FUSION: A PINN (Physics-Informed Neural Network) to Accurately Model Stellar Parameters from Minimal Input Parameters

Menghani, Ansh

January 26, 2025

Abstract

Physics is all about approximation. No one formula can account for every factor and predict any event with full accuracy. Over the years, scientists have used different methods to make these predictions, such as mathematical formulas and statistical models. This has been particularly difficult in fields such as astrophysics, where the subjects being studied are mind-blowing distances away from Earth. In recent years, the advancements in machine learning, specifically in neural networks, allow us to computationally model far more complex relationships than before. The FUSION Stellar Model is designed to model, predict, and classify stellar parameters more accurately than traditional methods. It takes inputs of a star's Effective Temperature, Luminosity, and Radius, and runs its Physics-Informed Neural Network (PINN) algorithm to find twenty-three other parameters of a star such as Mass and Surface Gravity. It was trained on the Gaia mission dataset with over 403 million stellar parameters. The performance of this model varies depending on the star type and parameter it predicts but has an average accuracy of 90.018% and an average speed of 0.008 seconds per round of predictions. While this model is currently limited to stars on the Hertzsprung-Russell diagram (i.e., the model cannot make accurate predictions on other objects in the universe, such as a neutron star), its main use is to better model the distant lights in our universe.

Keywords: Stellar Parameters - Neural Networks - Physical Modeling

1 Introduction

1.1 Stellar Parameters

From the dawn of human history, we have been looking at the night sky. In fact, we are one of the only species to ever do so. What appeared to be bright bulbs in the sky to the ancients are actually far more majestic. They are stars, big balls of hot gas held together by their own gravity. These stars undergo nuclear fusion, the process of releasing energy on an atomic scale by combining atomic nuclei. Most of the stars' gases are in the form of hydrogen. There are also some heavier elements inside stars. The metric that describes the ratio of hydrogen to other elements inside a star is called metallicity. A star's metallicity is one of its many attributes, or stellar parameters. From the beginning of a star's life to its end, it has varying values of temperature, luminosity, and radius (radius is measured by looking at the size of the star in the night sky versus how far away it is [the distance to the star can be measured using multiple techniques such as parallax, redshift, and standard candles]). The effective temperature of a star is simply its surface temperature (measured by looking at the type of light a star emits, specifically it's wavelength), while the luminosity of a star (measured by looking at the brightness of the star in the sky versus how far away the star is) measures the energy output of a star over a period of time. These three parameters are bound together by what is called the Luminosity-Radius-Temperature relation [1] for stars:

$$L = 4\pi R^2 \sigma T^4 \tag{1}$$

L = Luminosity of star in solar luminosities
R = Radius of star in solar radii
T = Temperature of star in Kelvin
σ = Stefan-Boltzmann constant, $5.67 *$

$10^{-8} W m^{-2} K^{-4}$

From these three fundamental parameters, many others can be derived.

Specifically regarding the dimensions of a star, we can use the star's radius to determine parameters like the star's volume. While these relations may be simple, other parameters are far more complicated to model. Here is the summary of the modeled parameters and formulas associated with them.

The absolute bolometric magnitude, which is a value that represents the total brightness of a star at all wavelengths of the electromagnetic spectrum on a logarithmic scale can be calculated using the luminosity of the star [1] by:

$$M_{bol} = 4.8 - 2.5log(L/L_{\odot}) \tag{2}$$

M_{bol} = Absolute Bolometric Magnitude of the star $L/L_{\odot}$ = Luminosity of star in solar units (the $\odot$ indicates a value in stellar units, e.g., $1L/L\odot = 3.83 * 10^{26}$ *Watts*)

In addition to this, the absolute magnitude of a star measures the brightness of a star as if it were to be seen from 10 parsecs six away from invisible spectra. For reference, one parsec is 3.26 light years. The difference between the absolute magnitude and the absolute bolometric magnitude of a star is known as the bolometric correction. The absolute bolometric luminosity of a star is similar. Instead of measuring the brightness across all wavelengths of a star, it measures the energy output across all wavelengths of a star [1]:

$$L_{bol} = 4\pi R^2 \sigma T^4 \tag{3}$$

L_{bol} = Luminosity of star in solar luminosities
R = Radius of star in solar radii
T = Temperature of star in Kelvin
σ = Stefan-Boltzmann constant, $5.67 * 10^{-8} W m^{-2} K^{-4}$

In addition to these parameters, there is also the mass of the star [2]:

$$M/M_{\odot} = (L/L_{\odot})^{2/7} \tag{4}$$

$M/M_{\odot}$ = Mass of the star in solar masses
$L/L_{\odot}$ = Luminosity of star in solar luminosities

There is also, the average density of the star, which is simply the current

$$\frac{mass}{volume} \tag{5}$$

of the star (as density fluctuates from being extremely dense to the core to less dense in the outer layers of a star). The central pressure of the star [3]:

$$P_c = \frac{3GM^2}{8\pi R^4} \tag{6}$$

P_c = Central pressure of the star in $\frac{N}{m^2}$
G = Newtonian Gravitational Constant, $6.674 * 10^{-11} \frac{Nm^2}{kg^2}$
M = Mass of star in *kilograms*
R = Radius of star in *kilometers*

The central temperature of the star [4]:

$$T_c = \frac{GM * u}{R * k} \tag{7}$$

T_c = Central Temperature in Kelvin
G = Newtonian Gravitational Constant, $6.674 * 10^{-11} \frac{Nm^2}{kg^2}$
M = Mass of star in *kilograms*
u = Mass of a Hydrogen atom, 1.661e-27 *kilograms*
R = Radius of star in kilometers
k = Boltzmann Constant 1.381e-23 *J/K*

The predicted lifespan of a star [5]:

$$L_{i\odot} = (L/M)_{\odot} \tag{8}$$

$L_{i\odot}$ = Estimated lifespan of a star in solar lifetimes
$L/L_{\odot}$ = Luminosity of star in solar units
$M/M_{\odot}$ = Mass of the star in solar units

Logarithmically scaled power of the surface gravity of a star [6]:

$$log(a_g) = log(\frac{GM}{R^2}) \tag{9}$$

a_g = Surface gravity of star in Newtons (*N/kg*)
G = Newtonian Gravitational Constant, $6.674 * 10^{-11} \frac{Nm^2}{kg^2}$
M = Mass of star in *kilograms*
R = Radius of star in *meters*

The gravitational binding energy of a star (which is a metric of how powerfully a star is bound together by itself) [7]:

$$G_{BE} = \frac{3GM^2}{5R} \tag{10}$$

G_{BE} = Gravitational binding energy of a star (in *Joules*)
G = Newtonian Gravitational Constant, $6.674 * 10^{-11} \frac{Nm^2}{kg^2}$
M = Mass of star in *kilograms*
R = Radius of star in *meters*

The bolometric flux of a star (which is a measure of the energy emittance of a star per unit of area) [8]:

$$F_{bol} = \sigma T^4 \tag{11}$$

F_{bol} = Bolometric flux of star (in W/m^2)
σ = Stefan-Boltzmann constant,
T = Effective Temperature of Star in Kelvin

Another stellar parameter is the metallicity of a star, the ratio of hydrogen to heavy elements in the star. Stars can also be classified into groups by temperature. The spectral classification table shown below uses the Yerkes spectral classification system [9] to determine the spectral class of a star.

