Draft version May 17, 2024
Typeset using LaTeX **twocolumn** style in AASTeX631

Using Hybrid Machine Learning for Fundamental Stellar Parameter Inference

Sujay Shankar,[1] Michael Gully-Santiago,[1] and Caroline V. Morley[1]

[1] *Department of Astronomy, The University of Texas at Austin, Austin, TX 78712, USA*

ABSTRACT

The advent of machine learning methods has revolutionized numerous scientific methods, among which is spectroscopic inference. Studies can and have shown that machine learning methods prove effective in inferring what we call fundamental stellar parameters. These include $T_{\rm eff}$, $\log(g)$, and [Fe/H] for the purposes of this study but can also encompass other parameters such as [α/Fe], C/O, and f_{sed}. While current methods tend to treat spectra purely from a data perspective, we propose to use a hybrid machine learning (HML) method that uses the semi-empirical approach of treating spectra as a set of spectral lines put forth by `blase` (Gully-Santiago & Morley 2022), combined with traditional linear interpolation as well as Bayesian optimization. We interpolate the four line properties (μ', $\ln(a)$, σ, γ) of 128,723 unique spectral lines across a subset of the PHOENIX synthetic model grid (Husser et al. 2013) to create the PHOENIX generator, allowing us to reconstruct spectra at any continuously valued point lying within the domain of the PHOENIX subset, and even reconstruct batches of spectra in pseudo-parallel fashion, with the speedup exceeding 20 times once more than 60 spectra are reconstructed. We then combine the PHOENIX generator with Bayesian optimization to efficiently infer fundamental stellar parameters. This study is however a proof of concept, as we have not yet included various features that would be necessary for a full-fledged tool for spectroscopic inference, however we show that semi-empirical HML methods are a viable, albeit fledgling alternative to more traditional approaches.

1. INTRODUCTION

Stellar spectra are exceedingly rich sources of information about the stars which create them. They contain information about the fundamental properties of the star such as its temperature and surface gravity, as well as countless absorption lines reflecting the star's chemical composition. On top of that, extrinsic factors such as the radial velocity of the star or the rotation of the star itself further transform spectra. When these spectra are observed, even the location of the observing instrument and the capabilities of the instrument itself introduce further modifications. In short, stellar spectra represent extremely complex, high-dimensional data that has gone through multiple transformations before reaching our eyes as a deceptively simple graph of flux versus wavelength.

Scientists want to extract as much information as possible from a star's spectra to get back the properties that lie at the root of it all. However, this has proven to be no easy task. Spectroscopic surveys such as APOGEE develop their own in-house pipelines to extract stellar parameters from spectra (ASPCAP in the case of APOGEE), and these pipelines are fairly effective (Majewski et al. 2017; García Pérez et al. 2016). However, they all work on the core assumption that the data to be analyzed is a list of pixels. In addition, the pipelines that surveys develop are typically closed-source and usually limited to the scope of the survey itself, causing a sort of fragmentation. This motivates the development of a more universal, open-source tool that can do what spectral analysis pipelines do, but in an alternative way.

Multiple efforts have been made in this direction, treating spectra in different ways. The standard practice is to treat the wavelength and flux as simply two arrays and implement any from a myriad of different algorithms to inference fundamental stellar parameters. Others take a more mathematical approach by decomposing spectra into some eigenbasis and treating it as a regression problem, such as `starfish` (Czekala et al. 2015). However, `blase` takes a different approach. It treats spectra not as a set of pixels or a set of eigenbasis coefficients but as a set of spectral lines, specifically Voigt profiles. There is no metric of this approach being better or worse than the others mentioned so far, but it is a different paradigm and is worth exploring as a semi-empirical tool. This study aims to take advantage of `blase`'s ability to change a spectrum's basis from flux versus wavelength into spectral line versus spectral

line properties, and use that to develop a robust tool for stellar parameter inference.

However, this is only a proof of concept. Here we develop a tool that represents a very basic realization of the idea of line-by-line inference. We make no claims as to the performance of this tool, but rather aim to show that the idea is sound and able to be implemented, even if rudimentary, and that with further research and development, this tool has the potential to become useful for spectral inference.

We chose the PHOENIX synthetic spectral model grid as the basis for this study, because it is fairly well-known. While we plan to extend this tool to leverage information across multiple model grids in the future, both for accuracy and to increase the scope to include substellar objects with grids such as Sonora (Marley et al. 2021; Karalidi et al. 2021; Morley et al. 2024; Mukherjee et al. 2024), for now we limit our scope to a subset of the PHOENIX grid.

Our study is divided into three main parts. First, we preprocess the PHOENIX subset and detect spectral lines with `blase`. Second, we interpolate the spectral line parameters, enabling us to evaluate the PHOENIX grid at any point within the bounds of the subset. Lastly, we use Bayesian optimization to power our inference algorithm and test its performance. An overview of this process is presented in Figure 1.

2. CLONING THE PHOENIX MODEL GRID

2.1. *The PHOENIX Subset*

For the purposes of this study, we did not consider the full range of the PHOENIX synthetic spectral model grid. Instead, we focused on a subset of the grid, whose parameter ranges are given in Table 1.

