Modelling stars with Gaussian Process Regression – I: Augmenting Stellar Model Grid

Tanda Li,[1]★ Guy R. Davies,[1]† Alex Lyttle,[1] Lindsey Carboneau,[1] and A. N. Others[1]
[1] *School of Physics and Astronomy, University of Birmingham, Birmingham, B15 2TT, United Kingdom*

Accepted XXX. Received YYY; in original form ZZZ

ABSTRACT
Grid-based modelling is widely used for estimating stellar parameters. However, theoretical model grid is sparse. This paper demonstrates an application of Gaussian Process Regression (GPR), which transfer a sparse model grid to a continuing function. We trained a GPR model with a model grid to learn the function from 5 independent fundamental inputs (Mass, fractional age, initial metallicity, initial helium fraction, and the mixing-length parameter) to 6 model outputs (age, seismic large separation, effective temperature, surface gravity, radius, and surface metallicity). The GPR function was then validated and the typical standard deviations of GPR predictions are 1% for the age, 0.5μHz for the seismic large separation, 12K for the effective temperature, 0.004dex for the surface gravity, 0.008$R_\odot$ for the radius, and 0.008dex for the surface metallicity. This indicates that GPR can be used to augmenting stellar grids and furthered model stars. We lastly applied the GPR function for modelling the Sun-as-a-star. The GPR function gives nicer statistical estimates than grid-based modelling does.

Key words: keyword1 – keyword2 – keyword3

1 INTRODUCTION

This will be the intro. (ask Guy to write some general idea of Gaussian Process and the GPR model).

2 THEORETICAL MODEL GRID

We built up a model grid as the training data for GPR models. We aimed to cover stars with approximate solar mass on the main-sequence and the subgiant phases. Thus, the mass range is set up as 0.8 – 1.2$M_\odot$, and we computed each stellar evolutionary track from the Hayashi line and terminated at the base of red-giant branch where $\log g$ = 3.6 dex. However, we only use models after the zero-age-main-sequence (ZAMS), which is defined as where the core hydrogen burning contributes over 99.9 percent of total luminosity of the star. The gird includes four independent model inputs: stellar mass (M), initial helium fraction ($Y_{\rm init}$), initial metallicity ([Fe/H]), and the mixing-length parameter ($\alpha_{\rm MLT}$). Details of the computations are summarised in Table 1. Input parameters of our primary training data is uniformly spaced except the input metallicity, which is double dense above 0.2 *dex*. We also computed an extra grid that fit in between the primary training data for $M > 1.05M_\odot$. Because evolutionary tracks in this mass range have the so-called 'hook' at the late main-sequence stage and relatively difficult to model. Lastly, we computed 4,880 tracks with input parameters that are randomly sampled in the grid ranges for validating GPR models. The evolution time step was mainly controlled by the set-up tolerances on changes in surface effective temperature and luminosity. We also saved structural models for computing theoretical oscillation models.

2.1 Stellar models and input physics

We used Modules for Experiments in Stellar Astrophysics (MESA, version 12115) to establish a grid of stellar models. MESA is an open-source stellar evolution package which is undergoing active development. Descriptions of input physics and numerical methods can be found in Paxton et al. (2011, 2013, 2015). We adopted the solar chemical mixture [$(Z/X)_\odot$ = 0.0181] provided by Asplund et al. (2009). The initial chemical composition was calculated by:

$$\log(Z_{\rm init}/X_{\rm init}) = \log(Z/X)_\odot + [{\rm Fe/H}]. \qquad (1)$$

We used the MESA $\rho - T$ tables based on the 2005 update of OPAL EOS tables (Rogers & Nayfonov 2002) and OPAL opacity supplemented by low-temperature opacity (Ferguson et al. 2005). The MESA 'simple' photosphere were used as the set of boundary conditions for modelling the atmosphere. The mixing-length theory of convection was implemented, where $\alpha_{\rm MLT} = \ell_{\rm MLT}/H_p$ is the mixing-length parameter. We also applied the MESA predictive mixing scheme (Paxton et al. 2018, 2019) in the model computation. Atomic diffusion of helium and heavy elements was also

★ E-mail: t.li.2@bham.ac.uk
† E-mail: G.R.Davies@bham.ac.uk

Table 1. Stellar model computations for training and validating sets.

	Primary Grid	
Input Parameter	Range	Increment
M $[M_\odot]$	0.80 – 1.20	0.01
[Fe/H] [dex]	-0.5 – 0.2/0.2 – 0.5	0.1/0.05
$Y_{\rm init}$	0.24 – 0.32	0.02
$\alpha_{\rm MLT}$	1.7 – 2.5	0.2
	Additional Grid	
Input Parameter	Range	Increment
M $[M_\odot]$	1.055 – 1.195	0.01
[Fe/H] [dex]	0.25 – 0.45	0.1
$Y_{\rm init}$	0.25 – 0.31	0.02
$\alpha_{\rm MLT}$	1.8– 2.4	0.2
	Off-grid Models	
GP Model	Number of Tracks	
2D	44	
3D	174	
5D	4880	

taken into account. MESA calculates particle diffusion and gravitational settling by solving Burger's equations using the method and diffusion coefficients of Thoul et al. (1994). We considered eight elements (^{1}H, ^{3}He, ^{4}He, ^{12}C, ^{14}N, ^{16}O, ^{20}Ne, and ^{24}Mg) for diffusion calculations, and had the charge calculated by the MESA ionization module, which estimates the typical ionic charge as a function of T, ρ, and free electrons per nucleon from Paquette et al. (1986). The MESA inlist used for the computation is available on `https://github.com/litanda/mesa_inlist`.

2.2 Oscillation models and seismic $\Delta\nu$

Theoretical stellar oscillations were calculated with the GYRE code (version 5.1), which was developed by Townsend & Teitler (2013). And we computed radial modes (for $\ell = 0$) by solving the adiabatic stellar pulsation equations with the structural models generated by MESA. We computed a seismic large separation($\Delta\nu$) for each model with theoretical radial modes to avoid the systematic offset of the scaling relation. We derived $\Delta\nu$ with the approach given by White et al. (2011), which is a weighted least-squares fit to the radial frequencies as a function of n.

