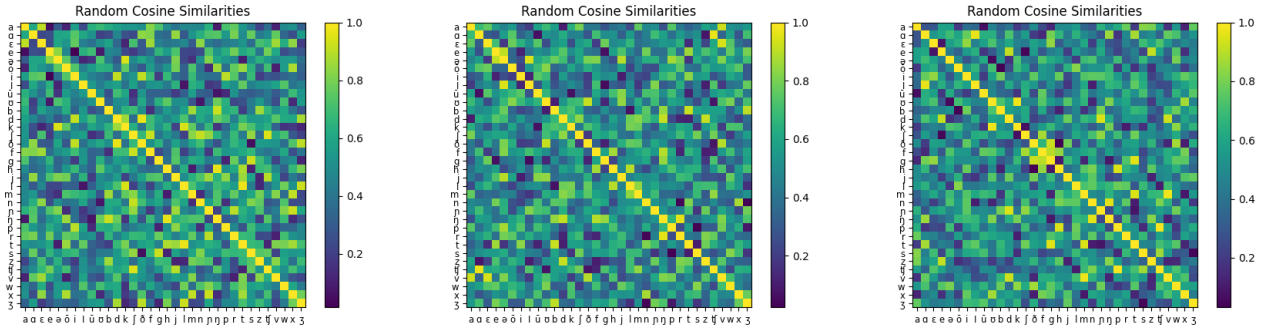


Figure 7: Random cosine similarity matrices generated in numpy and plotted in matplotlib. [11, 14]

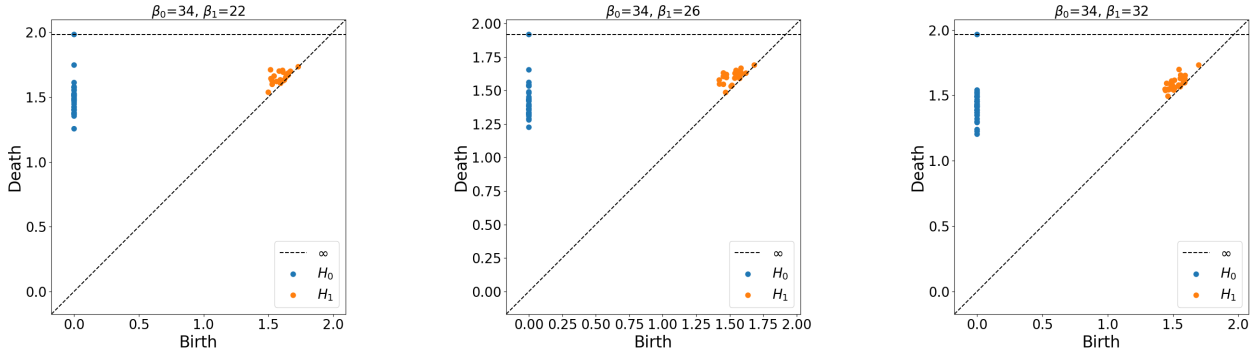


Figure 8: Persistence diagrams corresponding to the above cosine similarity matrices.

Persistence of clusters and holes in the random matrices did not match that of the language data; each topological feature appeared to have less variation in birth and death radii, and β_1 tended to be larger than for the language data. Thus, trends in my data were not random.

Results

Figure 9 gives the persistence diagram for Etruscan. Figure 10 then compares it using the bottleneck distance to each of the eight other languages across the six LLMs. While this is a valid and popular measure, this study found that it produced a very low distinguishability between data, especially for holes.

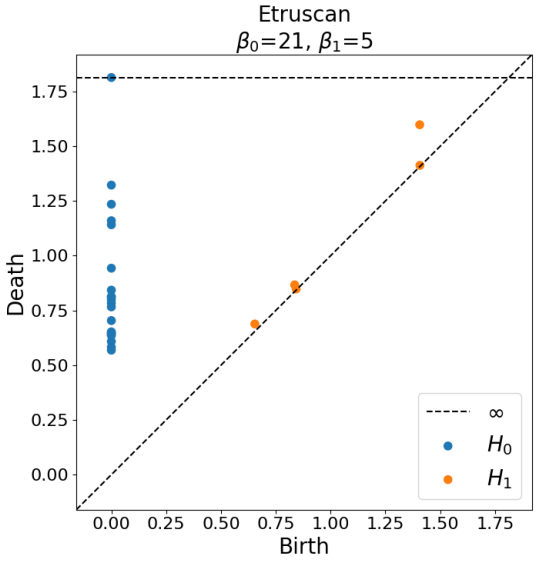


Figure 9: Persistence diagram for Etruscan.

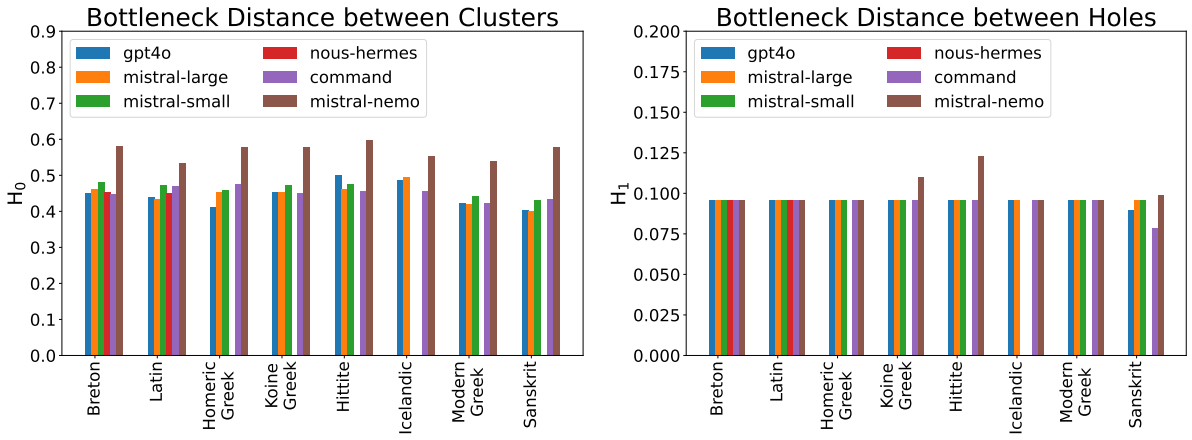


Figure 10: Bottleneck distances between Etruscan and eight other languages.