Parameter	**Symbol**	**Range**
Alpha Element Abundance	α	[0] dex
Iron Abundance	[Fe/H]	[-0.5, 0] dex
Effective Temperature	$T_{\rm eff}$	[2300, 12000] K
Surface Gravity	$\log(g)$	[2, 6]
Wavelength	λ	[8038, 12849] Å

Table 1. The subset of the PHOENIX grid used in this study. These limits were imposed to reduce the computational cost of the algorithms and to ensure a rectilinear parameter space in order to work with `scipy`'s `RegularGridInterpolator` (Virtanen et al. 2020). The wavelength limits in particular roughly line up with that of the Habitable Zone Planet Finder (HPF) spectrograph (Mahadevan et al. 2012). This subset is comprised of 1314 individual spectra.

2.2. *Preprocessing with* `gollum`

First, the PHOENIX subset was accessed directly from the PHOENIX website using `gollum`'s download option (Shankar et al. 2024). The spectra were then put through a three-step preprocessing pipeline.

1. *Blackbody Division*: Since the $T_{\rm eff}$ of each spectrum is known, the according blackbody spectrum was divided out.
2. *Percentile Normalization*: The spectra were normalized by dividing them by their 99th percentile.
3. *Continuum Normalization*: The spectra were further normalized by dividing them by a $5^{\rm th}$ order polynomial continuum.

Mathematically, we can express the preprocessing as follows:

$$\bar{\mathsf{S}} = \frac{\mathsf{S}}{\mathsf{B}\mathsf{Q}_5 P_{99}} \tag{1}$$

where $\bar{\mathsf{S}}$ is the preprocessed spectrum, S is the original spectrum, B is the blackbody spectrum, Q_n is the $n^{\rm th}$ order polynomial continuum fit, and P_n is the $n^{\rm th}$ percentile function. Arithmetic operations between arrays are assumed to be elementwise in all notation from here on out.

2.3. *Line Identification with* `blase`

The next step was to convert the PHOENIX subset into an interpretable format. We wanted to represent the spectra as a list of spectral lines rather than a list of fluxes. This was done using `blase`, which detects spectral lines as Voigt profiles and tunes the profiles to mimic the original PHOENIX spectrum with back propagation. Four parameters were optimized: the line center μ, the log-amplitude $\ln(a)$, the Gaussian width σ, and the Lorentzian width γ. The optimization used the Adam optimizer with a learning rate of 0.05 over 100 epochs (Kingma & Ba 2017). In addition, we limited two custom parameters: wing cut to 6000 and prominence to 0.005. Wing cut is a parameter that determines the extent of the Voigt profile to evaluate, saving computational resources by not evaluating small numbers. Prominence sets a lower limit for the amplitude of detected lines, which also saves resources by disregarding shallow lines. In short, larger values for wing cut and smaller values for prominence both increase the accuracy of using `blase`'s line detection and optimization at the expense of an increased computational cost. In addition, it should be mentioned that `blase` uses the pseudo-Voigt approximation, which saves on computational cost while remaining accurate to about 1% (Ida

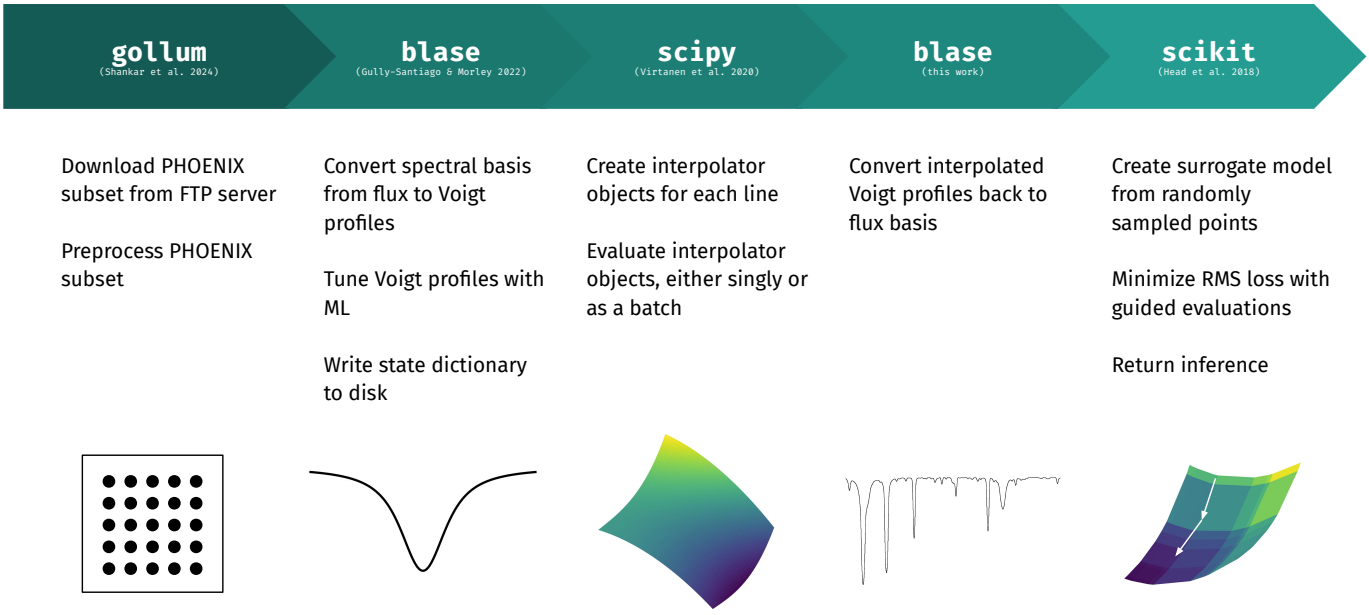

Figure 1. Overview of the process used in this study.