3 GAUSSIAN PROCESS REGRESSION MODEL

GP models can be applied as probabilistic models to regression problems. Here we will use the GP model to generalise a grid of stellar models to a continuous and probabilistic function that maps inputs (i.e., initial mass, chemical composition, etc.) to observable quantities (i.e., effective temperature, surface gravity, radius, etc.). We aim to use the GP model as a non-parametric emulator, that is emulating the comparatively slow calls to models of stellar evolution. We adopted the a tool package named GPyTorch, which is a GP framework developed by (Gardner et al. 2018). GPyTorch is a Gaussian process library implemented using PyTorch. We adopted this tool package because it provides significant GPU acceleration, state-of-the-art implementations of the latest algorithmic advances for scalability and flexibility, and easy integration with deep learning frameworks. Source codes and detailed introductions could be found on `https://gpytorch.ai`

3.1 Gaussian Process Application

We start with a grid of stellar models containing N models with a label we want to learn, for example model effective temperature, which we will denote with the general symbol $\mathbf{y}$, and a set of input labels $\mathbf{X}$ (e.g., mass, age, and metallicity). We can use a GP to make predictions of the effective temperature (or y) for additional input values given by $\mathbf{X}_\star$. The vector $\mathbf{y}$ is arranged $\mathbf{y} = (y_i, ..., y_N)^T$ where the subscript label references the stellar model. The input labels are arranged into a $N \times D$ matrix where D is the number of input dimensions (e.g., $D = 3$ for mass, age, and metallicity) so that $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_N)^T$ where $\mathbf{x_i} = (x_{1,i}, ..., x_{D,i})^T$. The matrix of additional inputs $\mathbf{X}_\star$ has the same form as $\mathbf{X}$ but size $N_\star \times D$.

Rasmussen and Williams (!REF!), from which our description below is based, define a GP as a collection of random variables, where any finite number of which have a joint Gaussian distribution. In general terms, GP's may be written as

$$y(\mathbf{x}) \sim \mathcal{GP}\left(m(\mathbf{x}), \mathbf{\Sigma}\right), \tag{2}$$

where $m(\mathbf{x})$ is some mean function, and $\mathbf{\Sigma}$ is some covariance matrix. The mean function controls the deterministic part of the regression and the covariance controls the stochastic part. The mean function defined here could be any deterministic function and we will label the additional parameters, or hyperparameters, ϕ. Each element of the covariance matrix is defined by the covariance function or *kernel function* k which has hyperparameters θ and is given by,

$$\mathbf{\Sigma}_{n,m} = k(\mathbf{x}_n, \mathbf{x}_m), \tag{3}$$

where the inputs $\mathbf{x}_i$ are D-dimensional vectors and the output is a scalar covariance.

As a GP is a collection of random variables, where any finite number of which have a joint Gaussian distribution, the joint probability of our data $\mathbf{y}$ is

$$p(\mathbf{y}|\mathbf{X}, \phi, \theta) = \mathcal{N}(m(\mathbf{X}), \mathbf{\Sigma}). \tag{4}$$

If we want to obtain predictive distributions for the output $\mathbf{y}_\star$ given the inputs $\mathbf{X}_\star$ the joint probability distribution of $\mathbf{y}$ and $\mathbf{y}_\star$ is Gaussian and given by

$$p\left(\begin{bmatrix}\mathbf{y}\\ \mathbf{y}_\star\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\mathbf{X}\\ \mathbf{X}_\star\end{bmatrix}, \begin{bmatrix}\mathbf{\Sigma} & \mathbf{K}_\star\\ \mathbf{K}_\star{}^T & \mathbf{K}_{\star\star}\end{bmatrix}\right), \tag{5}$$

where the covariance matrices $\mathbf{\Sigma}$ and $\mathbf{K}$ are computed using the kernel function so that

$$\mathbf{\Sigma}_{n,m} = k(\mathbf{X}_n,\ \mathbf{X}_m), \tag{6}$$

which is an $N \times N$ matrix.

$$\mathbf{K}_{\star\, n,m} = k(\mathbf{X}_n,\ \mathbf{X}_{\star\, m}), \tag{7}$$

which is an $N \times N_\star$ matrix, and finally

$$\mathbf{K}_{\star\star\, n,m} = k(\mathbf{X}_{\star\, n}, \mathbf{X}_{\star\, m}), \tag{8}$$

which is an $N_\star \times N_\star$ matrix. The predictions of $\mathbf{y}_\star$ are again a Gaussian distribution so that,

$$\mathbf{y}_\star \sim \mathcal{N}(\hat{\mathbf{y}}_\star,\ \mathbf{C}), \tag{9}$$

where

$$\hat{\mathbf{y}}_\star = m(\mathbf{X}_\star) + \mathbf{K}_\star^T\ \mathbf{\Sigma}^{-1}\ (\mathbf{y} - m(\mathbf{X})), \tag{10}$$

and

$$\mathbf{C} = \mathbf{K}_{\star\star} - \mathbf{K}_{\star}^{T}\,\mathbf{\Sigma}^{-1}\,\mathbf{K}_{\star}. \tag{11}$$

At point we can make predictions on model properties given a grid of stellar models using equation 9. But these predictions will be poor unless we select sensible values for the form and hyperparameters of the mean function and covariance function. In the following section we detail a number of kernel functions that will be tested against the data. We will then discuss the method for determining the values of the hyperparameters to be used.