Figure 11 then performs the comparison using the more sensitive sliced Wasserstein distance.

While Figure 11 shows more variation in the data, there is still no clear trend.

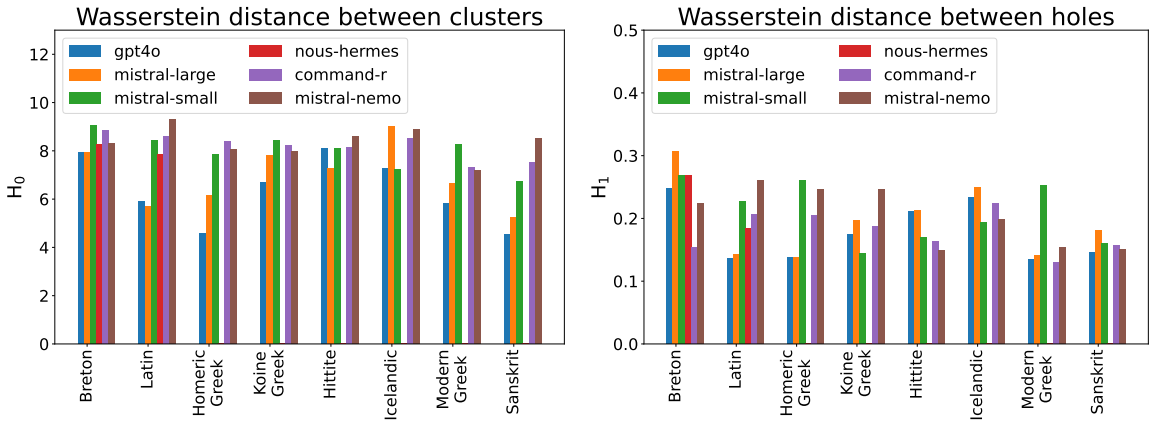


Figure 11: Sliced Wasserstein distances between Etruscan and eight other languages.

We must now consider the size of each LLM. To test the correlation between size of model and similarity in translations, each model's translation of the word list to Latin is compared to that of ChatGPT4o, the largest LLM. TDA is performed, and the sliced Wasserstein distances for H_0 and H_1 are shown in Figure 12. Mistral-large, the next largest model, produced the most similar Latin translations to that of ChatGPT4o. Thus, larger models will produce more similar translations. One can now hypothesize that as model size increases, a trend of the most similarity between Etruscan and a particular language from those tested will emerge.

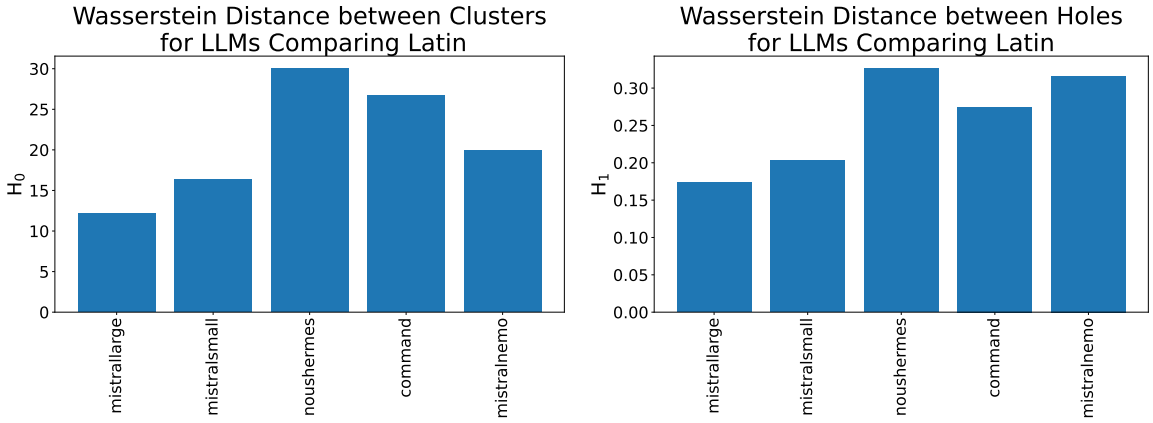


Figure 12: Comparison of LLMs to ChatGPT4o using Latin translations.

Figure 13 gives the sliced Wasserstein distance for H_0 between Etruscan and each of eight

other languages, arranged from greatest to smallest based on ChatGPT4o's output. As can be seen in the figure, ChatGPT4o translations point towards the Indo-Aryan language Sanskrit as being phonetically closest to Etruscan. A distinct trend can now be visualized; as models increase in size, they vary less from the results of ChatGPT4o, and the sliced Wasserstein distance between Sanskrit and Etruscan becomes progressively smaller (right side of Figure 13). Mistral-large and mistral-small agree that Sanskrit is closest to Etruscan. However, the smaller models of nous-hermes, command-r, and mistral-nemo show less of a trend, and other languages have a smaller sliced Wasserstein distance to Etruscan.

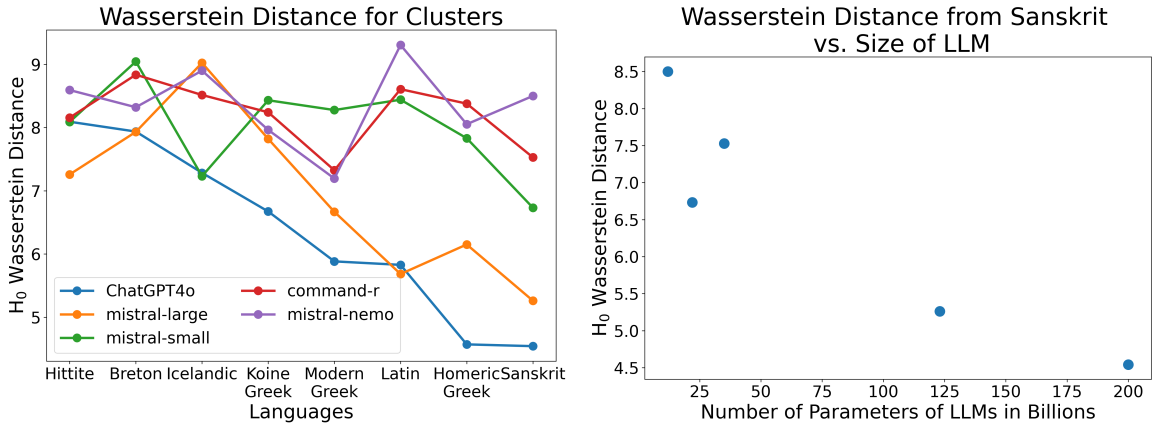


Figure 13: Sliced Wasserstein H_0 distance shows a correlation between Sanskrit and Etruscan.