Figure 1: Spectral Classification

Maximum Effective Temperature (°K)	Spectral Class
4000	M
5200	K
7000	G
20000	F
34000	A
42000	B
greater than 42000	O

Figure created by Ansh Menghani (author) using LaTeX.

The star peak emitted wavelength (the wavelength of light it emits the most):

$$\lambda_p = \frac{2897771.955}{T} \tag{12}$$

λ_p = Peak wavelength of star in *nanometers*
T = Effective temperature of star in *Kelvin*

Furthermore, stars can be classified into groups based on their luminosity using the same system. The luminosity classification table shown below uses the Yerkes spectral classification system to determine the luminosity class of a star.

Figure 2: Luminosity Classification

Maximum Luminosity ($\frac{L}{L_\odot}$)	Luminosity Class
0.1	D
25	V
100	IV
1300	III
8000	II
125000	Ib
greater than 125000	Ib

Figure created by Ansh Menghani (author) using LaTeX.

Another stellar parameter used in this project is the star type of the star; the different star types [10] are here:

1. Brown dwarf
   A brown dwarf is a celestial object between the size of the biggest gas giant planets and a star (i.e., a "failed star"). Because of their size (13–80 M_j[1]), they don't have quite enough mass to ignite hydrogen fusion. However, they do emit radiation through gravitational contraction[2].

2. Red dwarf
   A red dwarf is a small and cool star that has a very slow rate of nuclear fusion (they can live for trillions of years). They are usually between 0.08 and 0.6 $M_\odot$ (solar masses).

[1] M_j is a unit of mass where 1 M_j is equal to 1 Jovian (or Jupiter) Mass, 1.898e27 *kilograms*

[2] A large object will contract under its own gravity, causing particles to move closer together and collide. This releases gravitational potential energy in the form of heat energy

3. White Dwarf
   A white dwarf star is an exception to the list order of coolest/smallest to hottest/biggest. White dwarfs are the leftover core of a star that has run out of fuel[3]. It does not undergo nuclear fusion and shines because of leftover energy from the previous star. They have around the same radius as Earth (7000 km) and are between 0.13 and 1.4 $M_{\odot}$.

4. Main sequence
   The vast majority of stars (90%) are main sequence stars. These are stars currently undergoing nuclear fusion and are very diverse in type[4]. They are typically between 1 and 200 $M_{\odot}$.

5. Supergiant
   Supergiant stars are characterized by very high luminosities (1 thousand – 1 million+ $L_{\odot}$ [solar luminosities]) and this phase occurs very late in few stars' lifetimes. They are typically between 10 and 30 $M_{\odot}$.

6. Hypergiant
   A hypergiant star is any star more luminous, larger, and massive than a supergiant star. These are the biggest of all stars. Very few are known to exist and have been observed.

The last stellar parameter used in this project is the stellar classification. The stellar classification of a star is simply its spectral and luminosity class put together, with subdivisions of the spectral class being present. For example, our Sun is a G2V star (a G2 star is hotter than a G3 star. A star in any letter class may have subdivisions anywhere between 0 and 9, e.g., a G-spectral class star can range from a G0 [hottest in the G-spectral class] to a G9 [coldest in the G-spectral class]), indicating that its spectral class is the second subdivision of the G spectral class and its luminosity class is V.

The spectral and luminosity classes can be viewed using a Hertzsprung-Russell (HR) Diagram:

Figure 3: The Hertzsprung-Russell Diagram

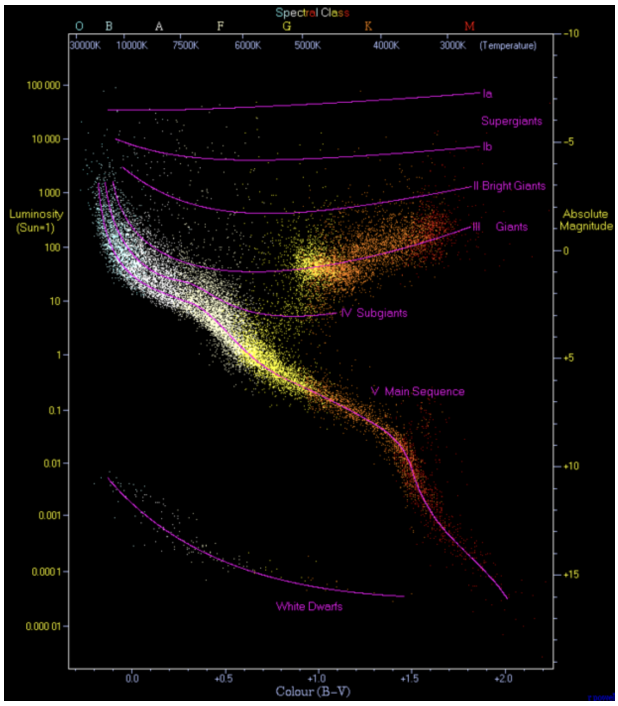


Image source: [11]
The HR diagram the relation between effective temperature and luminosity (on a logarithmic scale) for different types of stars. It also provides information about each star's respective spectral and luminosity classes, absolute magnitude, and color.

1.2 Neural Networks

Now that we have established the mathematical and statistical methods used to obtain stellar parameters, we can delve into the realm of machine learning. The basic idea behind machine learning is to train a "model" to recognize complex patterns in data, bolstering our ability to make accurate predictions and classifications. There are three main types of machine learning [12]:

1. Supervised learning

[3]When a medium-sized star runs out of fuel for fusion, it slowly sheds its layers of gas into space and leaves behind the dense core of the original star made mostly up of carbon and oxygen that the star was not massive enough to fuse. This leftover is a white dwarf

[4]i.e., they can be classified into the different spectral and luminosity classes based on their parameters, as shown in the Yerkes classification tables in Figures 1 and 2

(a) Supervised Learning algorithms take labeled input and output data to find patterns between them and make accurate predictions and classifications.

(b) Common examples of supervised learning methods include random forests, Bayesian learning, logistic regression tasks, and many types of neural networks.

2. Unsupervised learning

(a) This type of learning does not use labeled input and output data like supervised learning. The model is given more freedom to dynamically see complex patterns.

(b) Unsupervised learning is used in many statistical situations due to its benefit in computation.

3. Reinforcement learning

(a) This type of learning introduces rewards and penalties into the model training process to encourage the correct model behavior.

(b) Use cases of reinforcement learning include games, robotics, a few neural networks, etc.

In this project, neural networks are the chosen method. This is due to their ability to recognize complex patterns through the use of intermediate calculations. This is further elaborated on in Section 2. The presence of these intermediate calculations are why many neural networks are considered a form of deep learning[5].