3.2 GP Model Inputs and Outputs

We aim to derive stellar parameters based on fundamental inputs of the model grid. As mentioned in Section 2, our model grid has five independent fundamental inputs, says, mass, initial metallicity, initial helium fraction, mixing-length parameter, and the age. Among them, the age ranges of individual evolutionary tracks significantly vary with the mass and other model inputs and hence are not ideal as the GP model inputs. The fractional age is an option. However, the evolution of global parameters around the 'hook' and the turn-off point performs sharp changes within a short time scale as shown in the left panel of Figure 1. The sharp features are hard to learn and hence poorly predicted by GP models. To overcome this issue, we followed MIST(add ref here) and introduce an Equivalent Evolutionary Phase (EEP here after) to replace the age as fundamental inputs. Note that our definition of EEP is different from MIST. On each evolutionary track, we compute the displacement of a evolving step n on the $T_{\rm eff} - \log g$ as

$$\delta d_n = ((T_{\rm eff,n-1} - T_{\rm eff,n})^2 + (\log g_n - \log g_{n-1})^2))^f\,, \tag{12}$$

and the total displacement of a data point n

$$d_n = \sum_{i=0}^{i=n} \delta d_i. \tag{13}$$

The normalised d_n (to a 0 –1 range) is defined as EEP. On the same evolutionary track, EEP equals to 0 at the ZAMS and 1 at log = 3.6 dex. The factor f in Eq. 12 is for modulating EEP to avoid large gap in the parameter space, and we found that $f = 0.18$ gives the best data distribution. **add figures in appendix** We illustrated two GPR models with the fractional age and EEP as inputs in Figure **??** which shows the advantages of EEP. It can be seen in the top graph that the effective temperature evolution presents a sharp hump around 0.7 (the hook) and quickly drops in the last 10% lifetime. Using EEP instead of fractional age significantly smoothes the curve and make it easer to fit. As we compare in Figure 1, using EEP instead of fractional age gives a much smoother change at the hook and turn-off points.

We summary our selections of GPR model inputs and outputs as below.

- GPR model inputs and their dynamic ranges:
  Mass ($M = 0.8 - 1.2 M_{\odot}$)
  Equivalent Evolutionary Phase ($EEP = 0 - 1$)
  Initial metallicity ($[{\rm Fe/H}]_{\rm init}$ = -0.5 – 0.5)
  Initial helium fraction ($Y_{\rm init} = 0.24 - 0.32$)
  Mixing-length parameter ($\alpha_{\rm MLT} = 1.7 - 2.5$)
- GPR model outputs:
  Effective temperature ($T_{\rm eff}$)
  Surface gravity ($\log g$)
  Radius (R)
  The large spacing ($\Delta\nu$)
  Surface metallicity ($[{\rm Fe/H}]_{\rm surf}$)
  Age (τ)

Thus, our GPR model can be described as

$$\text{Outputs} = f(M, t', ({\rm Fe/H})_{\rm init}, Y_{\rm init}, \alpha_{\rm MLT}). \tag{14}$$

3.3 Training Procedure

3.3.1 Data Selection

We have three types of data for training, validating, and testing GP models. The model grid is apparently the training data and the off-grid models are divided into validating and testing datasets by half to half. Note that validating and testing data are not on same evolutionary tracks so they do not share any information. Validating dataset is for validating the GP model in the training process and is mainly used for early stopping, which is a form of regularisation for avoiding overfitting. We choose off-grid models but not on-grid models as validating data because the grid is equally spaced. Without additional information between grid points, an optimiser could either use a smooth function or a periodic function to fit the data and find no obvious differences in the likelihood. We describe how we use validating data in Section 3.3.2. Lastly, testing dataset is independent on the training process and used for evaluating the final GP models.

Because the computational and memory complexity exponentially increase with the number of data involved in the Gaussian Process, only a subset of the model grid can be adopted. Moreover, the data density are not same at different evolutionary stages due to the MESA step-control strategy: stellar models are dense at the main-sequence and the red-giant phases but quite sparse on subgiant stage. The sampling method is hence critical to the performance of GP models. Firstly, we want the data to uniformly cover the parameter space for the best efficiency. Secondly, we need to highly weight models at phases where sharp changes present, e.g., models around the hook and turn-off points. We tested a few methods and found that using the displacement (δd_n) defined in Eq. 12 to weight sampling meets the above two requirements.

3.3.2 Training, Validating, and Testing GP Models

We train GP models as a regression problem. We develop our training process based on two standard examples (Simple GP Regression and Stochastic Variational GP Regression) in the GPYTORCH packages. We test different approaches to determine the training method. Details are described and discussed in the next Section. In the training process, we validate the GP model in every iteration and save it as the best model when the validation index decreases. The training terminate when the validating results stop improving for 300 iterations. Lastly, the best GP model is tested with the testing dataset.

Here we discuss the method for validating and testing a GP model. We do not use popular methods, such like RME because the GP model performances are not uniform across the whole parameter space due to different evolutionary features. We illustrate this in Figure 2 with a 2D GP model for $T_{\rm eff}$. As it can be seen that, the function for the hook area ($M \geq 1.05 M_{\odot}$ and $EEP \leq 0.7$) is more complex than that for other smooth regions. The evolutionary features in this particular area are hence relatively difficult to learn by the GP model, leading to obvious different accuracy levels (as shown in the bottom graph). For other output parameters, surface gravity and radius are similar to the case of effective temperature:

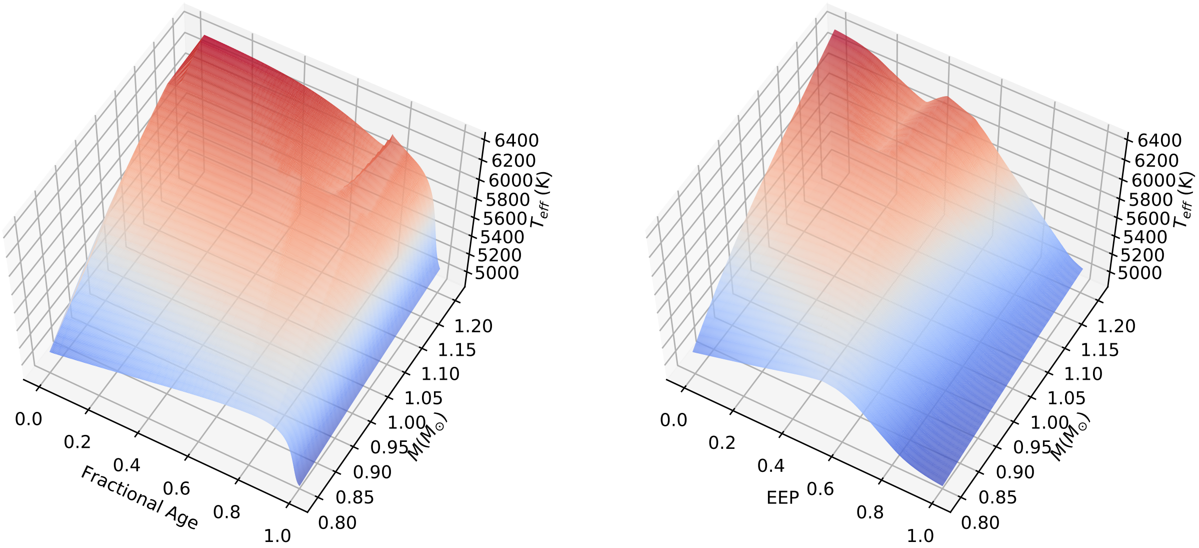

Figure 1. Surface plots of effective temperature as a function of fractional age (left) and EEP(right). It can be seen that using EEP instead of fractional age as the input gives much smoother features at the hook and turn-off points.