et al. 2000; Thompson et al. 1987). It is a weighted average of a Gaussian and Lorentzian as opposed to a convolution. `blase`'s pseudo-Voight profile implementation uses the following:

$$\tilde{\mathrm{V}}_\mu(\lambda) = a\left[\eta \mathbf{L}(\lambda - \mu'; f) + (1-\eta)\mathbf{G}(\lambda - \mu'; f)\right] \quad (2a)$$

$$\eta = \sum_{n=1}^{3} \mathbf{u}_n \left(\frac{2\gamma}{f}\right)^n \quad (2b)$$

$$f = 32 \sum_{n=0}^{5} \mathbf{v}_n \left(\sqrt{2\ln(2)}\sigma\right)^{5-n} (\gamma)^n \quad (2c)$$

$$\mathbf{u} = \begin{bmatrix} 1.36603 \\ -0.47719 \\ 0.11116 \end{bmatrix} \quad \mathbf{v} = \begin{bmatrix} 1 \\ 2.69269 \\ 2.42843 \\ 4.47163 \\ 0.07842 \\ 1 \end{bmatrix}$$

where $\mathbf{L}$ and $\mathbf{G}$ are abbreviations for Lorentzian and Gaussian profiles, respectively. Notice that we use μ' instead of μ in the formula. This is because `blase` allows the line center to shift slightly during optimization, and it is this shifted center which is used in computation. The individual Voigt profiles are still indexed by μ for cross-model line identification, explained in the next section. Once optimization was complete, the list of identified lines, present in the state dictionary of `blase`'s model, was saved to a `.pt` file for each PHOENIX subset grid point (Paszke et al. 2019). The total disk space these files took up is 465 MB.

3. INTERPOLATING MANIFOLDS

3.1. *Cross-Model Line Identification*

Earlier we mentioned that `blase` tuned the line centers of detected lines. This means that from one PHOENIX spectrum to the next, the same line could have a slightly different line center. Since the goal of this study is to interpolate the properties of each line, we needed to identify the presence of a particular line across the PHOENIX subset. We decided to do this by using the line centers μ of the detected lines pre-optimization. Now with each spectral line indexed by μ, we had four parameters to interpolate: μ', $\ln(a)$, σ, and γ. Note that since we are dealing with the parameters of Voigt profiles, we can see in Equation 2 that even if the interpolation method is linear, the actual result of interpolation varies non-linearly.

There is also a second issue, and that is that spectral lines were often only detected in some spectra from the PHOENIX subset. In Figure 2, we show that different grid points sport differing counts of detected spectral lines. This means that in theory, interpolation would be inaccurate in regions where a line does not

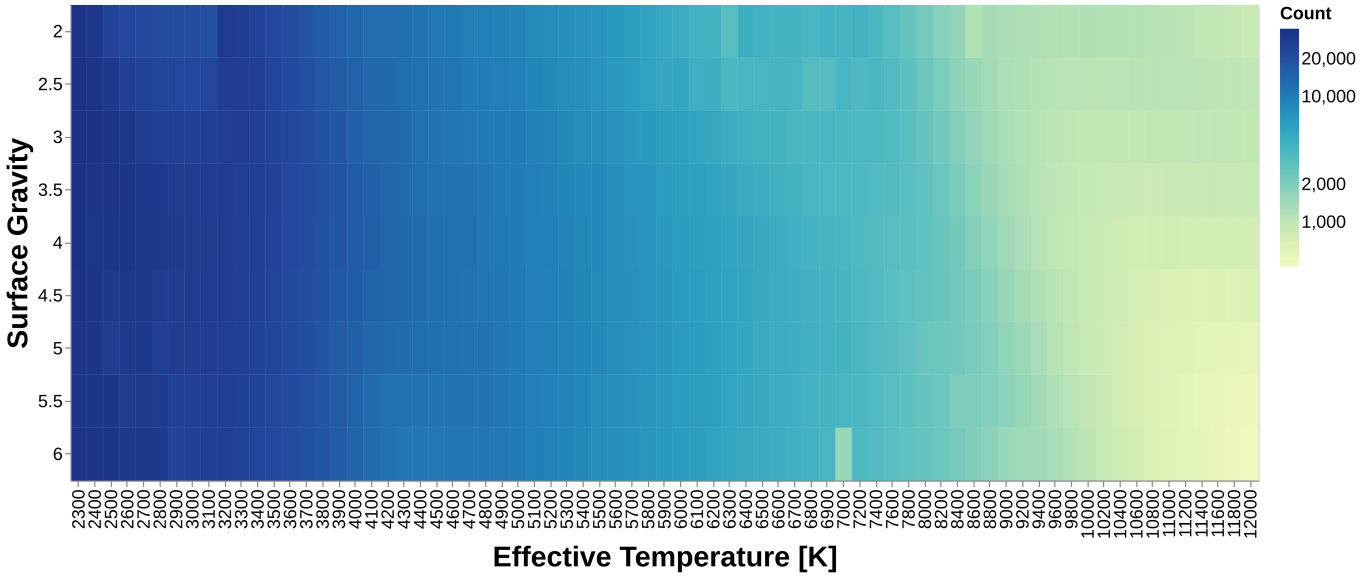

Figure 2. Number of detected spectral lines at each grid point of a slice of the PHOENIX subset at solar metallicity. We can see that the number of detected lines decreases with increasing $T_{\rm eff}$. Also note that from $T_{\rm eff}$ = 7000 K onward, the spacing between grid points increases from 100 K to 200 K.

appear, and also breaks in practice because `scipy`'s `RegularGridInterpolator` relies on a rectilinear grid, which we would not have if regions of grid points were missing.