Neural networks are made up of layers (each vertical column of dots in the diagram), and each layer has a specific number of nodes, or "neurons" (the dots in this diagram). The lines connecting the neurons are called weights, and they represent how much one neuron influences

Figure 4: The architecture of a neural network

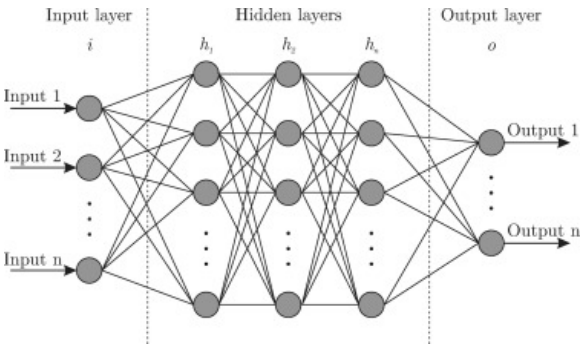


Image source: [13]

the next neuron it connects to (a simple example is when performing linear regression, the weights w_0 and w_1 that define the model are the y-intercept and slope of the line, respectively: $y = w_0 + w_1 x$). Essentially, a neural network will take x inputs, perform multiple "hidden" iterations of regression (where it uses the weights parameters in the regression) on the inputs, and use that to generate y outputs based on the training data. This is the basic structure of a neural network and many other techniques can be employed to improve the model. Many of these will be discussed in future sections [14].

A neural network is influenced by the way our brain works. They look at real-world data, just like our brains look at real-world stimuli, and make decisions based on the data. For example, if you've never ridden a bicycle in your life, the first time you ride one, you will likely fall. Probably the second or third time as well. Eventually, you get the hang of it. What happened was your brain gathered data about what you were doing and how well it worked or not, and using this, it adjusted the technique you used to balance on the bicycle until you could ride it with ease. This process is known as bottom-up learning (i.e., learning from experiences with external stimuli) and is very similar to what happens in a neural network.

Neural networks can be used for many tasks such as classification or regression. A classification task is a task in which the neural network sorts data (such as sorting images by background color). A regression task indicates a task

[5]Deep learning is a type of artificial intelligence that models the human brain through neural networks. They can work to find complex, high-level patterns in a dataset that wouldn't be traditionally (non-computationally) feasible to extract

in which the neural network makes some sort of numerical prediction (e.g., the price of a certain stock in the future).

A neural network model takes data as inputs, runs it through the model structure as described in Figure 4, and makes predictions based on that data. This process is known as a "forward pass." For example, if a neural network model is trained to classify the colors of the rainbow based on input images of individual colors, the images would be the input data, and the classification would be the network's output. The first prediction (or forward pass) the model performs will likely have a bad accuracy. However, just like a human, the model will look at its output and compare it to what the model should've done (this information is also a part of the training data). The measurement of "how good the model is performing" is calculated through a function called a loss function [15].

There are many kinds of loss functions. Some are used for special types of data, while others are not. The simplest one is probably the Mean Absolute Error function, which simply computes the average of the differences in model prediction to the expected outputs:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y_i}| \tag{13}$$

MAE = Mean Absolute Error

n = Number of samples used to calculate loss

y_i = True value (expected output) obtained from dataset for a sample of index i

$\hat{y_i}$ = Predicted value from the model for a sample of index i

Using this information, the model will update its weights to improve its accuracy. This update is called "back-propagation." The change in each weight is called that weight's gradient, and the sum of each weight's gradient is the model's gradient. The model attempts to find the best set of weights to minimize the loss and uses the gradient to do so as the gradient function is a partial derivative of the loss function (with respect to the weights of the model). This optimization process is called gradient descent. The gradient descent and back-propagation processes will repeat until "convergence," or when the best model to fit the data has been found (i.e., the loss stops decreasing) [16]:

$$w_{new} = w - \alpha\frac{\partial}{\partial w}J(w,b) \tag{14}$$

$$b_{new} = b - \alpha\frac{\partial}{\partial b}J(w,b) \tag{15}$$

w_{new} = Updated weights

w = Weights getting updated

α = Learning rate of model (explained in Section 3.4)

J = Loss function of model

w_{new} = Updated weights

w = Weight getting updated

b_{new} = Updated biases (explained in Section 3.4)

b = Biases getting updated (explained in Section 3.4)

The model executes the processes outlined above multiple times with a lot of input data/examples. This is known as model training, and is used to iteratively minimize the loss. Once the model is optimized and trained and performs well, it can be used to make new predictions.

A Physics-Informed Neural Network, or a PINN, is a special type of neural network specifically designed to tackle problems in the physical world in which neural networks are being utilized to solve. These networks act as normal neural networks with a twist. They incorporate actual physical laws and formulae and "inform" the model of these so that the model may use these formulas to generate a better prediction than it would just by looking at the training data [17].

2 The FUSION Stellar Parameter Model

Now that we've established the basics of stellar parametrization and neural networks, it begs the question, how are these related to each other?

Well, physics is all about approximation. No one formula can account for every factor and predict any event with full accuracy. Over the years,

scientists have used different methods to make these predictions, such as mathematical formulas and statistical models. However, these models typically assume certain conditions of a physical system that may not be true. Furthermore, these models do not capture how some aspects of a star affect other parameters (they only capture the relationship between a few parameters). Take a ball dropped on Earth from five meters. Neglecting air resistance (an example of an assumption), it takes the ball one second to hit the ground. But this is not a proper model of a ball falling on Earth. Earth has air resistance, wind forces, and so much more to factor in. In the end, simply modeling a falling ball on Earth turns into an ordeal. Similarly, modeling has been particularly difficult in fields such as astrophysics, where the subjects being studied are such mind blowing distances away from Earth, and very hard to gather data on.

So, when it comes to the topic of stellar modeling, what better route to go than neural networks? After all, recent advances in machine learning techniques have greatly increased the ability of these models to understand complex relationships that are present in stars. For example, bolometric flux (see Equation 11) and gravitational binding energy (see Equation 10) are not formulaically directly related, nor do their formulae share any common variables. However, a neural network's use of intermediate layers allows it to model a multiplex relationship between the two after proper training. In addition to this, if one wanted to model a relationship between three variables instead of two, neural networks are great due to their apt ability to work with higher dimensionality. Expanding this to encompass all the stellar parameters described in Section 1.1 will allow the accuracy of each parameter prediction to surpass that of most approximating formulas by creating a holistic model of a star where all parameters are somehow related—just as how nature is.

A neural network model that can successfully and accurately model the many parameters of stars will allow researchers to have more confidence in their results, knowing that while the complex processes observed in stars may not be able to be fully modeled using mathematical and statistical models, the accuracy of their modeling will be greatly improved when using a properly trained neural network to do similar tasks. It will evolve the field of physics by accelerating research progress and making researchers' jobs less tedious and more accurate. **Being able to speedily and accurately model these subjects with minimum input data, is critical to understanding our universe!** This brings us to the project:

FUSION (FUndamental Stellar Interactions using Optimized Neural Networks):

Developing a PINN (Physics-Informed Neural Network) to accurately model stellar parameters from minimal input parameters.