The idea of Etruscan as a relative of Sanskrit is not currently a mainstream theory. There is little evidence that the Etruscans, supposed relatives of the Romans, were Aryans and spoke a variant of Sanskrit. How, then, do these results make sense? Since Sanskrit is the oldest Indo-European language considered here, it may be closer linguistically to the original Indo-European language, Proto-Indo-European (PIE), than other languages. Two other languages older than the majority of the dataset, Homeric Greek and Latin, appear next-most-similar to Etruscan using ChatGPT4o data. This is summarized in Figure 14, which shows a moderate correlation between age of the Indo-European language and its sliced Wasserstein distance to Etruscan. Thus, Etruscan appears most similar to the oldest of the Indo-European languages, indicating that it may be older than Latin and other languages spoken nearby. This would explain why the Etruscans' language appears unique compared to that of their genetically related neighbors.

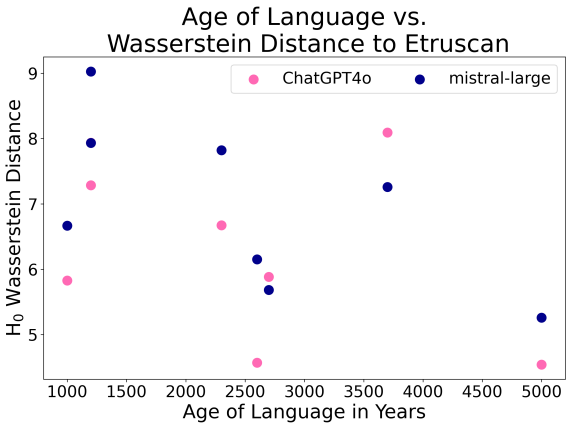


Figure 14: H_0 sliced Wasserstein distance vs. age of language compared to Etruscan.

Conclusions

Analysis of Results

In conclusion, results negate the Anatolian and Greek theories discussed in the Introduction, instead implying that Etruscan is most phonologically similar to older languages than to its neighbors. As such, it may have developed at an earlier time than some Indo-European languages. This would explain why the Etruscans lived in an Indo-European-speaking region and appear to be genetically related to the Romans but spoke a seemingly different language.

To test this hypothesis, access to phonological data for the original Indo-European language, PIE, is needed. However, PIE is a reconstructed language, and its IPA characters do not conform to those of better-understood languages. [17] In addition, LLMs even as large as ChatGPT4o may not provide translations to PIE to an adequately high percent of accuracy.

Analysis of Biases and Limitations

While this algorithm can be extended to a variety of other languages, the results obtained are dependent on the word list chosen, as well as the accuracy of the LLMs in providing translations. This study aimed to eliminate such biases by picking Etruscan words that are not proper names shared across languages, regardless of their inherent phonetic similarity, and utilizing multiple LLMs for translation to other languages. In addition, results could be skewed in the case of multiple IPA characters denoting the same phoneme, as the cosine similarity would not encode

all similarities that exist between two IPA characters. In fact, using more IPA characters than is linguistically accurate biases comparisons between persistence diagrams; the more noisy IPA characters are generated in a particular machine translation, the larger the bottleneck distance between that translation and Etruscan (see Figure 15).

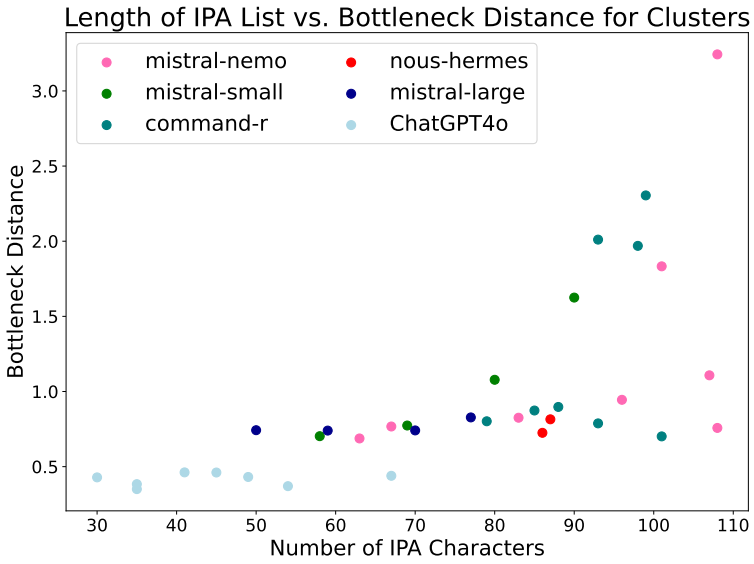


Figure 15: The greater the number of distinct IPA characters in a machine translation and language, the larger the Bottleneck distance between that translation and Etruscan.

To mitigate this bias, all IPA characters were researched, and superseded and nonstandard symbols were replaced to ensure that all translations of the word list conform to the same IPA encoding. Ultimately, this resulted in an IPA character list of 34 symbols.

Next Steps

Having constructed a working model for phonetic comparison between Etruscan and other languages, a more in-depth analysis of current findings and a study of Etruscan compared to a variety of other languages can be conducted. Immediate next steps also include comparing Etruscan to not only Indo-European languages but others language families (Semitic, Sino-Tibetan, etc.). In addition, more LLMs run on online servers, such as Claude and Gemini, will be used in collecting

translation data, and output of TDA will be analyzed through other metrics, such as the distribution of Betti numbers. [38] Hence accuracy of data will be improved.

Summary

Ultimately, the purpose of this project is threefold:

1. To determine the amount of variability between LLMs based on the parameter space in machine translations,
2. To discover the effectiveness of persistent homology as a more quantitative method for comparative historical linguistics, and
3. To conclusively compare the phonetic structure of Etruscan to various Indo-European languages in an effort to determine its origins and thus provide insight into the history behind the Etruscan people.