To solve this, we artificially populated missing grid points with log-amplitudes of -1000, which retained interpolator stability by not being an infinity, but also essentially nullified the line when evaluated. An example of the appearance of missing sections in heatmaps where a line does not appear is shown in Figure 3. In total, across the entire PHOENIX subset, `blase` detected 128,723 individual spectral lines, meaning we have that many manifolds to interpolate.

3.2. *Continuously Evaluable Manifolds*

For each line, the inputs to the interpolator were the three input parameters $T_{\rm eff}$, $\log(g)$, and [Fe/H], and the output was a list of four parameters, μ', $\ln(a)$, σ, and γ. For each line, one of these interpolator objects was created using linear interpolation, and these interpolators were aggregated into a single list, which was when pickled and written to a `.pkl` file. These interpolators generate multiple manifolds with the following mapping:

$$\begin{bmatrix} T_{\rm eff} \\ \log(g) \\ [\rm Fe/H] \end{bmatrix} \rightarrow \begin{bmatrix} \mu' \\ \ln(a) \\ \sigma \\ \gamma \end{bmatrix} \tag{3}$$

These interpolators could now be evaluated at any point lying within the domain of the PHOENIX subset, turning a discretely sampled PHOENIX subset into a continuous one. With the given size of the PHOENIX subset, the interpolator list takes up 13.2 GB of disk space. This evaluation is able to reconstruct an existing PHOENIX spectrum or alternatively interpolate a new spectrum within the domain of the PHOENIX subset, we call this the PHOENIX generator. In Figure 4, we show the same spectral line as in Figure 3, but now supersampled using the PHOENIX generator evaluated over the same slice.

The spectral reconstruction process is done by iterating over the PHOENIX generator, evaluating each interpolator at the given coordinates, then reshaping the data into the same format that PyTorch uses for state dictionaries. During the iteration, if the interpolated log-amplitude of the line is less than -100, the line is excluded from the state dictionary. Why -100 instead of -1000? This is because near the areas that we artificially populated, the manifold experiences artifacts due the sudden decline. We have to anticipate that decline and cut off the behavior in advance. This saves on computational time. Then, the state dictionary is fed into `blase`'s `SparseLinearEmulator`, which reconstructs the spectrum by constructing a forward model based on the input state dictionary. Any `nan` values are set to 1 (which we can do because the spectra are all normalized), and the spectrum is complete.

4. BAYESIAN INFERENCE AND TESTING

4.1. *Spectral Reconstruction Time*

The typical use case for the PHOENIX generator may be to batch reconstruct spectra from an array of input coordinates. Therefore, there is some motivation to at-