Goals:

Develop a physics-informed neural network that:

a) Takes two of the following three input stellar parameters (or features) of the star one wishes to model (as the other one can be calculated using the Luminosity-Radius-Temperature relationship of stars):
   - Effective Temperature K
   - Luminosity $L/L\odot$
   - Radius $R/R\odot$

b) Calculates the remaining parameters described above and listed here:

   Outputs calculated outside of the model based on input values:
   - Diameter $D/D_\odot$ (based on the radius input)
   - Volume $V/V_\odot$ (based on the radius input)
   - Surface Area $SA/SA_\odot$ (based on the radius input)

- Great Circle Circumference $GCC/GCC_{\odot}$ (based on the radius input)
- Great Circle Area $GCA/GCA_{\odot}$ (based on the radius input)

Outputs predicted by the model in order of prediction (and whether they are a regression or classification prediction, i.e., whether the parameter is a number like the temperature of a star, or a class like the spectral class of a star):

- Absolute Bolometric Magnitude M_{bol} (regression)
- Absolute Magnitude $M/M_{\odot}$ (regression)
- Absolute Bolometric Luminosity L_{bol} (*log(W)*) (regression)
- Mass $M/M_{\odot}$ (regression)
- Average Density $D/D_{\odot}$ (regression)
- Central Pressure (*log(N/m²)*) (regression)
- Central Temperature (*log(K)*) (regression)
- Lifespan $SL/SL_{\odot}$ (regression)
- Surface Gravity (*log(g)...log(N/kg)*) (regression)
- Gravitational Binding Energy (*log(J)*) (regression)
- Bolometric Flux (*log(W/m²)*) (regression)
- Metallicity (*log(MH/MH$_{\odot}$)*) (regression)
- Spectral Class (classification, possible outputs defined in Figure 1)
- Luminosity Class (classification, possible outputs defined in Figure 2)
- Star Peak Wavelength (*nm*) (regression)
- Star Type (classification, possible outputs defined in the list after Figure 2)

Outputs calculated outside the model but based on model predictions:

- Bolometric Correction (based on the predicted bolometric and absolute magnitude values)
- Stellar Classification (based on the predicted Spectral and Luminosity classes. For example, the Sun is a G2V star—the '2' is a subclass (from 0–9, where 0 is the hottest subclass and 9 is the coolest) of the 'G' spectral class, and the 'V' is the star's luminosity class)

3 Model Architecture

The FUSION Stellar Parameter Model is created using Keras [15] with a TensorFlow [18] back-end. These are machine learning packages used to perform complex calculations involving multi-variable calculus (to calculate the model's gradient, or learning rate) and linear algebra (vectors, matrices, and tensors) to define the structure of the model and its trainable parameters[6] of a model. This model uses Keras Functional API and operates in TensorFlow's Graph Mode.

The general structure of a neural network described in Section 1.2 is a very basic one. To succeed in the task of creating the PINN outlined in Section 2, we need to employ a more complex model architecture. Here are some of the complexifying factors in the model to improve performance:

3.1 Input Layers

As shown in Figure 4, there is an input layer to each model. To achieve the vision outlined in Section 3, where each stellar parameter contributes to the calculation of the next, we cannot simply tell the model to predict the sixteen outputs from the eight inputs it is given. We must tell the model to start with eight inputs, make the first prediction based on these eight inputs, and then use that prediction to make the second prediction. This way, the first prediction will be made with eight inputs, the second with nine,

[6] Mainly the weights of the model, among others

the third with ten, and so on until all sixteen predictions are made. This has been implemented in our case by recursing the output layer (refer to Figure 4) of each parameter back to the next parameter's input layer.

3.2 The Classification Outputs

This model (neural network) sorts each star into three separate "classes." The star's spectral class, luminosity class, and the type of the star. To encourage the model to recognize the similarities and differences between each class, two techniques have been used.

1. An *Embedding Layer* has been added to the model. This allows the model to realize connections between different classes. That is to say, with the use of an embedding layer, the model will be able to realize that a star with a spectral class of K will be slightly hotter than a star with a spectral class of M [15].

   The embedding layer is not used when making predictions of the star type as some of the star type categories are unrelated (e.g., white dwarfs and red giants and brown dwarfs and supergiants as both white dwarfs and brown dwarfs don't undergo nuclear fusion). Instead, the star type data is in the format of one-hot encoded data where each star type class is represented as a value in a list [15].

2. A *Discretization Layer* has been added to the model. This layer transforms continuous data into discrete categories. It takes input data and sorts them into different "bins" or categories based on the data's characteristics. To predict spectral class, the input data is sorted into seven different bins based on the Effective Temperature of the star and to predict luminosity class the data is sorted into seven bins based on its Luminosity. The same has been done for star type, but with six bins. Each bin corresponds to a class described in Figures 1 and 2 and Section 1.1 [15].

3.3 The Regression Outputs

This model predicts thirteen regression outputs. This means that it predicts thirteen stellar parameters that are simply numerical values (such as the star's central temperature). This is where the Physics-Informed idea of Physical-Informed Neural Networks comes into play. Each one of these parameters has a formula associated with it, as shown in Section 1.1. We want to make sure the model makes predictions based on all the model inputs, not only the ones that are represented in each parameter's respective formula. However, we also want to make sure that the model respects the general physics formulae that govern the stellar parameters. To do this, we implement a custom layer that transforms the input data using the physics formula that governs the parameter it is currently evaluating. This transformed data is then mixed with the original data at the beginning processes of the neural network.

3.4 General Model Structure

To learn the optimal weights of a model, many models use Dense layers. Dense layers are used to perform linear operations on the inputs by multiplying the inputs by their weights and adding a trainable "bias" term[7]. It then runs this result through an activation[8] function to introduce non-linearity [19]:

$$y = f(Wx + b) \tag{16}$$

y = Layer output
f = Activation function
W = Input weight matrix
x = Input vector
b = Bias vector

[7] A constant term added to the inputs of an activation function used to shift the data to make the model more flexible

[8] Activation functions mitigate irrelevant data points, transform data, and only allow those important enough to be passed on to the next layer of calculations. This has the positive consequence of introducing non-linearity to the model

This model is structured to use dense layers in conjunction with three different types of activation functions distributed over the sixteen outputs:

1. Most of the model's dense layers use Gaussian Error Linear Unit (GELU) or Parametric Rectified Linear Unit (PRELU) activation functions. Both of these transform negative outputs to values close to zero, hence introducing non-linearity to the model. The difference between the two being that PRELU determines what values negative outputs should be transformed to based on the model it is being used in. GELU has also seen recent success in the machine learning industry [18].

2. For the "Star Type" parameter, where the output data is encoded in a list (see Section 3.2), we implement an activation called the Softmax [15] activation on the final output of the model. The output of the Softmax activation function would simply tell us what probability a star has of being in each class (i.e., for star type, what is the probability the star is a brown dwarf, red dwarf, etc.)