Appendix

Mathematical Glossary

topological data analysis (TDA): mathematical method of analyzing structure in data using topological features such as clusters and holes

persistent homology: form of TDA used in this study; analyzes how clusters and holes within a data structure persist over multiple scales

simplex: simplest method of connecting a given number of points in a given space; 0-simplex: point, 1-simplex: line segment, 2-simplex: triangle; etc.

n-chain: combination of simplices

birth radius: radius of n-balls drawn around each point in an nth-dimensional point cloud at which a topological feature is formed

death radius: radius of n-balls drawn around each point in an nth-dimensional point cloud at which a topological feature is consumed by another feature

persistence diagram: plot of death radius vs. birth radius for every cluster, hole, etc., that occurs in a point cloud

Linguistic Glossary

phonology: the system of sound values associated with different written characters in a given language

phoneme: a single sound associated with one or multiple written characters

Indo-European: language family from which the majority of Eurasian languages derive

Proto-Indo-European: a completely reconstructed language from which scholars believe all Indo-European languages derive

International Phonetic Alphabet (IPA): system for encoding phonemes as characters that are standardized across all languages

Acknowledgments

This project could not have been initiated without the helpful advice from my parents; their encouragement and the deeper understanding of science, computer science, and mathematics they have given me have been irreplaceable. I would also like to thank the many judges and reviewers from Science Fair and the Supercomputing Challenge, who generously provided me with feedback along the way, and the authors of the numerous papers that contributed to my knowledge of the project's background. Together they have given me the motivation to see this project to completion.

References

[1] Sarit Agami. Comparison of persistence diagrams, 2020. URL https://arxiv.org/abs/2003.01352.

[2] Simon Ager. Etruscan (mekh rasnal). https://www.omniglot.com/writing/etruscan.htm, Omniglot. 2023.

[3] Naomi Altman and Martin Krzywinski. The curse(s) of dimensionality. *Nature Methods*, 15 (6):399–400, 2018. doi: 10.1038/s41592-018-0019-x. URL https://doi.org/10.1038/s41592-018-0019-x.

[4] Giuliano Bonfante and Larissa Bonfante. *The Etruscan Language: an Introduction*. University of Manchester Press, 2002.

[5] Mathieu Carrière, Marco Cuturi, and Steve Oudot. Sliced wasserstein kernel for persistence diagrams, 2017. URL https://arxiv.org/abs/1706.03358.

[6] R Drikvandi and O Lawal. Sparse principal component analysis for natural language processing. *Ann. Data. Sci.*, 10(1):25–41, 2023. doi: 10.1007/s40745-020-00277-x.

[7] John Fraser. *The Etruscans: Were They Celts?* Maclachlan & Stewart, Edinburgh, 1879. URL https://archive.org/details/etruscanswerethe00fras.

[8] Ulderico Fugacci, Sara Scaramuccia, Federico Iuricich, and Leila De Floriani. Persistent homology: A step-by-step introduction for newcomers. 10 2016. doi: 10.2312/stag.20161358.

[9] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins, 3 edition, 1996.

[10] Aidan Gomez. Introducing command r7b: Fast and efficient generative ai. https://cohere.com/blog/command-r7b, 2024.

[11] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`.

[12] Csaba Horvath. Redefining pre-indo-european language families of bronze age western europe: A study based on the synthesis of scientific evidence from archaeology, historical linguistics and genetics. *European Scientific Journal*, 15(26):1–25, 2019. doi: 10.19044/esj.2019.v15n26p1.

[13] Caleb Howells. The surprising etruscan influence on the early celts. *The Collector*, 2023. URL `https://www.thecollector.com/etruscan-influence-celts-connection/`. accessed Jan 24, 2025.

[14] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

[15] Milorad Ivankovic. Etruscan as an aryan indo-european language. 11 2021.

[16] Eamonn Keogh and Abdullah Mueen. Curse of dimensionality. *Enclyclopedia of Machine Learning and Data Mining*, 2017.

[17] Winfred P. Lehmann. Proto-indo-european phonology: 2. pie phonology. https://lrc.la.utexas.edu/books/piep/2-pie-phonology, 2005.

[18] maix. Image own work based on: Europe countries.svg by tintazul, cc by-sa 3.0. https://commons.wikimedia.org/w/index.php?curid=1636794.

[19] Rick McCallister and Silvia McCallister-Castillo. Etruscan glossary, 1999. URL https://etruscans1.tripod.com/Language/EtruscanIntro.html. accessed Jan 24, 2025.

[20] Mistral AI. Au large. https://mistral.ai/news/mistral-large, 2024.

[21] Mistral AI. Mistral nemo. https://mistral.ai/news/mistral-nemo, 2024.

[22] Mistral AI. Mistral small 3. https://mistral.ai/news/mistral-small-3, 2025.

[23] Prerna Nadathur. An introduction to homology. 2007. URL https://api.semanticscholar.org/CorpusID:1431656.

[24] OLLAMA Website. Ollama. https://ollama.com/, 2025.

[25] OpenAI. Chatgpt4o (may 24 version). https://platform.openai.com/docs/overview, 2022.

[26] Alexander Port, Iulia Gheorghita, Daniel Guth, John M. Clark, Crystal Liang, Shival Dasu, and Matilde Marcolli. Persistent topology of syntax, 2015. URL https://arxiv.org/abs/1507.05134.

[27] Alexander Port, Iulia Gheorghita, Daniel Guth, John M. Clark, Crystal Liang, Shival Dasu, and Matilde Marcolli. Persistent topology of syntax. *Mathematics in Computer Science*, 12 (1):33–50, 2018. doi: 10.1007/s11786-017-0329-x. URL https://doi.org/10.1007/s11786-017-0329-x.

[28] Cosimo Posth, Valentina Zaro, Maria A. Spyrou, Stefania Vai, Guido A. Gnecchi-Ruscone, Alessandra Modi, Alexander Peltzer, Angela Mötsch, Kathrin Nägele, Åshild J. Vågene, Elizabeth A. Nelson, Rita Radzevičiūtė, Cäcilia Freund, Lorenzo M. Bondioli, Luca Cappuccini, Hannah Frenzel, Elsa Pacciani, Francesco Boschin, Giulia Capecchi, Ivan Martini, Adriana Moroni, Stefano Ricci, Alessandra Sperduti, Maria Angela Turchetti, Alessandro Riga, Monica Zavattaro, Andrea Zifferero, Henrike O. Heyne, Eva Fernández-Domínguez, Guus J. Kroonen, Michael McCormick, Wolfgang Haak, Martina

Lari, Guido Barbujani, Luca Bondioli, Kirsten I. Bos, David Caramelli, and Johannes Krause. The origin and legacy of the etruscans through a 2000-year archeogenomic time transect. *Science Advances*, 7(39):eabi7673, 2021. doi: 10.1126/sciadv.abi7673. URL https://www.science.org/doi/abs/10.1126/sciadv.abi7673.