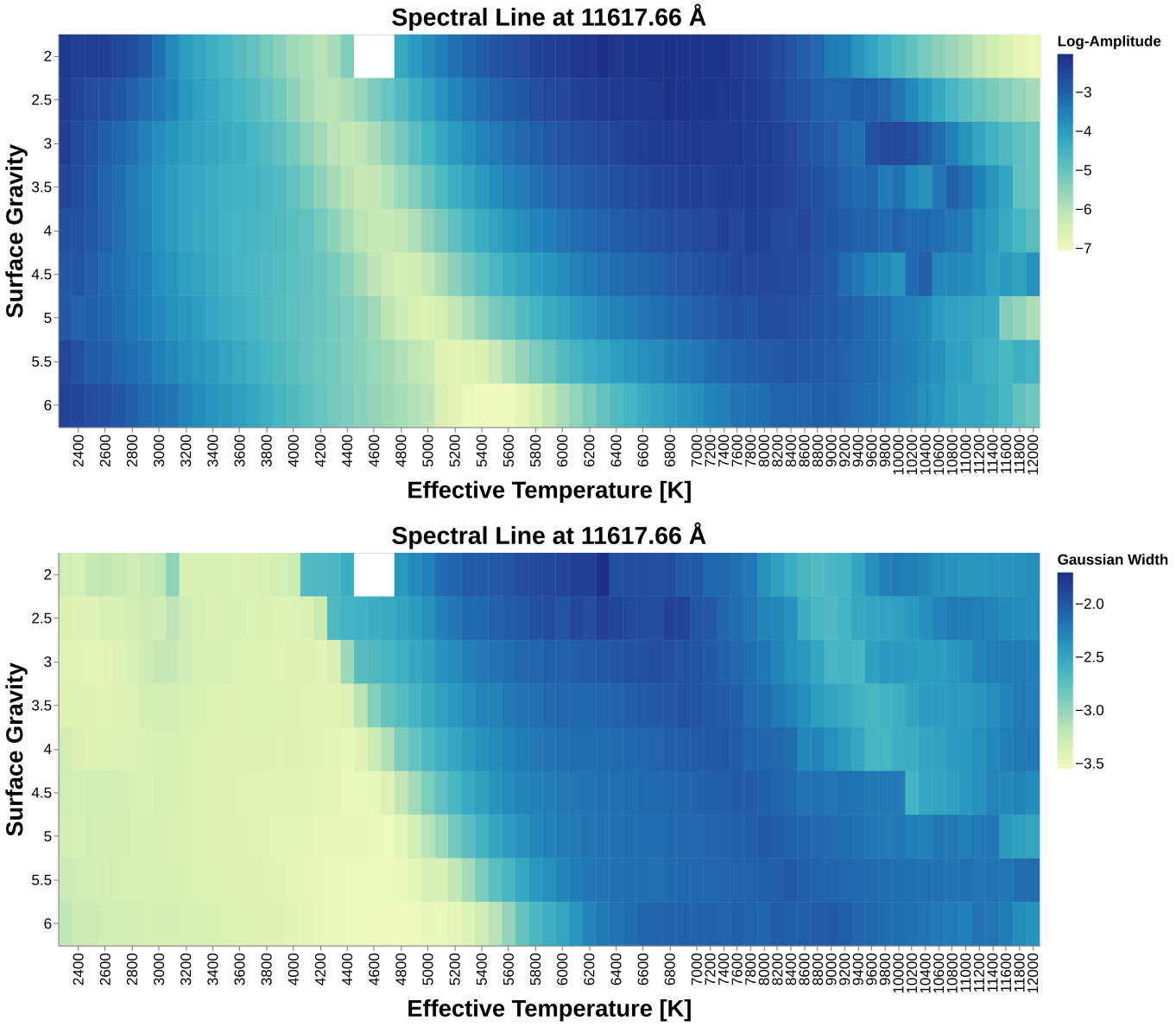

Figure 3. Heatmap showing how $\ln(a)$ and σ vary over the PHOENIX subset slice at solar metallicity. Notice the missing chunk in the top left of the figure; `blase` did not detect a spectral line here, but we have to artificially populate those points with lines that have $\ln(a) = -1000$. The change in spacing on the x-axis is owing to the step of $T_{\rm eff}$ in the PHOENIX grid changing from 100 K to 200 K after $T_{\rm eff} = 7000$ K.

tempt to reduce the computational cost of this procedure to tractable levels. We evaluate the computational time needed to use the PHOENIX generator in two distinct ways. First, for a single input, which would be relevant in serial applications. Second, for an array of multiple inputs, which would be relevant in parallel applications. The `RegularGridInterpolator` API allows for the passing in of an entire array of input coordinates to be evaluated at once, however using `blase` to reconstruct the spectrum from our interpolated state dictionary is done serially, leading to what is actually more of a psuedo-parallel evaluation. Performance results are shown in Figure 5, and we can see that the pseudo-parallel implementation is much faster.

4.2. *Inference Algorithm*

We elected to use Bayesian optimization as the inference algorithm, specifically the `gp_minimize` function from the `scikit-optimize` library (Head et al. 2018). This algorithm uses a Gaussian Process to model the objective function, which in this case was the RMS (Root-Mean-Square) loss between the interpolated spectrum