obviously differences between the 'hook' and other area. The evolutionary feature of surface metallicity is not significantly affected by the hook, but the testing errors of its GP model are obviously higher at the early subgiant phase for high-mass tracks. This is because high-mass tracks maintain shallow convective envelope and hence have strong diffusion effect during the main-sequence stage. Their surface metallicities at then end of the main-sequence are generally much lower than the initial value. When high-mass models evolve to the early subgiant phase, the quick expansion of the surface convective envelope brings back the settling heavy elements to the surface, resulting in a sharp raise of the surface metallicity. When there is a sub-region in the parameter space where GP models always peforme worse than other areas, a RME value is not suitable. We hence need a validation method that reflects the general accuracy of a GP model as well as its performance in these particular regions. To find a proper method for validation, we examine the error distribution as shown at the bottom of Figure 2. For the case of effective temperature, errors of most loss-mass data follows a Gaussian distribution. Data in the hood-affected regions is ~ 10% of the total sample. This is a small proportion hence does not significantly affect the main feature but form long tails on both sides. For validating the general accuracy, a standard deviation is suitable. The long tail could be quantity by two cumulative values: one at 95% indicates the median error of the these values, the other at 99.7% to describe the scatter. Thus, we use the sum of three cumulative values (at 68%, 95%, and 99.7%) of absolute validation errors as an error index to qualify a GP model in validating and testing progresses. For the case in Figure 2, cumulative values at 68%, 95%, and 99.7% are 1.07, 4.86, and 11.05K, which give an error index equals to 16.98.

3.4 Primary Tests and Settings of Training

Before training the grid, we need to set up the GP model including mean function, kernel, likelihood function, loss function, and the optimiser. As there are not many relevant studies, we test how different options affects the GP models and decide which to use. These primary tests are done with simple 2-demission (2D) GP models, which have two inputs: M and EEP and can be described as $p = f(M, EEP)$. Data for training 2D models (2D data here after) are selected from the primary grid with fixed [Fe/H]$_{\rm init}$ (0.0), $Y_{\rm init}$ (0.28), and $\alpha_{\rm MLT}$ (2.1). The training dataset contents 41 evolutionary track and 24,257 data points. We also computed 44 evolutionary tracks with same input [Fe/H]$_{\rm init}$, $Y_{\rm init}$, and $\alpha_{\rm MLT}$ but random M for validating and testing the GP models.

The 2D model can be trained with the Exact GP approach because the training data size is approximate 20,000. We start with the Simple GP Regression example (https://docs.gpytorch.ai/en/stable/examples/01_Exact_GPs/Simple_GP_Regression.html) to develop our training script.

3.4.1 *Mean Function*

We start with testing different mean functions. As demonstrated in Figure 1, the map between model inputs and outputs show various of features in different regions of the parameter space: models with relatively high mass have complex curvatures around the hook and the turn-off point while other models follow smooth functions. Although the GP model does not significantly affected by the choice of the mean function due to the flexibility of kernels, we find that a simple mean function (constant or linear) leads to a long time for training and a significant increase of the complexity of kernels. We hence applied a more complex mean function which is flexible enough to map smooth as well as curved features. A Neural Network is well suitable. To cover higher-demissions GP model, we adopt an architecture includes 6 hidden layers and 128 nodes per layer. All layers apply the linear transformation to the incoming data. We apply the element-wise function (Elu) because it give relatively smooth mean functions. (add ref for NN and Elu)

3.4.2 *Likelihood and Loss Function*

We then test for the likelihood and loss function. Our training object is a theoretical model grid, there is hence no observed uncertainty for each data point. However, a tiny random uncertainty exists due

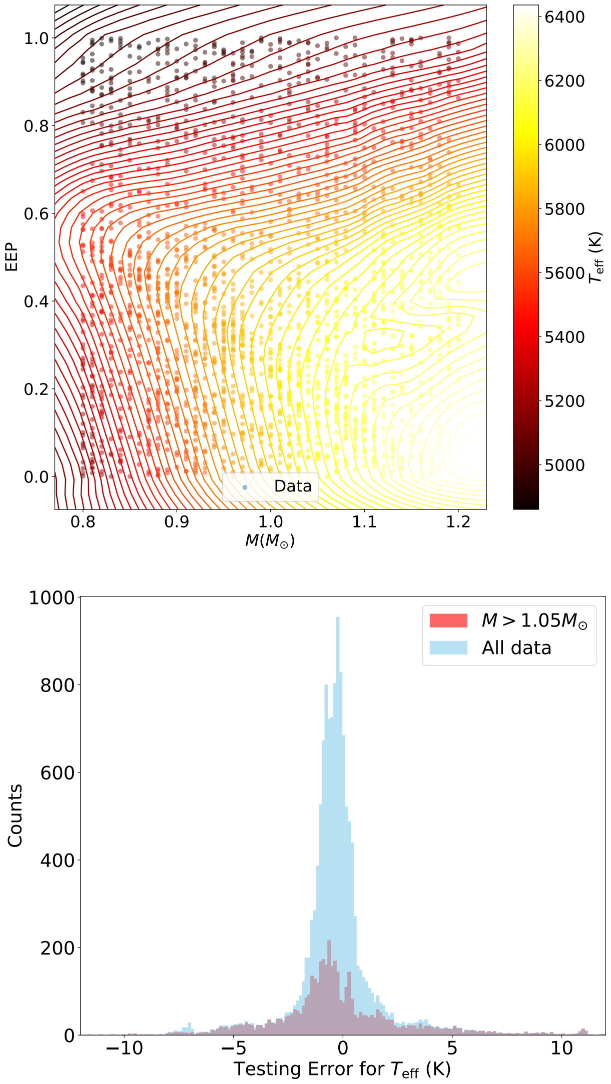

Figure 2. Top: The 2D GP model for $T_{\rm eff}$. Bottom: probability distributions of validating errors of the GP model.

