NeurIPS – Ariel Data Challenge 2025: 4th Place Winning Solution

Thomas Dueholm Hansen

October 8, 2025

1 Team description

- Competition name: NeurIPS – Ariel Data Challenge 2025
- Team Name: Thomas Dueholm Hansen
- Public leaderboard score: 0.587
- Public leaderboard place: 6th
- Private leaderboard score: 0.590
- Private leaderboard place: 4th

The team consisted of a single team member:

- Name: Thomas Dueholm Hansen
- Location: ...
- Email: ...
- LinkedIn: `https://www.linkedin.com/in/thomas-dueholm-hansen-333634178/`

1.1 Academic and professional background

I received a PhD in computer science in 2012 from Aarhus University, Denmark. I then did one-year postdocs at Tel Aviv University and Stanford University, respectively, after which I returned to Aarhus University for a tenure track assistant professor position. In 2018 I moved to The University of Copenhagen, Denmark, for a position as associate professor of algorithms. My most notable academic achievement is a best paper award from the 43rd Annual ACM Symposium on Theory of Computing (STOC 2011), one of the two flagship conferences in theoretical computer science.

In 2017 and 2018 I traded actively with equities, long and short, index futures, and options. The associate professor position at The University of Copenhagen was offered permanently full-time, with a half-time leave of absence to be reevaluated yearly. Half of the time was to be spent on trading. My discretionary approach to trading did not work out, however, and I instead joined Nordea Markets, Denmark, in 2019 as a quantitative analyst. For about the first three years, I worked with fixed income, and in November 2021, I moved to FX spot as a quantitative trader.

I decided to take an extended sabbatical in 2023 in order to learn more about machine learning and to attempt to develop a profitable, automatic trading algorithm for FX. Since then I have worked on a mix of this private project and Kaggle competitions. The first time I did GPU programming was after leaving Nordea.

1.2 Prior experience

I have worked with time-series data in the above mentioned trading strategy project, as well as the following Kaggle competitions:

- *Child Mind Institute – Detect Sleep States*
- *Optiver – Trading at the Close*
- *HMS – Harmful Brain Activity Classification*
- *Jane Street Real-Time Market Data Forecasting*

In some sense I am starting to specialize in extracting information from noisy time series data, which was what appealed to me about the *NeurIPS – Ariel Data Challenge 2025*.

I also made use of experience from the competition *ICR – Identifying Age-Related Conditions*. Specifically, that competition taught me what *not* to do when working with a small dataset.

All of these experiences have contributed to proficiency with the automatic differentiation library JAX, understanding of how to limit overfitting, and various other machine learning skills.

From my time at Nordea, I have some experience with the Kalman smoother from the statsmodels python package. In this project I implemented a Kalman smoother myself with JAX. The automatic initialization of parameters for the Kalman smoother was something I invented for this project, i.e., it was not something I had used previously.

1.3 Time spent

About six weeks were spent on the competition.

2 Summary

I implemented a bespoke model for simulating the physics involved in a planet moving in front of a star. The model was implemented with JAX, and the automatic differentiation made it possible to fit simulated transits to the given data. The main challenge was that it was non-trivial to optimize the parameters of the model without ending up in an undesirable solution. Given the output from the physics-based model, I estimated transit depth with a very simple linear regression model and standard deviation with a neural network. In particular, the transit depth in my solution is independent of the standard deviation.

3 Overall model architecture

The idea for my solution is to implement a model that mimics the physics of a planet moving in front of a star, while explicitly taking limb darkening into account. This is done to obtain high quality estimates of the transit depth, which then require minimal use of modeling later in the pipeline. In fact, the final depth estimates for the submission are simply constructed

with linear regression on answers from the physical model. In particular, the depth is estimated independently of the standard deviation. The standard deviation is modeled with small neural networks. These downstream models were required to be simple due to the small number of planets in the training data.

Before applying the physics-based model, a starting point for the search is provided by a simpler model that involves Kalman Smoothers. I also implemented preprocessing similar to what was suggested by the organizers. Finally, I made use of principal component analysis to refine the spectrum produced by the physical model.