[29] Helmut Rix. *Etruscan*, page 141–164. Cambridge University Press, 2008.

[30] Adelle Rogers. Theories on the origin of the etruscan language. *Open Access Theses*, page 1587, 2018. URL https://docs.lib.purdue.edu/open_access_theses/1587.

[31] Nathaniel Saul, Michael Catanzaro, Kostya Lyman, and Roberto Panai. scikit-tda/scikit-tda: v1.1.1, jul 2024.

[32] Ulrich Schädler. Greeks, etruscans, and celts at play. *Archimède. Archéologie et histoireancienne*, 6:160–174, 2019. doi: 10.47245/archimede.0006.ds2.08.

[33] Rohit P. Singh, Nicholas O. Malott, Blake Sauerwein, Neil Mcgrogan, and Philip A. Wilsey. Generating high dimensional test data for topological data analysis. In Sascha Hunold, Biwei Xie, and Kai Shu, editors, *Benchmarking, Measuring, and Optimizing*, pages 18–37, Singapore, 2024. Springer Nature Singapore.

[34] Shaun Sutner. Openai advances llm with gpt-4o; google gemini update looms. https://www.techtarget.com/searchenterpriseai/news/366584023/OpenAI-advances-LLM-with-GPT-4o-Google-Gemini-update-looms, 05 2024.

[35] Funmilola Mary Taiwo, Umar Islambekov, and Cuneyt Gurcan Akcora. Explaining the power of topological data analysis in graph machine learning, 2024. URL https://arxiv.org/abs/2401.04250.

[36] Ryan Teknium, Jeffrey Quesnelle, and Guang Chen. Hermes 3 technical report. https://arxiv.org/abs/2408.11857, 2024.

[37] Bouke Van der Meer. Etruscan origins: Language and archaeology. *BABESCH*, 79:51–57, 2004. doi: 10.2143/BAB.79.0.504735.

[38] Catherine Wolfram. Persistent homology on phonological data: a preliminary study. https://math.uchicago.edu/ may/REU2017/REUPapers/Wolfram.pdf, 2017.

Modeling Fast Moving Objects in Crowded Astronomical Neighborhoods

New Mexico

Supercomputing Challenge

Final Report

April 2, 2025

Welch Homeschool

Team Members:

Kalliope Luna Welch

Teacher:

Cindy Welch

Project Mentor:

Paul Welch

Executive Summary

The purpose of this project is to model the motions of ‘Oumuamua (an unidentified astronomical object) as it traveled through our Solar System. ‘Oumuamua attracts the attention of astronomers due to its sudden increase in acceleration after its slingshot around the Sun. I used Gravitational Particle Dynamics (applying the force and kinematic equations) to create a model of ‘Oumuamua in the gravitational field of our Solar System. I compared this to the observed data of ‘Oumuamua’s actual motions. The model and observed data did not match, suggesting that ‘Oumuamua’s acceleration was not solely gravitational. To test other theories on ‘Oumuamua’s origins and its cause of acceleration, I applied extra forces to the model. I shall continue to add more forces in the future.

Introduction

The movements of a certain set of traveling interstellar objects can be discovered without directly observing them. One can do this by modeling the objects using Gravitational Particle Dynamics.

The object analyzed in this project is known as ‘Oumuamua. When first discovered on Oct. 19, 2017, by the Pan-STARRS1 telescope (funded by NASA’s NEOO), ‘Oumuamua was speculated to be a large chunk of metal and rock that came to our solar system by chance from the distant star Vega. [2, 10] It attracted the attention of astronomers due to its sudden increase in acceleration that occurred while it passed through the orbits of the inner planets. There are many theories as to why it did this.

The first of these theories is that ‘Oumuamua is not made of metal, but of a type of N_2 ice similar to what is found on the surface of Pluto. This theory explains ‘Oumuamua’s movements based on the non-gravitational acceleration caused by evaporating N_2 ice. Furthermore, N_2 gas is difficult to detect, and the red color and sloping surface of ‘Oumuamua are similar to that found on Pluto. ‘Oumuamua’s odd shape could be a result of mass loss the ice suffered as a result of coming too close to the Sun. [9]

A second theory is that ‘Oumuamua is a light sail, or an object that was blown by the solar winds of the sun. This theory hypothesizes that ‘Oumuamua was a comet-like object that was propelled by the Sun’s solar winds (thus explaining the anamolous accleration). [3] A question regarding this theory is whether or not the solar winds of the Sun are powerful enough to counteract the force that had caused it to travel its former course.

The third theory concerning ‘Oumuamua’s origins is that it is an alien spaceship, sent to examine the Sun. [4] A question regarding this theory is that an object moving at the rate of ‘Oumuamua may take a long amount of time to reach the Sun from another star system.

‘Oumuamua is currently difficult to directly view due to the distance between it and the Earth. I modeled it through the application of Gravitational Particle Dynamics to constrain the several theories concerning its origins.