The model also uses the following:

1. Layer Normalization — normalizes the inputs of a layer to ensure they have a consistent distribution [18].

2. Gaussian Dropout — multiplies a value randomly sampled from a Gaussian distribution with random neuron outputs to increase the noise in the network [15]. Increasing noise, to a certain extent, ensures the model will not overfit (when the model memorizes the input data without actually finding the pattern that we want it to find. An overfitted model performs extremely well when fed data that it used to train, but poorly when asked to make predictions based on new data):

3. L1L2 Regularizers — another technique used to manage overfitting, regularizers will apply a penalty on the "loss" function of a model [22]. The L1L2 penalty technique uses both L1 and L2 regularization to prevent overfitting by discouraging model complexity. It sacrifices marginal amounts of accuracy for model generalization (the ability to perform well on new data):

Figure 5: Overfitting and Underfitting in a Model (Data)

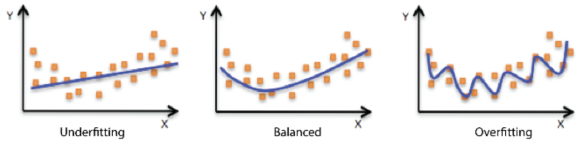


Image source: [20]

Figure 6: Overfitting and Underfitting in a Model (Complexity)

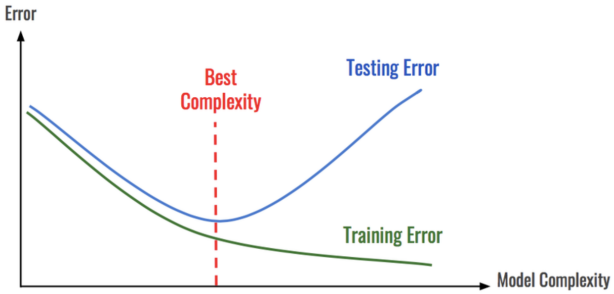


Image source: [21]
A model should be complex enough and have enough training data such that the model is balanced and performs well on new data. Very basic models result in underfitting, and too complex models result in overfitting (the model starts "memorizing" training data and performs badly on data it hasn't seen before).

$$L1 = \lambda \sum_{j=0}^{M} |W_j| \tag{17}$$

$L1$ = L1 Regularization penalty
λ = L1 Regularization Constant, 0.01
M = Number of weight parameters in the model
W_j = Value of the j^{th} weight parameter of a model

$$L2 = \lambda \sum_{j=0}^{M} W_j^2 \tag{18}$$

$L2$ = L2 Regularization penalty
λ = L2 Regularization Constant, 0.01
M = Number of weight parameters in the model
W_j = Value of the j^{th} weight parameter of a model

The loss of each predicted value is calculated using either Mean Absolute Error (MAE, refer to Section 1.2), Mean Squared Logarithmic Error (MSLE), Root Mean Squared Logarithmic Error (RMSLE), or Categorical Crossentropy (for classification outputs) [23] [15]:

$$MSLE = \frac{1}{n} \sum_{i=1}^{n} (log(1 + \hat{y}_i) - log(1 + y_i))^2 \tag{19}$$

n = Number of predictions made
$\hat{y}_i$ = Value of the i^{th} prediction made by the model during the loss calculation procedure
y_i = Value of the i^{th} "answer" or truth value during the loss calculation procedure (i.e., when the model predicts $\hat{y}_i$, it attempts to predict as close to y_i as possible)

Categorical Crossentropy is specifically used for the star type parameter that has the categorical data as this type of data is what Categorical Crossentropy was built for.

The model also implements a custom procedure that rewards the model every time its validation loss decreases. A model's validation loss looks at how well the model performed when testing itself on new data. This gives us an idea of how well the model will do in the real-world. This custom procedure is able to train a variable (or weight) in the neural network that optimizes how much of a reward the model should get for every decrease in validation loss based on training conditions for the best model performance. Essentially, the difference between the current and previous validation loss is calculated every training step, and a fraction of that difference is "rewarded" back to the model (subtracted from the model's loss). The ratio of difference in validation loss to reward is what is trained by the model. This ratio is also limited by what is called a constraint. This constraint sets limits on how much the model can be rewarded based on how much the validation loss of the model changes between training steps. This prevents the model's loss from going out of control in the early stages of training where changes in validation loss are more random, as the model has not looked at enough data to see patterns in it. A side effect of this validation loss reward is that when the validation loss increases, the model penalizes itself by *adding* to its loss.

Many hyperparameter settings (settings that "tweak" the model) were also set during the creation of this model. The optimizer used was called the Adam (or Adaptive Moment Estimation; Adam handles the optimization of the loss function during gradient descent, i.e., it tries to optimize the model by attempting to find the minimum possible loss of the model [24]) with a learning rate of 0.0001 (the model will not update its parameters greater than 0.0001 every step, see Equation 20) [15]. This allows for model training to occur at a rate where the model can learn the patterns between the testing and training data without causing many inaccuracies):

$$\frac{\mathrm{d}p}{\mathrm{d}s} < 0.0001 \tag{20}$$

$\mathrm{d}p$ = Change in model's parameters
$\mathrm{d}s$ = Change in steps (see Equation 20)

The model was trained on a batch size of 128 (it looked at 128 rows of data at a time) and 17 epochs (it looked at the whole dataset 17 times). There are

$$\frac{\#\ of\ input\ data\ rows}{batch\ size} \tag{21}$$

steps per epoch (different batches of data per epoch) [15]. The validation batch size is 32 and the model sets aside 5% of data for testing once the model is trained, 18.5% of the remaining data for validation testing used during model training,

and the rest is to actually train the model. Model validation is performed at the end of every epoch.

3.5 Data Preprocessing

Before a model is fed data to train on, the data must be transformed to a more "digestible" format for the model. This is known as *data preprocessing*. The data used to train this model is preprocessed as such:

1. The raw data are obtained from the European Space Agency's Gaia Data Release 3, which contains billions of observed parameters of celestial objects in our galaxy and beyond.
2. This database was created from observational data collected by the Gaia spacecraft, operational from 2014 to 2025. The data was mainly gathered from measurements by the spacecraft in the visible light spectrum, with some measurements also being in the near-infrared and near-ultraviolet light range. The Gaia spacecraft surveyed objects across the whole sky in the Milky Way galaxy with some objects in nearby galaxies.
3. Usable data for the model are extracted, specifically, stars for which at least two of the three parameters (luminosity, radius, and temperature) have been measured, along with measured surface gravity and metallicity.
4. Additional values, as described in Section 2, are derived from the existing data.
5. The "Star Type" data column is converted from integer indexes to one-hot encoded data.
6. The dataset rows are shuffled to randomize them for the model.
7. The training dataset is standardized using the Robust Scaler method [25], which scales data based on the median and inter-quartile range, reducing the effect of outliers on the data.
8. The data set consists of more than 15.5 million rows of "ground truth," or observed data and 403 million individual data points.
9. 5% of this data is set aside for later testing, with the results shown below analyzed from this subset.
10. 18.5% of the remaining data is reserved for validation testing during model training.
11. The rest of the data is used to train the model.