At a high level, my solution contains the following elements:

1. Preprocessing (Section 4)
2. Initial depth and transit window estimation (Section 5)
3. Depth estimation with explicit limb darkening (Section 6)
4. A simple linear regression model for producing the depth for the submission (Section 6.3)
5. A neural network for producing the standard deviation for the submission (Section 7)
6. A distinction between whether or not to use principal component analysis (PCA) (Section 8)

All in all, my approach ended up being relatively complex, but I believe that the use of explicit modeling of physics may be worth the effort to obtain good results. The main downside is that such an approach is unlikely to generalize to other problems, which makes it inherently less interesting.

4 Preprocessing

I generally follow the approach suggested by the organizers (see `https://www.kaggle.com/code/gordonyip/calibrating-and-binning-ariel-data`). I perform no binning, so the time series for FGS1 have length 67500, and the time series for AIRS have length 5625.

I treat hot and dead pixels as missing values. I replace such missing values with linear interpolation of neighboring pixels, four neighbors for FGS1 and two neighbors with the same wavelength for AIRS. Note that clusters of missing values remain missing. If one of the four most significant pixels are missing for AIRS, I remove that entire wavelength. The later model is set up to handle such missing values.

I use a rolling window of length 255 to discard outliers beyond five standard deviations from the centered rolling mean, again treating such values as missing.

Inspired by last year's first place solution (see `https://www.kaggle.com/competitions/ariel-data-challenge-2024/writeups/c-number-daiwakun-1st-place-solution`), I use the rim of the image to estimate foreground noise, which I then subtract from the center of the image. Except, I do not explicitly use the rim as in last year's first place solution. Instead I use the six rows with lowest intensity for AIRS, and the six columns and then six rows with lowest intensity for FGS1. Figure 1 illustrates the foreground noise processing step. Note that the foreground noise is not uniform, so I may actually throw away too much of the data.

I was pressed on time and did not put effort into verifying that this form of preprocessing was beneficial. It seemed reasonable, so I spent my time elsewhere.