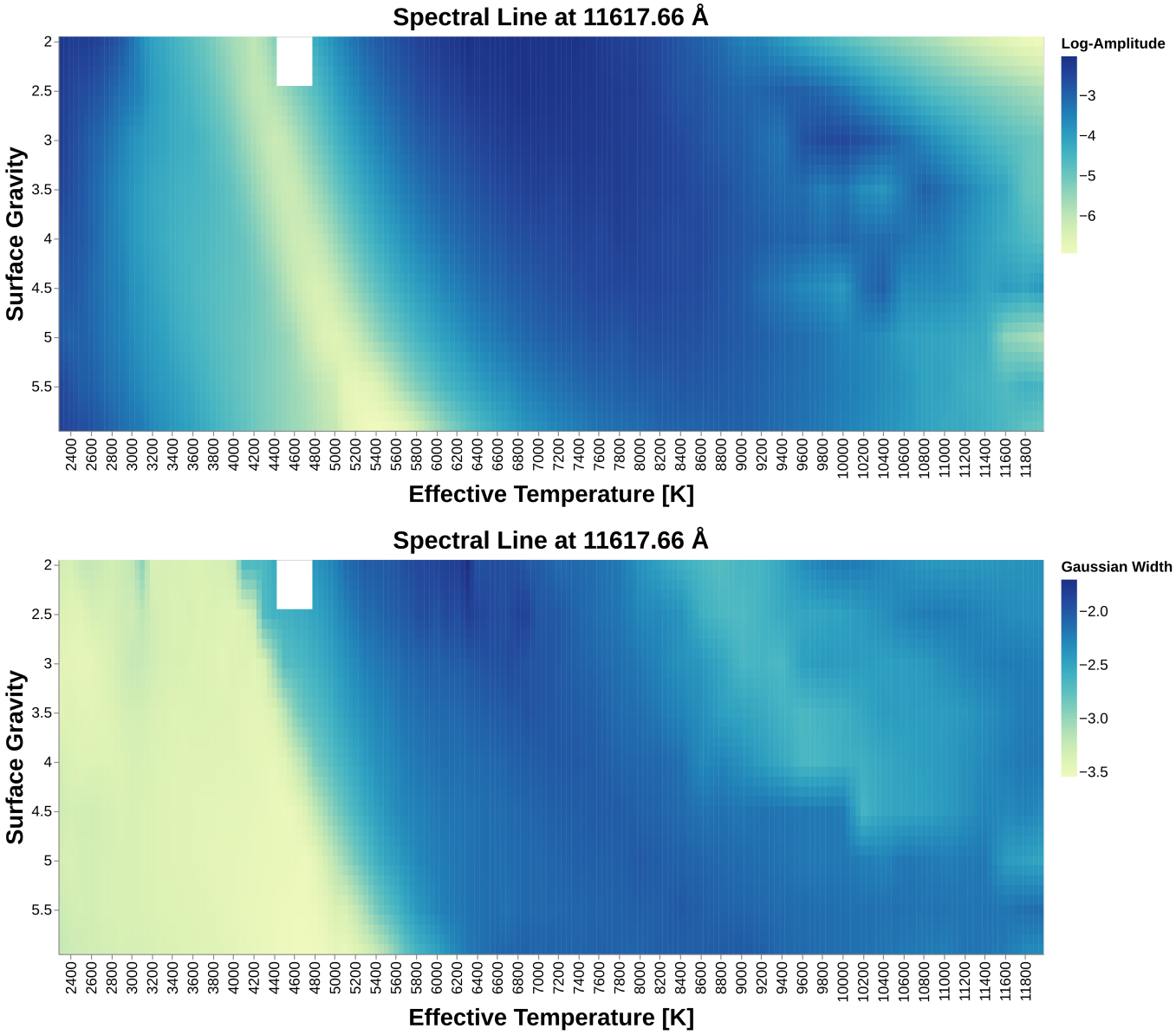

Figure 4. Heatmap showing how $\ln(a)$ and σ vary over the PHOENIX subset slice at solar metallicity, now supersampled with the PHOENIX generator. Notice that the missing chunk in the top left still exists and does not display any artifacts, as the artificially populated points are removed after interpolation to retain the model's integrity. Also see that the x-axis spacing is now uniform, as the PHOENIX generator was evaluated at constant step.

M and the true spectrum D, defined as:

$$\mathcal{L} = \left\langle (\mathsf{M} - \mathsf{D})^{\circ 2} \right\rangle^{1/2} \tag{4}$$

The optimizer was configured to first run 100 random evaluations to seed the surrogate model, then run 20 more evaluations now guided by the surrogate model. This totals to 120 evaluations, which was deemed sufficient for this study. One inference run takes us on average, just under 7.5 minutes to complete.

4.3. *Bayesian Optimizer Performance*

To test the performance of the inference algorithm, we used the PHOENIX subset itself. At first glance, this may seem like a useless endeavor, as the PHOENIX generator has indeed memorized the PHOENIX subset, being able to reconstruct a PHOENIX spectrum when evaluated at that grid point. However, that is precisely what allows us to use the PHOENIX subset as test data. The Bayesian optimizer's surrogate model is seeded by random continuously sampled generator evaluations within the search space, *not* grid points of the PHOENIX subset, meaning the surrogate model has no 'memorization' to speak of. If the optimizer had been

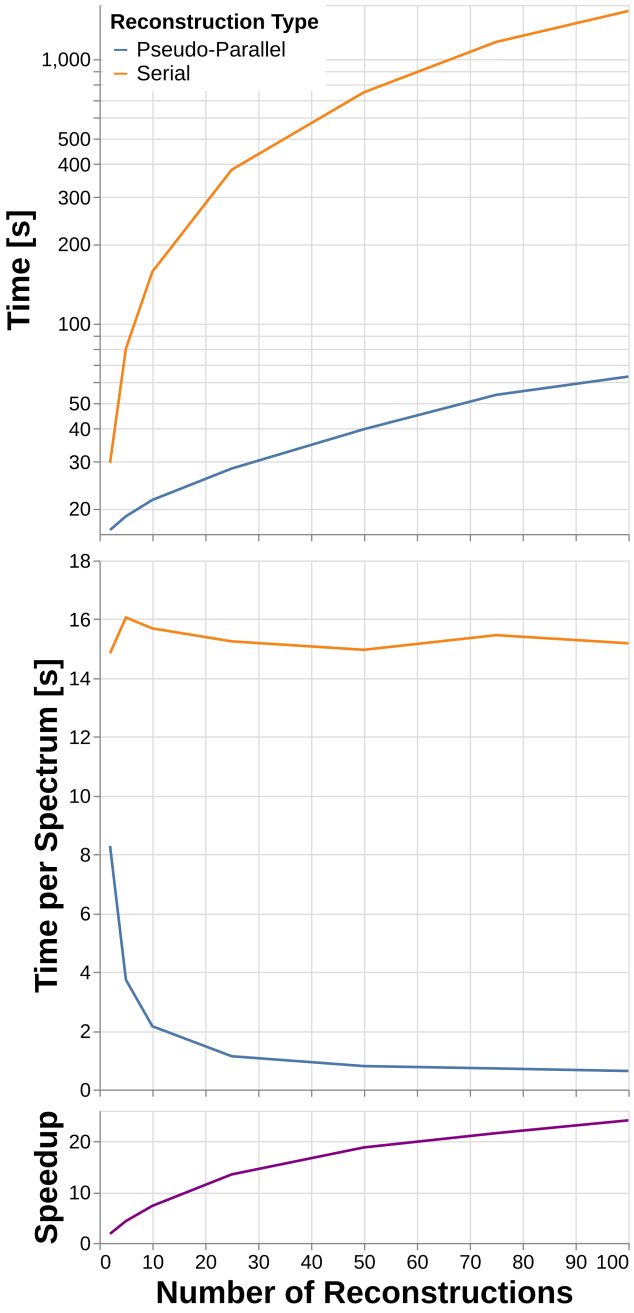

Figure 5. Line chart showing the time taken to reconstruct varying numbers of spectra using the PHOENIX generator (lower is better). We can see that the time taken per spectra for the serial implementation hovers around 15 seconds within run-to-run variance, while the pseudo-parallel implementation continually decreases in time taken per spectrum as the number of inputs increases. The speedup factor (higher is better) increases as more spectra are generated, which is also a desirable outcome.

an iterative grid refinement search, this would not have been possible, because grid methods rely on just that, the grid.