The model employs Early Stopping, a technique used to take action based on whether or not some metric is changing over time. For example, if the metric is set to validation loss, the Early Stopping procedure will look to see if the validation loss is consistently improving over time. If it is not, then the model has likely had enough training to the point where it will not get much better and training will stop. The model will also revert itself to its best "version" (when it had the lowest loss) once Early Stopping is triggered. This way, training time is saved.

Last but not least, model runtime data, model testing data, and the model data itself are all saved to be used for model performance analysis and for practical use of the model.

After prototyping by trying out different model structures and data, the best model was found. The results of this model's testing are elaborated on in the next sections.

3.6 Flowchart

Figure 7: Flow of Processes in Model

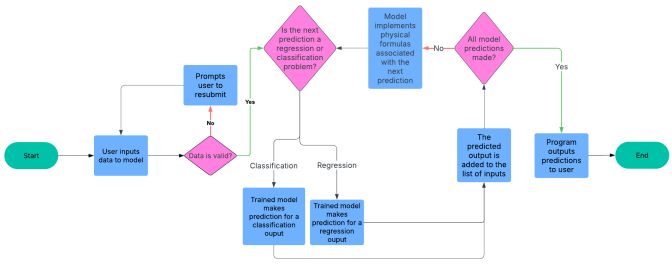


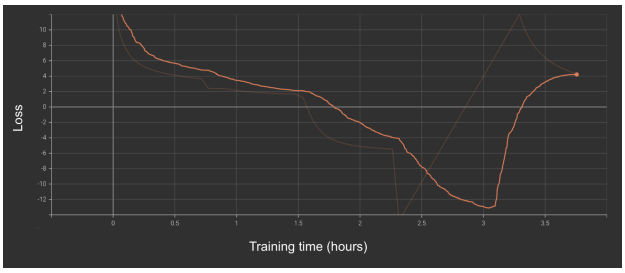
Figure created by Ansh Menghani (author) using Lucidchart.

4 Results

These results were obtained by using the model to make predictions on input data it had not seen before (the data set aside for testing described in Section 3.5).

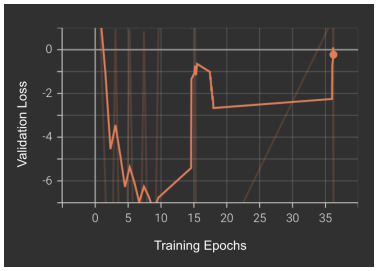
4.1 Loss Metrics

Figure 8: Model Loss vs. Time



This represents the loss, or the model's error when fed training data each time it tested itself during training. This graph is smoothed to show the loss trend. Figure created by Ansh Menghani (author) using TensorBoard.

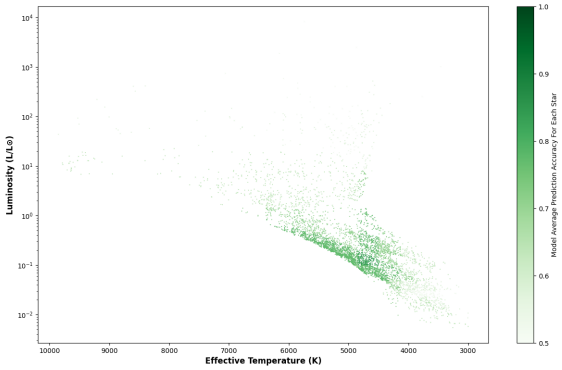
Figure 9: Model Validation Loss vs. Time



This represents the validation loss, or how well the model performed when tested with data that it hadn't seen during training. This graph has been smoothed to show the validation loss trend. Figure created by Ansh Menghani (author) using TensorBoard.

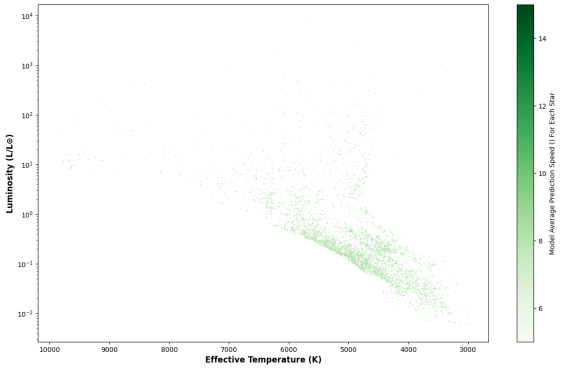
4.2 Model Accuracy and Speed

Figure 10: Two-Variable Hertzsprung-Russell Diagram of Tested Stars Color Coded by Model Average Prediction Accuracy



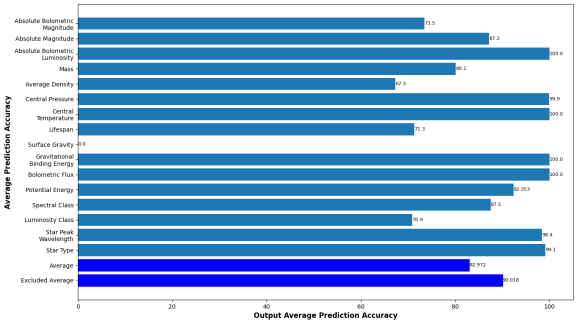
This represents the accuracy of each prediction the model makes and compares them to the star type of the star being analyzed (1.0 = 100%). Figure created by Ansh Menghani (author) using matplotlib.

Figure 11: Two-Variable Hertzsprung-Russell Diagram of Tested Stars Color Coded by Model Average Speed



This represents the speed of each prediction the model makes and compares them to the star type of the star being analyzed (1.0 = 1ms). Figure created by Ansh Menghani (author) using matplotlib.

Figure 12: Model Accuracy by Prediction Type



This represents the prediction accuracy of each stellar parameter the model predicts. Figure created by Ansh Menghani (author) using matplotlib.

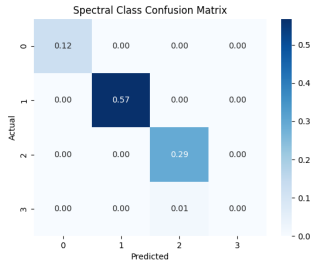
4.3 Classification Output Analysis

The following three figures show the confusion matrices, accuracy scores, recall scores, precision scores, and F1 scores that the model scored when classifying the input star into a spectral class, luminosity class, and star type. The best possible value for the latter four "scores" is 1.0. Their definitions are as follows:

Figure 13: Classification Analysis Definitions

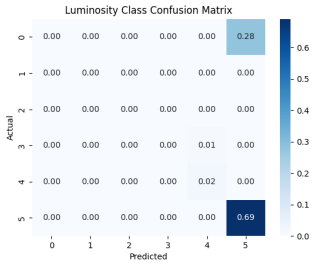
Term	Definition
Confusion Matrix	Frequency figure used to visualize predicted vs. actual values of a classification model
Accuracy Score	$\frac{\#\ of\ correct\ predictions}{number\ of\ predictions\ made}$
Recall Score	Represents how often the model correctly makes positive predictions relative to how many positive instances are in the testing dataset
Precision Score	Represents how good the the model is at predicting each class
F1 Score	Harmonic mean of recall and precision score. Provides another metric for model evaluation

Figure created by Ansh Menghani (author) using LaTeX.