to the approximations in the MESA numerical method. The noise model can be assumed as a Gaussian function with a very small deviation. We hence applied the standard Gaussian Likelihood function in GPyTorch. A Likelihood specifies the mapping from latent function values $f(X)$ to observed labels y. We adopt the the standard likelihood for regression which assumes a standard homoskedastic noise model whose conditional distribution is

$$p(y|f(x)) = f + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma^2), \tag{15}$$

where σ is a noise parameter. Given the random uncertainty is small, we set up a small and fixed noise parameter and run a few tests. However, we found that it makes the GP models hard to converge and sometimes lead to obvious overfitting (poor behaviour between grid). When we set up this noise parameter free and start with a large initial value, it reduces to a small number anyway in the training progress because it is data-driven. For these reasons, we decide not put strict constraint for or prioritise this noise parameter and let the data determine. In practice, we set up a loose upper limit ($\sigma < 0.1$) for the noise parameter. However, it should be note that a GP model with a larger noise parameter can not be a proper description for the training data even if it gives good validating or testing errors. Because of this, we only use GP model with a noise parameter $\lesssim 10^{-4}$. The the loss function is simply the exact marginal log likelihood.

3.4.3 *Optimiser*

With a Neural Network mean function and Gaussion likelihood function, we then run tests to decide the optimiser. We mainly compare two optimisers named SGD and Adam. Here SGD refers to Stochastic Gradient Descent, and Adam is a combination of the advantages of two other extensions of stochastic gradient descent, specifically, Adaptive Gradient Algorithm and Root Mean Square Propagation. The SGD optimiser in the Troch packages involved the formula from Sutskever et al. (2013), which makes it possible to train using stochastic gradient descent with momentum uses a well-designed random initialisation and a particular type of slowly increasing schedule for the momentum parameter. The application of momentum in SGD could improve its efficiency and make it less likely to stuck in local minimums. On the other hand, the Adam optimiser includes the AMSGrad variant developed by Reddi et al. (2018) to improve its weakness in the convergence to an optimal solution. With these new developments, the two optimisers give very similar results. We finally choose Adam because it works more efficiently and stable than the SGD. We adaptive learning rate instead of a fixed value. All training starts with a learning rate of 0.01 and decreases by a factor of 2 when the loss value stop reducing in the previous 100 iterations.

3.4.4 *Early stopping*

We set up an early stoping method to determine when to terminate the training progress based on the validation index introduced in Section 3.3.2. In our training progress, we validate the GP model every iteration and save the current model if the validating results is by far the best. When an optimal solution is found, the validating errors stop decreasing and start increasing at some point when overfiting occurs. To be efficient and prevent overfitting, we terminate the training progress when the validating index does not decrease in the previous 300 iterations. The last saved GP model will be adopted.

3.4.5 *GP kernels*

With the above settings, we lastly test for selecting the best kernel for this work. We involved four basic kernels the tests as listed below:

- RBF: Radial Basis Function kernel (also known as squared exponential kernel)
- RQ: Rational Quadratic Kernel (equivalent to adding together many RBF kernels with different lengthscales)
- Mat12: Matern 1/2 kernel (equivalent to the Exponential Kernel)
- Mat32: Matern 3/2 kernel

We applied each basic kernel and numbers of combinations (RBF + Mat21, RQ + Mat21, Mat32 + Mat21, RBF + Mat32, RQ + Mat32) to train the 2D GP models. Among these kernels, the combined one RBF+Mat21 give the best fit to the training data, however, it does not give the best predictions for the validating and testing data (off-grid). On the other hand, the GP model with the Mat32 kernel gives good (but not the best) fit for the training data, but the best accuracies for validating and the testing data. Comparing between the two kernels, the RBF+Mat12 Kernel is a combination of a smooth and a spiky kernel, which offers enough flexibility fit most features in the training data. However, the spiky function well fitting the sharp features will be applied in the nearby off-grid regions and this causes poor predictions. The smoothness of Mat32 is somewhere between a spiky (Mat21) and a smooth kernel (RBF). For the sharp changes, it gives a relatively smooth function compared with the Mat21 kernel and hence perform a good balance between on- and off-grid data. In the final models, we hence adopted Mat32 kernels for all trainings.

3.5 Strategy for Large sample

The model grid we aim to train contents about 10,000,000 data points, which is much more than the upper limit of data size of the Exact GP (20,000). We hence consider other GP approaches. Our tests for the large-sample strategy are carried out with a 3-demission (3D) GP model, which have three fundamental inputs: M and EEP and [Fe/H]$_{\rm init}$. The 3D model can be described as ($p = f(M, EEP, [\mathrm{Fe/H}]_{\rm init})$). Data for training 3D models (3D data hereafter) are from the primary grid with fixed $Y_{\rm init}$ (0.28), and $\alpha_{\rm MLT}$ (2.1). The training sample include 562 tracks and $\sim$ 300,000 data points. For validating and testing purposes, we computed 174 evolutionary tracks with the same input $Y_{\rm init}$, and $\alpha_{\rm MLT}$ but random M and [Fe/H]$_{\rm init}$ within grid ranges.