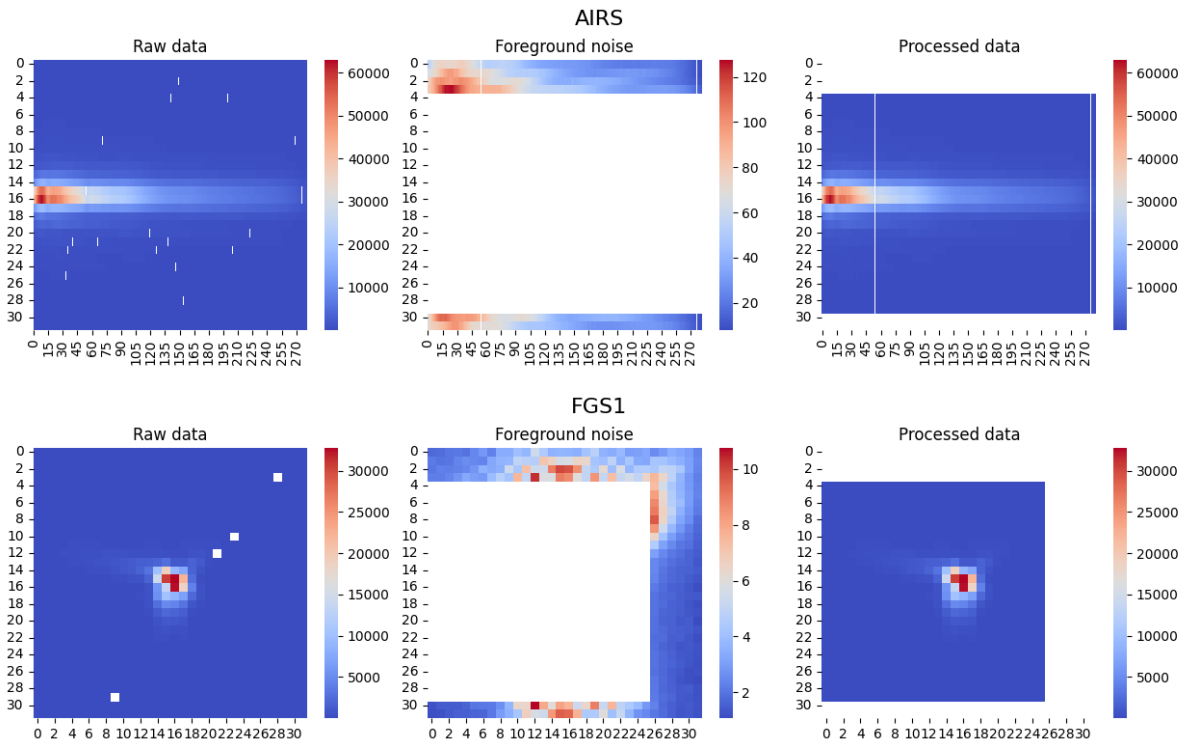

Figure 1: Foreground noise processing for planet 34983.

5 Initial depth and transit window estimation

My main model explicitly takes into account limb darkening, but it is non-trivial to optimize without ending up in an undesired local optimum. To improve the search, an initial solution is provided by a simpler model. This initial model was my baseline submission, and it is likely more complex than it needs to be when further refinement is performed later.

The idea for the initial model is to apply a Kalman Smoother to a Local Linear Trend model (see, e.g., `https://www.statsmodels.org/dev/examples/notebooks/generated/statespace_local_linear_trend.html`), to get a very smooth underlying level and trend from which the beginning and end of the transit can easily be extracted. The Kalman Smoother is designed to handle missing values, and the computation is performed twice: Once with all the data, and once with the data from the transit removed (set to NaN). The depth of the transit is extracted from the difference of the levels resulting from these two computations.

The beginning, middle, and end of the transit are identified with the first Kalman Smoother. I identify as breakpoints the indices of the minimum and maximum values of the trend component. The middle of the transit is defined to be the average over these breakpoints, including both AIRS and FGS1. For AIRS, I use the sum across the entire spectrum to decrease noise. The beginning of the transit is the latest point prior to the first breakpoint where either the trend reaches zero, or the one-step difference of the trend is above that of the breakpoint. The end of the transit is defined similarly, but for the earliest point after the second breakpoint. The data from AIRS and FGS1 is combined by choosing the latest of the two beginnings and the earliest of the two ends. Transit identification is illustrated in Figure 2.

Robustness of transit identification relies on the produced curves being very smooth, and the Kalman Smoother is parameterized to produce such smooth curves. The Local Linear Trend