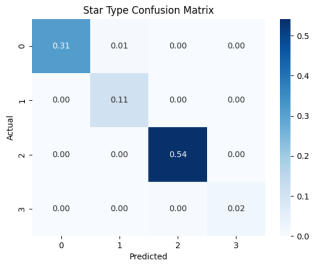
Accuracy Score	0.9860
Recall Score	0.9860
Precision Score	0.9726
F1 Score	0.9792

Figure 14: Spectral Class



Accuracy Score	0.7112
Recall Score	0.7112
Precision Score	0.5058
F1 Score	0.5912

Figure 15: Luminosity Class



Accuracy Score	0.9870
Recall Score	0.9870
Precision Score	0.9873
F1 Score	0.9871

Figure 16: Star Type

Figure 17: Analysis of Model Performance When Classifying Stars.
Figures created by Ansh Menghani (author) using scikit-learn.
Analysis and discussion of results on the next page

5 Discussion

Figures 8 and 9 show direct data during the model's training. Figure 8 shows the model's loss and Figure 9 shows its validation loss. They both follow a pretty common trend of falling at a relatively high rate at the beginning of training. The fact that the loss and validation loss becomes negative is in part due to the custom loss reward the model is getting for improvements in validation loss as described in Section 3.4. Regardless, we can see the model attempting to get as close to 0 as possible in both of these graphs (to reduce the magnitude of the loss and validation loss), which is optimal training behavior.

Figure 10 shows the accuracy of a prediction based on its location on the Hertzsprung-Russell Diagram. That is to say, if we feed the model input data from a star that has some effective temperature and luminosity and calculate the prediction accuracy of the parameters the model outputs, and then plot that on an HR diagram, we get Figure 10. The same process is repeated for Figure 11, except instead of the accuracy being plotted, we plot the prediction speed of the model.

From this data, there are a few crucial pieces of information we can conclude. For starters, the model performs much better when modeling stars on the main sequence track of the HR diagram (refer to Figure 3). Specifically, the model made *good* predictions on stars with an effective temperature between 4200 and 7000 degrees Kelvin and a luminosity between 0.01 and 1 $L/L_{\odot}$. Furthermore, the *best* selection of stars on which it performed on had an effective temperature between 4500 and 4800 degrees Kelvin and a luminosity between 0.95 and 1.4 $L/L_{\odot}$. The differences between the ranges of the effective temperature and luminosity of the *good* and the *best* stars indicate that the effective temperature of the star actually has more of an effect on model accuracy than the luminosity.

In Figure 10, we also see stars branch off into the later main sequence areas on the right side of the graph and into supergiant and hypergiant tracks at the top of the graph. As this happens, the prediction accuracy gradually decreases. This is likely due to the fact that there are fewer of these types of examples in the training dataset for the model to look at during training. This in itself is a large reason as to why these types of stars have lower modeling accuracy. However, because of the relative scarcity of later main sequence, supergiant, and hypergiant stars in the training dataset, they may also be treated as *outliers* by the model, whose influence the model actively attempts to minimize due to the large range of the training dataset.

Figure 11 is very similar to Figure 10, except that it models prediction speed rather than accuracy by star on the HR diagram. The result from this is very simple: star position on the HR diagram has little to no affect on the prediction speed. This result is expected as the model's calculation speed should not be affected by any input data within a reasonable range.

Figure 12 shows how accurately the model predicted each stellar parameter's values (e.g., the model predicted any given star's potential energy with an accuracy of 92.353%). The accuracy is calculated by dividing the number of correct predictions the model makes by the amount of data it was tested on per parameter. A correct prediction is considered as a prediction that falls within $\pm$ 5% of the true value. Figure 12 shows many interesting trends. Most obviously, the fact that the surface gravity output has a *0%* accuracy. It is thought that this is due to the fact that surface gravity is logarithmically scaled, so a prediction of 1 and a truth value of 2 would yield a false prediction. This way, none of the predictions fell within the $\pm$ 5% range. It is also possible that this result is due to overfitting problems.

Another interesting detail is that, in Figure 12, there are many more "lower" accuracies in the first half of parameter prediction than in the second half. In the first nine predictions, four of them fell under an average accuracy of 85%, while in the last seven, only one did. This indicates the method the model implements to recurse outputs into inputs for the next parameter prediction (as explained in Section 3.4) works!

The average accuracy of the model is 82.972%. However, if we simply take out the

two lowest accuracies by parameter (i.e., the *outliers*, the Surface Gravity and Average Density parameters), we see nearly a 7.5% jump in accuracy to a solid 90.018%. The average speed per prediction of this model is 8 *milliseconds*.

Lastly, Figures 14, 15, and 16 represent the "Analysis of Model Performance When Classifying Stars." Figures 14 and 16 (the spectral class and star type figures respectively) exhibit excellent scores (refer to the tables), with each value close to the optimal value of 1. The confusion matrices on these show a solid trend in predicting the right value with testing data. The matrices are only 4x4 despite there being seven classes for spectral class and six for star type because each one of them had data on only four of their classes. Specifically, the model was trained to predict spectral classes of M, K, G, and F. It was trained to predict Star Types of red dwarfs, white dwarfs, main sequence, and supergiants.

Despite the excellent scores in Figure 14 and 16, Figure 15 is less. This may be the case as it has six classes (Luminosity Classes D, Ib, II, III, IV, V) instead of the four the spectral class and star type have, which causes an issue because temperature and luminosity are very much related and an output with four classes (the spectral class) being fed into the model to predict the luminosity class (which has six possible classes) will limit the model's ability to properly sort stars into all all six luminosity classes.

In comparison with other research done in this space, this model is different and an improvement due to the fact that other stellar models use mathematical and statistical methods and detailed measurements to model stars. This model sees more complex relationships between stellar parameters and can make predictions from minimal input data. In addition to this, the model predicts many more parameters than the typical effective temperature, luminosity, radius, surface gravity, and metallicity than typical models currently do. It also combines spectral modeling as one of the predictions, which has been the focus of many studies.

This model has many practical applications:

1. For starters, this project further expands machine learning and deep learning into the field of physics as a whole. This is a benefit for the whole field as it adds to the subfield of computational physics. It improves upon current mathematical and statistical methods of stellar modeling, capturing complex relationships between all parameters and speedily delivering a wholistic simulation of a star from minimal input parameters.

2. This model is the first model that does the job of sixteen models (i.e., if each stellar parameter were to be modeled by a separate model) with similar accuracy metrics and the same speed as one model, allowing for detailed analysis of large datasets with high speeds.

3. This tool can be used to simulate star systems unfathomably far from Earth. Due to their distance, it is very difficult to get diverse data on them, so we must rely on simulations to find this. For many of these stellar parameters, this model can be used to simulate them with an accuracy arguably better than traditional mathematical and statistical methods, as described in Section 2.