We know that $T_{\rm eff}$ tends to be fairly well-constrained prior to even looking at the spectra, so when testing the inference algorithm, we limited its search space to only include $T_{\rm eff}$ which lay within 500 K of the true value on either side. $\log(g)$ and [Fe/H] were allowed to vary freely. The test sample consisted of 9 unique $T_{\rm eff}$ values, 3 unique $\log(g)$ values, and 2 unique [Fe/H] values, totaling 54 unique spectra. $T_{\rm eff}$ ranged from 3000 K to 11000 K in increments of 1000 K, $\log(g)$ ranged from 2 to 6 in increments of 2, and [Fe/H] ranged from -0.5 to 0 in increments of 0.5.

The results of the inference algorithm are as follows: $T_{\rm eff}$ was off from the true result by an average of 185 K or 2.6%. $\log(g)$ was off by an average of 0.19 or 6.8%. [Fe/H] was off by an average of 0.12 dex (no percentage, because you can't divide by zero).

5. FINAL CONSIDERATIONS

The computer used for this study, Triton, has the following specifications:

CPU	AMD EPYC 7513
RAM	256 GB
GPU	Nvidia A100 40GB (×2)

Table 2. Triton is owned by Caroline Morley and is available to members of her research group. It was used for all computations, but not for generating visualizations. The EPYC 7513 is a 32c/64t CPU with a boost clock of 3.65 GHz. The A100 has 6912 CUDA cores.

It should be noted that this study represents a proof of concept, and that there are numerous design considerations that could be improved upon with future work. These considerations include but are not limited to the following:

- *Limited PHOENIX Subset*: The PHOENIX subset used in this study was just that, a subset. The full PHOENIX grid not only expands the [Fe/H] range to [-4.0, 1.0] dex and the $\log(g)$ range to [0, 6], but also includes the alpha element abundance parameter, which we elected to fix at 0 for this study. In addition to the actual fundamental stellar parameters, we also took a subset of the PHOENIX wavelength range, with the full [500, 55000] Å wavelength range also being left to future work.

- *Strict Wavelength Range*: Currently, the generator only supports inference on spectra whose wave-

length limits are either equal to it, or encompass that of the generator and have been truncated to match. However, when the spectrum in question has a smaller wavelength range than the generator, currently there is no functionality to truncate the generator. This would require externally indexing the generator's individual interpolators by line center position and selectively evaluating those to eliminate wasteful computation.

- *Single Model Grid*: The PHOENIX grid is not the only model grid of synthetic spectra available, and it does not apply to all types of stars. Future work would extend the reach of this study's algorithm to encompass other model grids such as the Sonora series for substellar models. `blase` should be able to have an option for the user to input which model grid they would like to base the inference on, and to get even more advanced, perhaps even have the ability to intelligently determine which model grid to use automatically.

- *Memorization vs. Generalization*: The current design of the algorithm constructs manifolds using interpolation. This means that performance is good at points close to PHOENIX subset grid points, but is highly dependent on the type of interpolation used. As interpolators require memorization of the data, advanced interpolation becomes extremely expensive in terms of disk utilization. Future work would involve constructing manifolds using regression, which would allow for much better generalization and lower disk utilization at the expense of some accuracy.

- *Extrinsic Absence*: The current design of our algorithm does not account for extrinsic parameters that modify the appearance of spectra such as rotational broadening and doppler shifting. Future work would need to develop ways to tune these extrinsic parameters alongside the fundamental stellar parameters.

- *Framework Overhead*: As this algorithm is currently more proof of concept than practical, it uses convenience functions from various libraries, which naturally introduces some level of overhead and leaves performance on the table. Future work would involve writing custom functions expressly designed for `blase`, most likely a complete rewrite of the library from the ground up.

- *Pseudo-Interpretability*: Our algorithm boasts interpretability by considering spectral lines as the objects of interest as opposed to the rather uninterpretable flux values of other approaches. However, this is only a step in the direction of interpretability. True interpretability would decompose a spectrum not into a set of spectral lines, but into a set of species component spectra, which requires a much more advanced understanding of different species and their behavior, as well as direct access to a radiative transfer code as opposed to an off-the-shelf model grid. This approach would also extend the inference from just fundamental stellar parameters defined by a grid to any set of parameters accounted for in the radiative transfer model, down to specific species abundances. So while readers may be tempted to try to identify certain spectral lines such as the one we use for our figures, this is not the point. `blase` is *agnostic* to the identity of the line that it is optimizing. We study these lines as `blase` sees them (i.e. their four shape parameters), because for the purposes of this study, that is the only information that is useful.

- *The Continuum Black Box*: Continuum normalization is a process that is not yet completely understood, and is currently done as a preprocessing step with a fairly simple algorithm. Future work would dive deeper into the science of continuums and develop more advanced methods that can discern continuums with greater accuracy and less modeling restrictions.

- *One Voigt Fits All*: The current assumption of `blase` is that every spectral line is a Voigt profile. This assumption is largely true, but there are situations where that is simply not enough. Future studies need to account for more advanced spectral line profiles and procedures to deal with phenomena such as ro-vibrational bands.

In short, it is quite clear that `blase` is a long way from being a powerhouse in spectral inference, however it represents a step down a road not yet traveled, and the potential for future growth is immense. At the end of the day, we aim to develop a tool that can analyze spectra with the growing power of physics-informed machine learning.