We first consider the Stochastic Variational GP (SVGP) with GPyTorch ApproximateGP module and test with the example on `https://docs.gpytorch.ai/en/v1.1.1/examples/04_Variational_and_Approximate_GPs/SVGP_Regression_CUDA.html`. SVGP is an approximate scheme rely on the use of a series of inducing points which can be selected in the parameter space. It trains using minibatches on the training dataset and build up kernels on the inducing numbers. Underline principles and detailed descriptions of this approach can be found in Hensman et al. (2014). The advantage of SVGP is the large capacity of sample size, however, the kernel complexities is still limited by the amount of inducing numbers. If SVGP uses inducing numbers as many as the training data of the Excat GP, better prediction accuracy will be expected because more information is given. Unfortunately, the the GPU memory is limited. When more training data is loaded, the inducing number for SVGP can not go up to 20,000 in our training. On the same GPU, a SVGP model is able to use up to10,000 inducing number when 100,000 training data is loaded. This is to say, although a SVGP model involves more data but sacrifices the kernel complicity compared with an Exact GP model. Our comparison between a SVGP model (with 10,000 inducing number with 100,000 training data) and an Exact GP model (with 20,000 training data) shows that the SVGP framework does not improve the GP model performance. For instance, the testing errors of $T_{\rm eff}$ at 68%, 95%, and 99.7% are 2.0, 5.8, and 15.7 K (error index = 23.5K) for Exact GP model and 2.2, 6.8, and 15.1 ($errorindex$ = 24.1) for the SVGP model.

We then investigate another approach designed for the large dataset named Structured Kernel Interpolation (SKI GP). SKI GP was introduced by Wilson & Nickisch (2015). It produces kernel approximations for fast computations through kernel interpolation and is a great way to scale a GP up to very large datasets (100,000+ data points). We follow the example on `https://docs.gpytorch.ai/en/stable/examples/02_Scalable_Exact_GPs/KISSGP_Regression.html` to develop our script. We run a few tests to train a 3D SKI GP model with 100, 000 training data. Compare with the Exact GP and SVGP, its testing errors of $T_{\rm eff}$ are slightly improved to 2.0, 6.1, and 14.8 K (error index = 22.9 K). However, the further test on the 5-demission data is not ideal: a SKI GP model with 100,000 training data performs much worse than an Exact GP model with 20,000 training data. The poor behaviour of the 5D model consist with what has been discussed in Wilson & Nickisch (2015). The method won't scale to data with high dimensions, since the cost of creating the grid grows exponentially in the amount of data. We attempt to make some additional approximations with the GpyTorch AdditiveStructureKernel module. It makes the base kernel to act as one-dimension kernels on each data dimension. and the final kernel matrix will be a sum of these 1D kernel matrices. Although the testing errors are improved but still worse than the simple Exact model. Thus, the SKI GP framework is also not ideal for our goal.

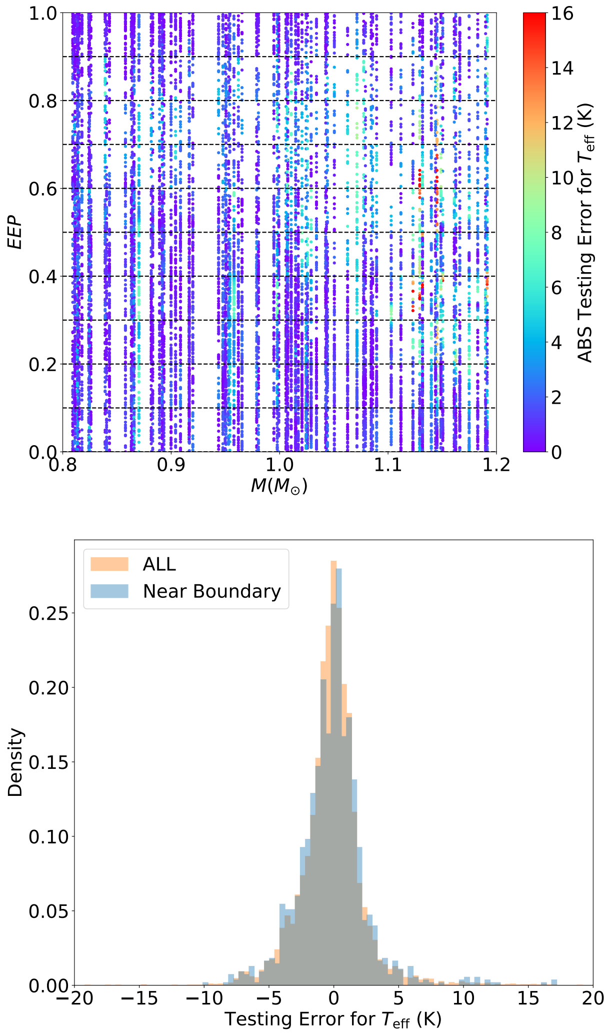

Figure 3. Top: Testing errors of 3D GP model for $T_{\rm eff}$ on the $M - -EEP$ diagram. Dashes indicates section boundaries. Bottom: examination of the edge effects of the section scenario. Probability distributions of testing errors of all testing data and those near the boundary (±0.01 EEP) in the upper graph are compared. As it can be seen, testing errors do not raise around the boundary.

As mentioned above, the GPU memory captivity limits the actual number of data that induce the kernel and hence the accuracy level. This limit become critical for high-demission cases, because the parameter space exponentially increases with model demission and hence GP model accuracy inevitable declines. Thus, a simple way of improving GP models is using more data to induce kernels. For this reason, we break the grid into a number of sections and train GP models for each section separately. With the same 20,000 training data for a single GP model, 10 sections means 10 times of the data are used to describe the grid. The downside of this section scenario is obvious: there will not be one GP model that maps the whole grid. However, as long as the goal of this work is augmenting a model grid but not deriving a universal function for stellar evolutions, this scenario is suitable.

Here we test how this approach works for the 3D data. We divide the dataset into 10 equal segments by EEP. We train one Exact GP model with 20,000 training data for each output parameter in each section. We then use the same testing dataset as used above to quantify predictions for the five outputs, and the error index are averagely improved by around 10%. For instance, the error index of $T_{\rm eff}$ decreases from 23.5 to 21.6 (1.7/5.0/14.9 K at 68/95/99.7%).