4. The machine learning techniques developed and used in this model (especially the custom ones) can be replicated in other projects across the field of machine learning and neural networks to greatly improve them.

5. The program can also be used to speedily simulate many stars at once in the form of big data. This will allow scientists to analyze many stars at once in great detail for data aggregation and research purposes while saving the time of simulating/calculating each parameter individually.

6. The model can be used to simulate certain patches of the sky or objects that scientist want to see if they are worth studying. This way, the software will allow researchers to get a preliminary model of interesting study subjects so that they can decide where they

want to spend their resources (such as telescope or spacecraft resources).

7. This model can be used to find previously unknown relations between different stellar parameters that may not currently share a simple mathematical relation. This will allow scientists to confirm their results.

6 Conclusion

This project had very intriguing outcomes and the goals described in Section 2 have been met. As described in Section 5, it has many possible practical applications. These practical applications can be used to conduct further research. Naturally, there are also limitations. The model doesn't predict rare star types so well due to the scarcity of data on them. In the future:

- Parameter accuracy will be worked on and increased
- A more diverse dataset will be used to optimize the model for special types of celestial objects such as neutron stars
- A user interface will be built and the application will be made available for the public to use

Overall, the project has provided results that can be used to further science.

7 Acknowledgments

I would like to extend my sincere thanks to my family for supporting me throughout this research. I would also like to thank Mr. Charlie Payne for inspiring this project. And I would most certainly like to thank you for reading this paper.

8 The Author and The Code

My name is Ansh Menghani, and as of January 2025, I am a junior in high school at the North Carolina School of Science and Mathematics. My main passions are physics and computer science and I love cooking, music, stargazing, and reading. The GitHub link to all the code, explanations, material used in this project, installation instructions, and repository cloning instructions can be found here: `https://www.github.com/anshmenghani/Fusion/`. Contact me at `ansh.menghani@gmail.com`

References

[1] astro.princeton.edu. https://www.astro.princeton.edu/~gk/A403/constants.pdf. [Accessed 14-10-2024].

[2] The Mass-Luminosity Relationship — Astronomy 801: Planets, Stars, Galaxies, and the Universe — e-education.psu.edu. https://www.e-education.psu.edu/astro801/content/l7_p3.html. [Accessed 13-10-2024].

[3] Vik Dhillon. vik dhillon: phy213 - the equations of stellar structure - minimum value for central pressure of a star — vikdhillon.staff.shef.ac.uk. https://vikdhillon.staff.shef.ac.uk/teaching/phy213/phy213_minp.html. [Accessed 16-10-2024].

[4] David Weinberg. A162, Lecture 6 — astronomy.ohio-state.edu. https://www.astronomy.ohio-state.edu/weinberg.21/Intro/lec6.html. [Accessed 15-10-2024].

[5] Stellar Lifetimes — hyperphysics.phy-astr.gsu.edu. http://hyperphysics.phy-astr.gsu.edu/hbase/Astro/startime.html. [Accessed 11-11-2024].

[6] Surface Gravity on Jupiter — pages.uoregon.edu. http://pages.uoregon.edu/jimbrau/astr121/Notes/Jupiter/jupitergrav.html#:~:text=The%20surface%20gravity%20on%20a,m1m2%20%2F%20R. [Accessed 09-11-2024].

[7] Gravitational Binding Energy of a Uniform Sphere — Wolfram Formula Repository — resources.wolframcloud.com. https://resources.wolframcloud.com/FormulaRepository/resources/Gravitational-Binding-Energy-of-a-Uniform-Sphere. [Accessed 09-11-2024].

[8] Britannica. Stefan-Boltzmann law — Definition & Facts — Britannica — britannica.com. https://www.britannica.com/science/Stefan-Boltzmann-law, December 2024.

[9] Jesse S. Allen. The Classification of Stellar Spectra — star.ucl.ac.uk. http://www.star.ucl.ac.uk/~pac/spectral_classification.html. [Accessed 16-11-2024].

[10] Types - NASA Science — science.nasa.gov. https://science.nasa.gov/universe/stars/types/. [Accessed 16-11-2024].

[11] Richard Powell. HRDiagram.png - Wikipedia — en.wikipedia.org. https://commons.wikimedia.org/wiki/File:HRDiagram.png, May 2011. [Accessed 27-11-2024].

[12] Supervised vs Reinforcement vs Unsupervised - GeeksforGeeks — geeksforgeeks.org. https://www.geeksforgeeks.org/supervised-vs-reinforcement-vs-unsupervised/, September 2024.

[13] Facundo Bre, Juan M. Gimenez, and Víctor D. Fachinotti. Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings*, 158:1429–1441, January 2018.

[14] Neural Networks 101: Understanding the Basics of This Key AI Technology — online.nyit.edu. https://online.nyit.edu/blog/neural-networks-101-understanding-the-basics-of-key-ai-technology, December 2023.

[15] François Chollet et al. Keras. https://keras.io, 2015.

[16] Navaneeth Sharma. Guide to Gradient Descent: Working Principle and its Variants - DataMonje — datamonje.com. https://datamonje.com/gradient-descent-and-variants/. [Accessed 07-02-2025].

[17] Ian Henderson. Physics Informed Neural Networks (pinns): An intuitive guide. https://towardsdatascience.com/physics-informed-neural-networks-pinns-an-intuitive-guide-fff138069563, October 2022.

[18] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[19] Valentine Shkulov. Neural Network Layers: All You Need Is Inside Comprehensive Overview — HackerNoon — hackernoon.com. https://hackernoon.com/neural-network-layers-all-you-need-is-an-inside-comprehensive-overview, July 2023.

[20] https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html. Accessed: 2025-02-13.

[21] Overfitting, bias-variance and learning curves. https://rmartinshort.jimdofree.com/2019/02/17/overfitting-bias-variance-and-leaning-curves/, February 2019. Accessed: 2025-02-13.

[22] Dhaval Taunk. L1 vs L2 Regularization: Which is better — medium.com. https://medium.com/analytics-vidhya/l1-vs-l2-regularization-which-is-better-d01068e6658c, March 2020.

[23] Diana Nersesyan. Evaluation Metrics for Regression: A Comprehensive Overview. https://medium.com/@diana_nersesyan/understanding-evaluation-metrics-for-regression-a-comprehensive-overview-56450a5bea49, July 2023.

[24] What is adam optimizer? https://www.geeksforgeeks.org/adam-optimizer/, October 2020. Accessed: 2024-11-11.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

This work has made use of data from the European Space Agency (ESA) mission *Gaia* (https://www.cosmos.esa.int/gaia), processed by the *Gaia* Data Processing and Analysis Consortium (DPAC, https://www.cosmos.esa.int/web/gaia/dpac/consortium). Funding for the DPAC has been provided by national institutions, in particular the institutions participating in the *Gaia* Multilateral Agreement.

- Gaia Collaboration et al. (2016b): The Gaia mission (provides a description of the Gaia mission including spacecraft, instruments, survey and measurement principles, and operations);
- Gaia Collaboration et al. (2023j): Gaia DR3: Summary of the contents and survey properties.