Methods

Retrieving Data

The first step to this project was obtaining the positions of planetary objects in our Solar System. To do this, I obtained data from NASA JPL Horizons Ephemeris files [6] for the Solar System (JPL Ephemeris DE421) and 'Oumuamua (JPL Ephemeris 3788040) and used the Python library of Skyfield [11] to calculate the positions of these planetary objects for an 80-day period in which 'Oumuamua passed through our Solar System. The second step was to obtain 'Oumuamua's positions during these same 80 days. This also required the loading of an Ephemeris file using SPKType21. [12]

Effects of Gravity on 'Oumuamua

I used this data to gain 'Oumuamua's initial position and velocity over the aforementioned 80-day period in the gravitational field of the Solar System. I then applied the force and kinematic equations to simulate the motions of 'Oumuamua due to gravity. These included the equation for gravitational force ($\vec{F}$) (Equation 1), Newton's first law (Equation 2), the equation for acceleration ($\vec{a}$) (Equations 3-4), and the equation for velocity ($\vec{v}$) (Equation 5).

$$\vec{F} = (-Gm_1m_2/r^3)\hat{r} \tag{1}$$

$$\vec{F} = m_1\vec{a} \tag{2}$$

By setting Equation 1 as equal to Equation 2, I get the result of Equation 3.

$$\vec{a} = ((-Gm_1m_2/r^3) * \hat{r})/m_1 \tag{3}$$

$$\vec{a} = \Delta\vec{v}/\Delta t \tag{4}$$

$$\vec{v} = \Delta\vec{r}/\Delta t \tag{5}$$

In these equations, t is time, G is gravity, m_1 represents the mass of 'Oumuamua, m_2 represents the mass of each planetary object in our Solar System, and r is a position magnitude:

$$r^2 = x^2 + y^2 + z^2 \tag{6}$$

The total gravitational force affecting 'Oumuamua is found by adding up the forces due to each planetary object. The gravitational force resulting from Equation 1 is then used in Equation 2 (Newton's first law) to find 'Oumuamua's acceleration. Although m_1 is an unknown variable, it cancels out in the equation for acceleration (Equation 3), and we do not need to know it to calculate the effect of gravity.

For each day in the 80-day time period, I applied the Velocity Verlet Algorithm, as described in Equations 7 - 9. These equations are given for the x-component of the position. There are similar expressions for the y- and z-components. [7]

Solve for $x(t + \Delta t)$:

$$x(t + \Delta t) = x(t) + v_x(t)\Delta t + 1/2a_x(t)\Delta t^2 \tag{7}$$

Solve for $v_x(t + \Delta t)$:

$$v_x(t + \Delta t) = v_x(t) + 1/2a_x(t)\Delta t \tag{8}$$

Calculate $a_x(t + \Delta t)$from the gravitational force (as described in Equations 1 - 4) using $x(t + \Delta t)$. Solve for $v_x(t + \Delta t)$:

$$v_x(t + \Delta t) = v_x(t + \Delta t) + 1/2a_x(t + \Delta t)\Delta t \tag{9}$$

The Theory of N_2 Outgassing

To calculate the effect of the outgassing of N_2, I used the rocket propulsion equations (10 - 12).

$$\vec{F} = m_1\vec{a} = \vec{v_e}\frac{dm_1}{dt} \tag{10}$$

Equation 10 is Newton's first law, in the situation of an object ('Oumuamua) changing in mass. The solution of Equation 10 is shown in Equation 11. [5]

$$\frac{(v(t) - v_g(t))}{v_e} = \ln\frac{m_1}{m_{1,0}} \tag{11}$$

Equation 12 gives the speed of gases that would be ejected from cometary outflows, a phenomena similar to that of N_2 outgassing.

$$v_e = -\tau\sqrt{8kT/\pi m_3} \tag{12}$$

In Equations 10 - 12, $v(t)$ is 'Oumuamua's observed speed and $v_g(t)$ is 'Oumuamua's calculated speed; $m_1/m_{1,0}$ is the mass ratio of 'Oumuamua (where $m_{1,0}$ is the initial mass); t is time; v_e is the speed of the gases that are ejected from the outgassing 'Oumuamua; T is the surface temperature of 'Oumuamua; m_3 is the mass of N_2 (which is 28 g/mol); τ has a value of 0.45; and k (or the Boltzmann constant) is 1.28×10^{23} J/K. In this calculation, τ is an efficiency factor, which is put in to compensate for the fact that N_2 gas spreads out in many different directions as it is ejected from 'Oumuamua. [9].

'Oumuamua as an Alien Spaceship

To simulate 'Oumuamua as an alien spaceship, I used rocket propulsion Equations 10 - 11 to calculate the effects of rocket boosters on a spaceship. These were exactly the same equations as were used in the calculation for N_2 outgassing, excluding Equation 12, which calculates cometary outgassing. As Equation 12 does not apply, v_e (the speed of the gases ejecting from 'Oumuamua) was assumed to be 3 km/s, a typical value for rocket propellants.

Computational Resources

To do these calculations, I used Python and the Python library Numpy, graphing the results with Matplotlib [8] and visualizing the model and data with Paraview. [1] My processor was an AMD Ryzen 9.

Results

The Velocity Verlet Algorithm and kinematic equations were used to a create a model of 'Oumumua's motions under the influence of gravity. When the created model of 'Oumuamua and the observed data taken from the Ephemeris file were placed side by side, they evidently did not match. Figure 1 illustrates the difference in position between the simulated and observed 'Oumuamua over the course of the chosen 80-day time span. This difference was calculated at each day. As the plot

shows, the difference increased over time. Figure 2 displays the difference in speed between the simulated and observed ‘Oumuamua (also over the chosen 80-day time span, and also calculated at each day). The velocities in this circumstance also changed over time. Figure 3 illustrates the difference in acceleration between the simulated and observed ‘Oumuamua, which was a decrease over the 80-day period of time.

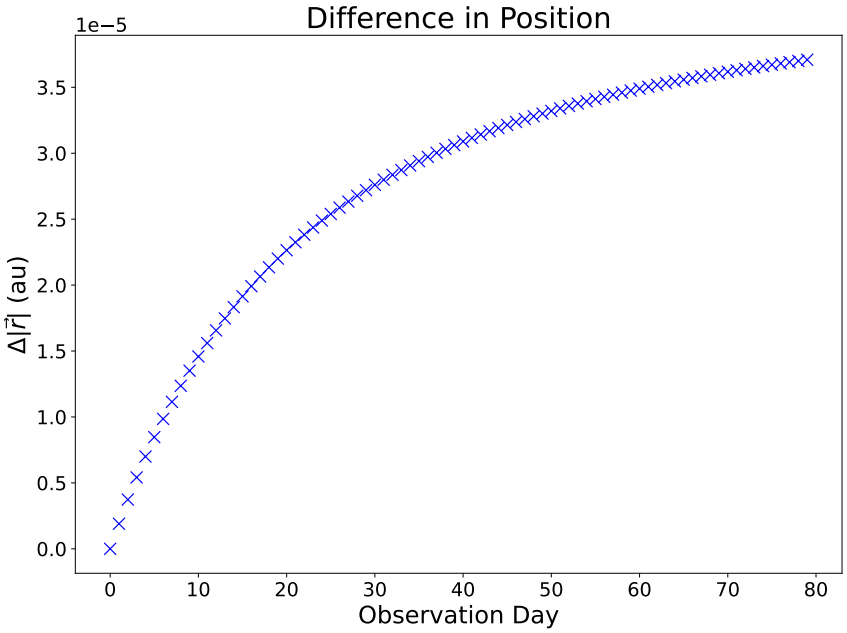


Figure 1: The difference in the positions of calculated and observed ‘Oumuamua data increased over time.