1 Text

Software: `altair` (VanderPlas et al. 2018; Satyanarayan et al. 2017), `astropy` (Astropy Collaboration et al. 2013, 2018, 2022), blasé/`blase` (Gully-Santiago & Morley 2022), CUDA (NVIDIA et al. 2020), `gollum`

(Shankar et al. 2024), `matplotlib` (Hunter 2007), `numpy` (Harris et al. 2020), `pandas` (pandas development team 2020; Wes McKinney 2010), Python (Van Rossum & Drake 2009), PyTorch/`torch` (Paszke et al. 2019), scikit-optimize/`skopt` (Head et al. 2018), `scipy` (Virtanen et al. 2020), `tqdm` (da Costa-Luis 2019), `vegafusion` (Kruchten et al. 2022),

REFERENCES

Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., et al. 2013, A&A, 558, A33, doi: 10.1051/0004-6361/201322068

Astropy Collaboration, Price-Whelan, A. M., Sipőcz, B. M., et al. 2018, AJ, 156, 123, doi: 10.3847/1538-3881/aabc4f

Astropy Collaboration, Price-Whelan, A. M., Lim, P. L., et al. 2022, ApJ, 935, 167, doi: 10.3847/1538-4357/ac7c74

Czekala, I., Andrews, S. M., Mandel, K. S., Hogg, D. W., & Green, G. M. 2015, ApJ, 812, 128, doi: 10.1088/0004-637X/812/2/128

da Costa-Luis, C. O. 2019, Journal of Open Source Software, 4, 1277, doi: 10.21105/joss.01277

García Pérez, A. E., Allende Prieto, C., Holtzman, J. A., et al. 2016, AJ, 151, 144, doi: 10.3847/0004-6256/151/6/144

Gully-Santiago, M., & Morley, C. V. 2022, ApJ, 941, 200, doi: 10.3847/1538-4357/aca0a2

Harris, C. R., Millman, K. J., van der Walt, S. J., et al. 2020, Nature, 585, 357, doi: 10.1038/s41586-020-2649-2

Head, T., MechCoder, Louppe, G., et al. 2018, scikit-optimize/scikit-optimize: v0.5.2, v0.5.2, Zenodo, doi: 10.5281/zenodo.1207017

Hunter, J. D. 2007, Computing in Science & Engineering, 9, 90, doi: 10.1109/MCSE.2007.55

Husser, T. O., Wende-von Berg, S., Dreizler, S., et al. 2013, A&A, 553, A6, doi: 10.1051/0004-6361/201219058

Ida, T., Ando, M., & Toraya, H. 2000, Journal of Applied Crystallography, 33, 1311, doi: https://doi.org/10.1107/S0021889800010219

Karalidi, T., Marley, M., Fortney, J. J., et al. 2021, ApJ, 923, 269, doi: 10.3847/1538-4357/ac3140

Kingma, D. P., & Ba, J. 2017, Adam: A Method for Stochastic Optimization. https://arxiv.org/abs/1412.6980

Kruchten, N., Mease, J., & Moritz, D. 2022, in 2022 IEEE Visualization and Visual Analytics (VIS), 11–15, doi: 10.1109/VIS54862.2022.00011

Mahadevan, S., Ramsey, L., Bender, C., et al. 2012, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 8446, Ground-based and Airborne Instrumentation for Astronomy IV, ed. I. S. McLean, S. K. Ramsay, & H. Takami, 84461S, doi: 10.1117/12.926102

Majewski, S. R., Schiavon, R. P., Frinchaboy, P. M., et al. 2017, AJ, 154, 94, doi: 10.3847/1538-3881/aa784d

Marley, M. S., Saumon, D., Visscher, C., et al. 2021, ApJ, 920, 85, doi: 10.3847/1538-4357/ac141d

Morley, C. V., Mukherjee, S., Marley, M. S., et al. 2024, arXiv e-prints, arXiv:2402.00758, doi: 10.48550/arXiv.2402.00758

Mukherjee, S., Fortney, J. J., Morley, C. V., et al. 2024, arXiv e-prints, arXiv:2402.00756, doi: 10.48550/arXiv.2402.00756

NVIDIA, Vingelmann, P., & Fitzek, F. H. 2020, CUDA, release: 10.2.89. https://developer.nvidia.com/cuda-toolkit

pandas development team, T. 2020, pandas-dev/pandas: Pandas, latest, Zenodo, doi: 10.5281/zenodo.3509134

Paszke, A., Gross, S., Massa, F., et al. 2019, PyTorch: An Imperative Style, High-Performance Deep Learning Library. https://arxiv.org/abs/1912.01703

Satyanarayan, A., Moritz, D., Wongsuphasawat, K., & Heer, J. 2017, IEEE transactions on visualization and computer graphics, 23, 341

Shankar, S., Gully-Santiago, M., Morley, C. V., et al. 2024

Thompson, P., Cox, D. E., & Hastings, J. B. 1987, Journal of Applied Crystallography, 20, 79, doi: https://doi.org/10.1107/S0021889887087090

Van Rossum, G., & Drake, F. L. 2009, Python 3 Reference Manual (Scotts Valley, CA: CreateSpace)

VanderPlas, J., Granger, B., Heer, J., et al. 2018, Journal of Open Source Software, 3, 1057, doi: 10.21105/joss.01057

Virtanen, P., Gommers, R., Oliphant, T. E., et al. 2020, Nature Methods, 17, 261, doi: 10.1038/s41592-019-0686-2

Wes McKinney. 2010, in Proceedings of the 9th Python in Science Conference, ed. Stéfan van der Walt & Jarrod Millman, 56 – 61, doi: 10.25080/Majora-92bf1922-00a