The section scenario method gives a better accuracy as expected, but there is a major concern about the edge effects at the boundary between sections. If the model works significantly poor at the boundary between sections, it will be difficult to map the systematic errors across the whole parameter space. We examine this in Figure 3. We show absolute testing errors for $T_{\rm eff}$ on the $M--EEP$ diagram (the top panel) and no obvious edge effects can be seen. We also do a statistical comparison between all data and those around the boundary ($\pm 0.01EEP$) as demonstrated at the bottom. The density distributions of the two datasets are very similar. We hence conclude that there is no obvious edge effect. The section scenario is adopted in the following study. Our set up for the GP model is summarised as below:

- Model Type: Exact GP with the section scenario
- Kernel: Mat32 (for all outputs)
- Mean Function: Neural Network with 6 linear layers x 128 nodes and element-wise function (Elu)
- Likelihood Function: Gaussian Likelihood Function
- Loss Function: Exact marginal log likelihood
- Optimiser: Adam including AMSGRAD variant
- Early Stoping: determined by the validating error index

4 AUGMENTING THE MESA GRID

With the GP model described in Section 3, we train GP models for the MESA grid. The training data includes the primary and the additional girds as mentioned in Table 1. The role of the additional grid is increasing the grid resolution for the high-mass models with hooks. As discussed in the previous section, the kernel function in the 'hook' area are much more complex than other regions , and hence a GP model requires more information to map the feature in this region. The total training data includes ~ 15,000 evolutionary tracks and ~ 10,000,000 data points. For validating and testing the data, we computed another 4880 tracks with random model inputs within the grid ranges. These tracks are split 50-to-50 for validating (in the training progress) and testing (after the training progress) GP models. We used the same sampling method of training, validating, and testing data. For each EEP section, we use 20,000 training data points and 20,000 validating data in the training progress.The testing data are selected in the whole EEP range and we use 50,000 (out of ~1,000,000) data points. Note that we do not use models with $\tau \geq 20.0$ Gyr, [Fe/H]$_{\rm surf}$ $\leq$ -0.6 dex, or $T_{\rm eff} \geq 7000K$ as testing data.

4.1 Overall Results

We firstly train an Exact GP model for the whole grid with 20,000 model, corresponding to a sampling rate of ~0.2%. The testing indexes of outputs remarkably increase compared to those of 3D GP models. We listed the testing errors of this 5D GP model in Table 2. Results of 2D and 3D GP models are also given for comparisons. It shows clear decline in GP model accuracy with increasing input demission. We then train GP models with the section scenario. We gradually increase the number of sections and track changes in testing errors. We found that improvements in testing errors becomes not significantly when the section number goes above 10. As it shown in Table 2, the 68th and 95th testing errors are at same accuracy levels for sections more than 10. Although the 99.7th errors seem to be better with more sections, this is more like a random error given the change is a small fraction. It turns out that the most efficient sampling rate is around 1% (10 sections). Above this rate, GP models learn no additional information about the grid. (references about this point?) Because less sections means less time and memory consumed for predicting. When take the 10-sections GP model as the final solution. Following analysis are all done with this model.

We show the testing errors in Figure 4. Because of the tail feature as illustrated in Figure 2, 95% confidential intervals vary in larger dynamic ranges than 68% confidential intervals. Testing errors of $T_{\rm eff}$, $\log g$, and R mainly depend on M and EEP. Metallicity error strongly depends on M, EEP, and [Fe/H]$_{\rm init}$, and age error has a significant correlation to EEP and [Fe/H]$_{\rm init}$. However, testing errors do not obviously relate to $Y_{\rm init}$ or $\alpha_{\rm MLT}$.

4.2 Mapping Systematical Uncertainties across the input parameter space

- Accuracy goes down. Because the multiple-demission space size increase while the training dataset has a limitation of 10,000 points.
- We hence need to divide the grid into small chunks and train each chunk separately. (illustrate how chunks are divided)
- show training and validation results in a fancy way.

5 DISCUSSION

Do some disccusion.

6 CONCLUSIONS

Restate the main results of the paper.

ACKNOWLEDGEMENTS

Development of GPyTorch is supported by funding from the Bill and Melinda Gates Foundation, the National Science Foundation, and SAP.

Table 2. Training and validating errors for GPR Models

Model Type	Inputs	N_{Training}	Sampling rate	Testing Errors (at 68/95/99.7%)				
				T_{eff} (K)	$\log g$ (10^{-3}dex)	R ($10^{-3}R_{\odot}$)	[Fe/H]$_{\mathrm{surf}}$ (10^{-3}dex)	τ (10^{-2}Gyr)
Exact GP	2D	20,000 x 1	96%	1/5/11	1/3/8	2/6/14	0.5/2/12	1/3/9
Exact GP	3D	20,000 x 1	5%	2/6/16	1/4/10	3/7/17	2/6/22	2/7/22
Exact GP (10 sections)	3D	20,000 x 10	50%	2/5/15	1/4/11	2/7/17	1/3/20	2/6/19
Exact GP	5D	20,000 x 1	0.2%	3/9/34	2/5/18	4/11/36	2/7/30	3/9/27
Exact GP (3 sections)	5D	20,000 x 3	0.6%	3/8/27	2/5/18	3/7/26	1/4/24	3/7/22
Exact GP (5 sections)	5D	20,000 x 5	1%	2/7/25	1/4/15	3/7/24	1/4/21	2/6/22
Exact GP (10 sections)	5D	20,000 x 10	2%	2/7/27	1/4/14	2/7/26	1/4/20	2/6/21
Exact GP (20 sections)	5D	20,000 x 20	4%	2/7/26	1/4/14	2/7/27	1/3/18	2/6/22
Exact GP (100 sections)	5D	20,000 x 100	20%	2/7/25	1/4/14	2/7/26	1/3/17	2/6/18

REFERENCES

Asplund M., Grevesse N., Sauval A. J., Scott P., 2009, Annual Review of Astronomy and Astrophysics, 47, 481

Ferguson J. W., Alexander D. R., Allard F., Barman T., Bodnarik J. G., Hauschildt P. H., Heffner-Wong A., Tamanai A., 2005, ApJ, 623, 585

Gardner J. R., Pleiss G., Bindel D., Weinberger K. Q., Wilson A. G., 2018, in Advances in Neural Information Processing Systems.