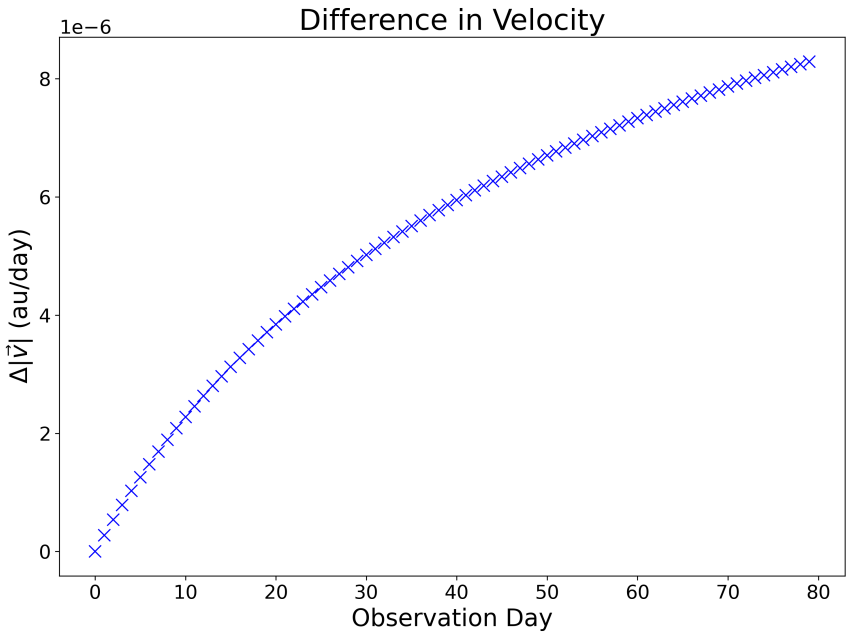


Figure 2: The difference in the velocities of calculated and observed ‘Oumuamua data changed over time.

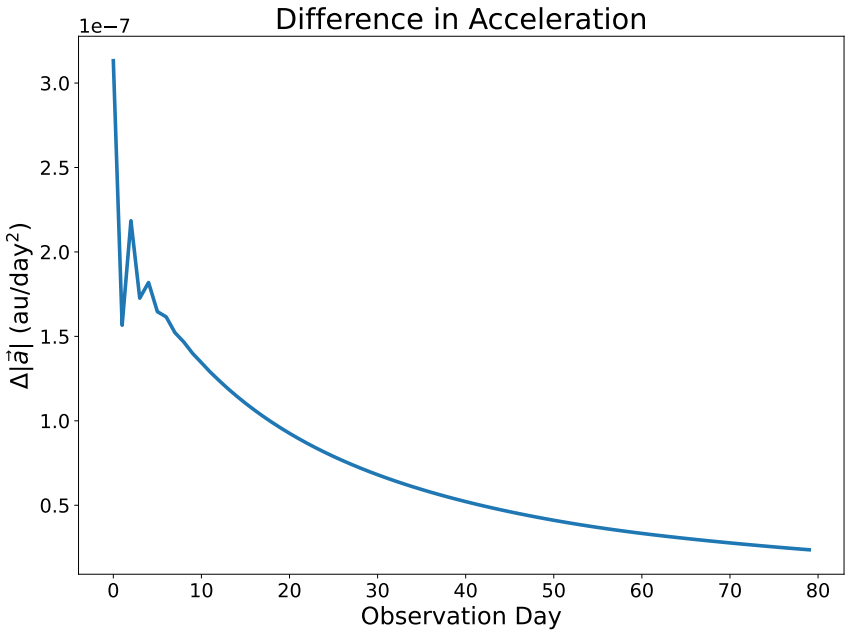


Figure 3: The difference in acceleration decreased over time.

Rocket Equation Implications

After obtaining the results portrayed in Figures 1-3, I used the rocket propulsion equations to calculate the effects of N_2 outgassing and and rocket boosters to account for the difference in speed between the observed and calculated ‘Oumuamua. The next step was to theorize which option was more likely from the calculated motions of each model force.

Figure 4 portrays the mass loss of an outgassing object over time and temperature. While an extreme change in temperature or mass and a sudden change in acceleration would suggest that ‘Oumuamua is an alien spaceship, neither of these were the case. ‘Oumuamua’s acceleration was shown to be too gradual for rocket boosters, and the changes in temperature and mass were (although still significant) relatively mild. This would suggest that ‘Oumuamua is made of N_2 ice. The model of an outgassing ‘Oumuamua that was created in the course of this project closely matched the model proposed by Jackson and Desch[9], who proposed the N_2 theory. In their model, v_e = 62 m/s, and the mass ratio is 88% (or 12% mass loss) at Observation Day 25 and a temperature of 25 K. In my calculations, the % mass loss at Day 25 and 25 K was 11.55%.

Figure 5 shows the mass loss of a rocket with boosters over time. The time was spread across 80 observation days (with an increment of one day). A rocket booster would not have used much fuel to achieve the small mass loss shown in Figure 5. The plotted results do not suggest that there was a sudden jump in mass, making it unlikely that ‘Oumuamua is an alien spaceship.