Hensman J., Matthews A., Ghahramani Z., 2014, arXiv preprint arXiv:1411.2005

Paquette C., Pelletier C., Fontaine G., Michaud G., 1986, ApJS, 61, 177

Paxton B., Bildsten L., Dotter A., Herwig F., Lesaffre P., Timmes F., 2011, The Astrophysical Journal Supplement Series, 192, 3

Paxton B., et al., 2013, The Astrophysical Journal Supplement Series, 208, 4

Paxton B., et al., 2015, The Astrophysical Journal Supplement Series, 220, 15

Paxton B., et al., 2018, ApJS, 234, 34

Paxton B., et al., 2019, ApJS, 243, 10

Reddi S., Kale S., Kumar S., 2018, in International Conference on Learning Representations.

Rogers F. J., Nayfonov A., 2002, ApJ, 576, 1064

Sutskever I., Martens J., Dahl G., Hinton G., 2013, in International conference on machine learning. pp 1139–1147

Thoul A. A., Bahcall J. N., Loeb A., 1994, ApJ, 421, 828

Townsend R. H. D., Teitler S. A., 2013, MNRAS, 435, 3406

White T. R., Bedding T. R., Stello D., Christensen-Dalsgaard J., Huber D., Kjeldsen H., 2011, ApJ, 743, 161

Wilson A., Nickisch H., 2015, in International Conference on Machine Learning. pp 1775–1784

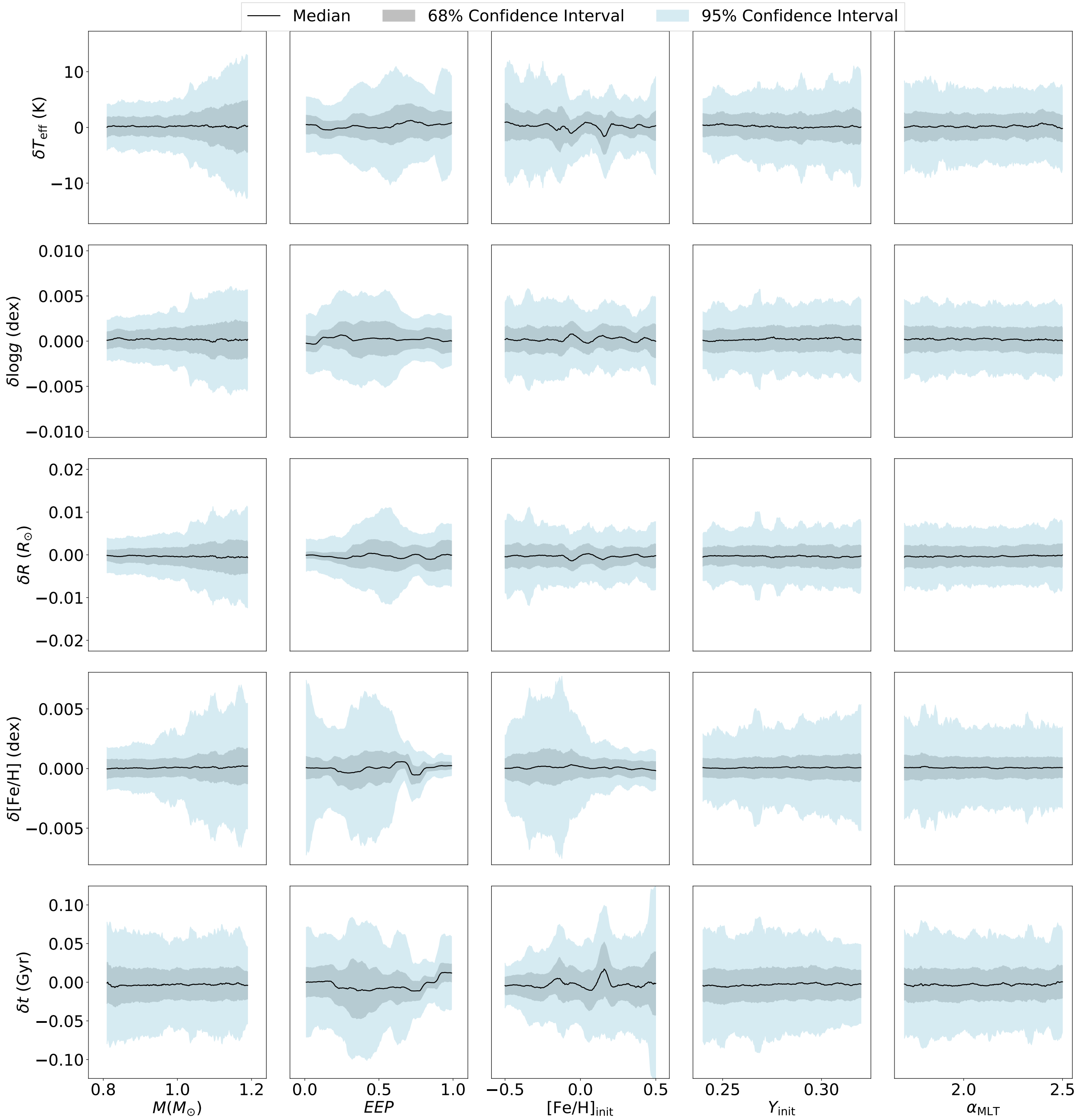

Figure 4. Roll medians and confidential interval of testing errors against GP model inputs. Black solid line indicate the median; grey and blue shadow represent the 68% and 95% confidential interval. Testing errors of $T_{\rm eff}$, log g, and R mainly depend on M and EEP. Metallicity error strongly depends on M, EEP, and [Fe/H]$_{\rm init}$, and age error has a significant correlation to EEP and [Fe/H]$_{\rm init}$. However, testing errors do not obviously relate to $Y_{\rm init}$ or $\alpha_{\rm MLT}$.

Figure A1. Validations for 5D inputs GPR models before and after chunking.

APPENDIX A: VALIDATION AND PREDICTION OF GPR MODELS

A1 GPR model with 3-D inputs

- Training set
- validation
- GP predictions

A2 GPR model with 5-D inputs

- Training set
- validation
- GP predictions

This paper has been typeset from a TEX/LATEX file prepared by the author.