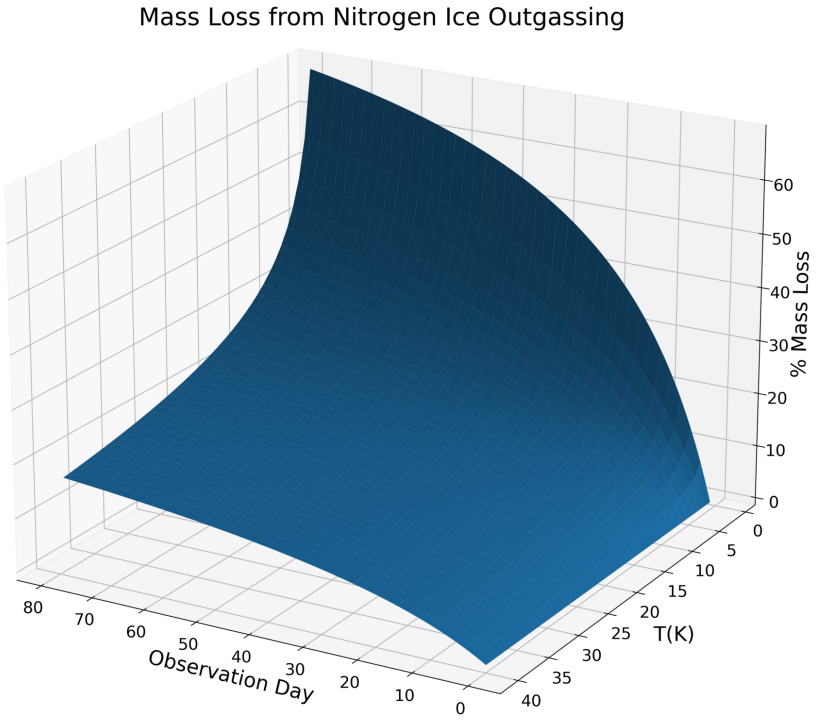


Figure 4: Calculated mass loss due to temperature (in K) over time.

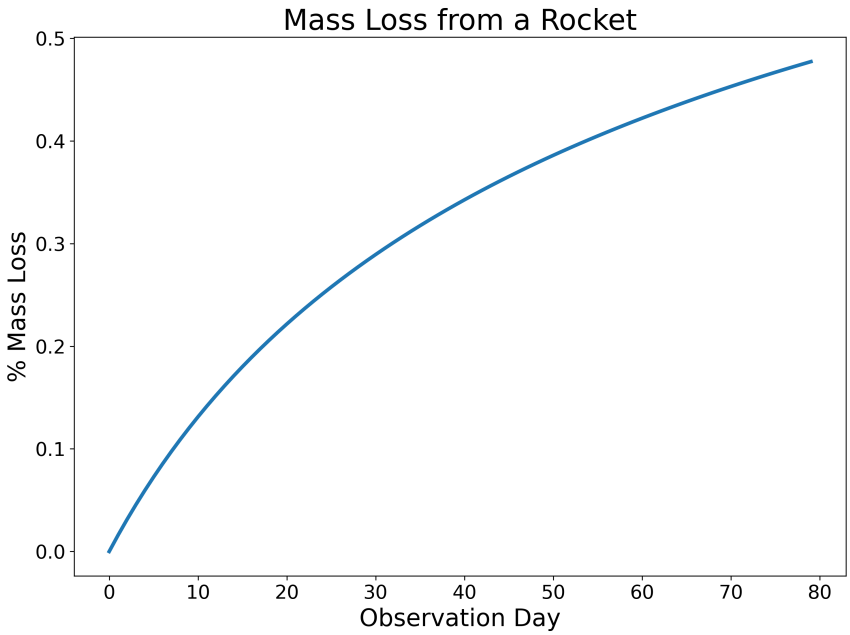


Figure 5: Calculated mass loss due to propulsion from rocket boosters over time.

Conclusions

In conclusion, the motions of the modeled ‘Oumuamua did not match the motions of the observed ‘Oumuamua; this implies that ‘Oumuamua’s acceleration was not solely gravitational. Upon applying the forces of an alien spaceship’s rocket boosters and the outgassing of N_2 ice, the model created throughout the course of this project suggests that ‘Oumuamua was most likely made of N_2 ice. To continue this project, I plan to explore the idea of another additional force that would test the light sail theory. This force would calculate the impact of solar winds on ‘Oumuamua. Should the light sail calculations disagree with the observational data, I would be able to conclude that ‘Oumuamua is most likely made of N_2 ice given the theories presented.

References

[1] J. Ahrens, B. Geveci, and C. Law. Paraview: An end-user tool for large data visualization. Elsevier, 2005.

[2] A. Barnett. ‘oumuamua. https://science.nasa.gov/solar-system/comets/oumuamua/, accessed Oct. 26, 2024, 2024.

[3] S.J. Curran. ‘oumuamua as a light sail – evidence against artificial origin. *Astronomy & Astrophysics*, 649:L17, 2021.

[4] M. Elvis. Research programs arising from ‘oumuamua considered as an alien craft. *arXiv*, page 2111.07895, 2021. URL `https://arxiv.org/abs/2111.07895`.

[5] Douglas C. Giancoli. *Physics for Scientists & Engineers with Modern Physics*. Pearson Prentice Hall, 2008.

[6] J.D. Giorgini and JPL Solar System Dynamics Group. NASA/JPL Horizons On-Line Ephemeris System, https://ssd.jpl.nasa.gov/horizons/, data retrieved 2025-Jan-10.

[7] H. Gould, J. Tobochnik, and W Christian. *An Introduction to Computer Simulation Methods: Applications to Physical Systems, Revised Third Edition*. CreateSpace Independent Publishing Platform, 2017.

[8] J.D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. URL `https://doi.org/10.1109/MCSE.2007.55`.

[9] A. P. Jackson and S. J. Desch. 1i/‘oumuamua as an n_2 ice fragment of an exo-pluto surface: I. size and compositional constraints. *Journal of Geophysical Research: Planets*, 126: e2020JE00670, 2021.

[10] Karen J. Meech, Robert Weryk, Marco Micheli, Jan T. Kleyna, Olivier R. Hainaut, Robert Jedicke, Richard J. Wainscoat, Kenneth C. Chambers, Jacqueline V. Keane, Andreea Petric, Larry Denneau, Eugene Magnier, Travis Berger, Mark E. Huber, Heather Flewelling, Chris

Waters, Eva Schunova-Lilly, and Serge Chastel. A brief visit from a red and extremely elongated interstellar asteroid. *Nature*, 552:378–381, 2017.

[11] B. Rhodes. Skyfield: High precision research-grade positions for planets and earth satellites generator. Astrophysics Source Code Library, record ascl:1907.024, 2019.

[12] S. Uetsuki. spktype21 0.1.0. https://pypi.org/project/spktype21/, accessed Jan 18, 2025, 2018.