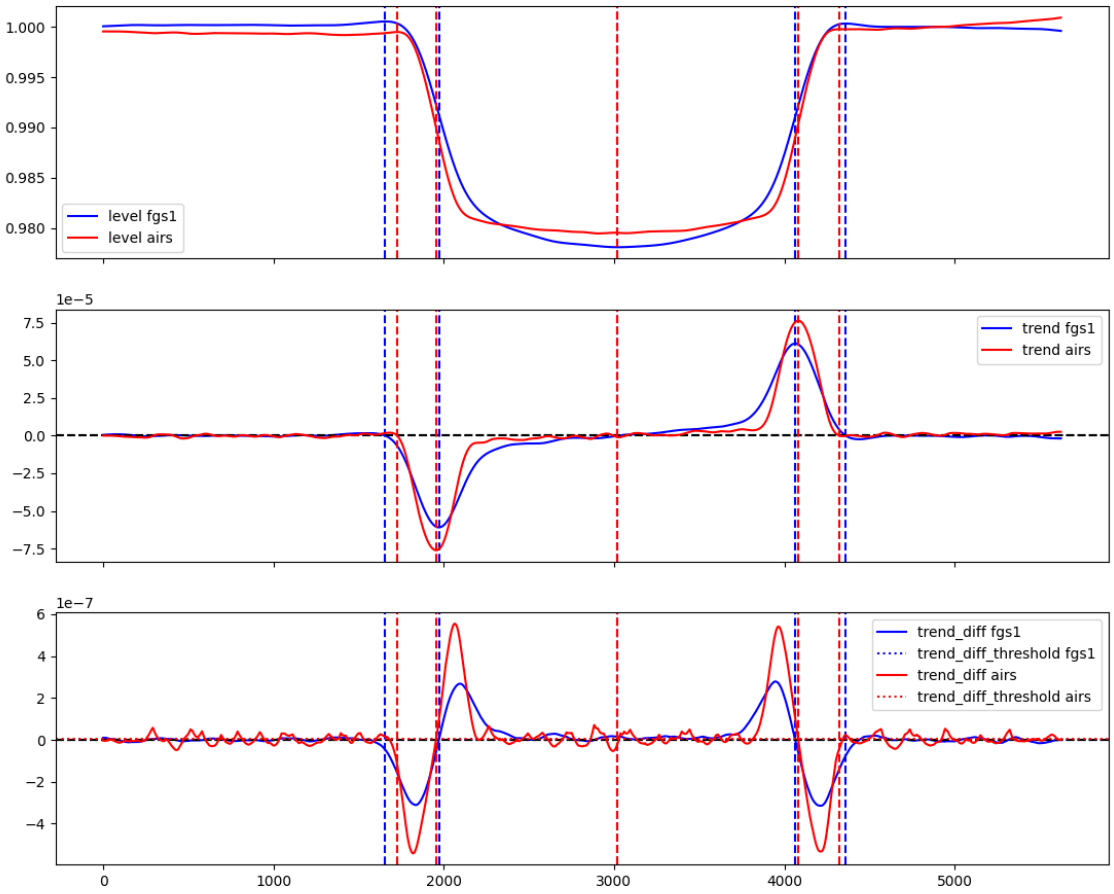

Figure 2: Transit identification for planet 34983.

model is defined by the linear system:

$$\begin{aligned} y_t &= \mu_t + \epsilon_t & \epsilon_t &\sim N(0, \sigma_\epsilon^2) \\ \mu_{t+1} &= \mu_t + \nu_t + \xi_t & \xi_t &\sim N(0, \sigma_\xi^2) \\ \nu_{t+1} &= \nu_t + \zeta_t & \zeta_t &\sim N(0, \sigma_\zeta^2) \end{aligned}$$

Where y_t is the observed data, μ_t is the level, and ν_t is the trend. The smoothness is determined by the noise parameters σ_ϵ, σ_ξ, and σ_ζ. Instead of fitting such parameters to the data, I generate the parameters for a particular graph from three corresponding hyperparameters and a window length k. More precisely, I compute a moving average with the given window length k as a stand-in for the level, and a moving average of that as a stand-in for the trend. σ_ϵ, σ_ξ, and σ_ζ are products of the respective hyperparameters and the following quantities. For σ_ϵ, the variance around the stand-in level. For σ_ξ, the maximum absolute change in stand-in level. For σ_ζ, the maximum absolute change in stand-in trend. I did not get around to optimizing the hyperparameters, so they were set manually.

Additionally, the Kalman Smoother requires μ_0 and ν_0 for initialization. Let $y_1, y_2, \ldots, y_N$ be the sequence of observed data. μ_0 and ν_0 are set to:

$$\begin{aligned} \mu_0 &= \frac{1}{k} \sum_{t=1}^{k} y_t \\ \nu_0 &= \left(\left(\frac{1}{k} \sum_{t=N-k+1}^{N} y_t \right) - \mu_0 \right) / (N-k) \end{aligned}$$

Finally, the Kalman Smoother requires as input the covariance matrix Σ_0 for μ_0 and ν_0. This is set to the steady state for σ_ϵ, σ_ξ, and σ_ζ, which is computed by solving Discrete Algebraic Riccati Equations (DARE). (See, e.g., section 2.2.1 of `https://www.robots.ox.ac.uk/~ian/Teaching/Estimation/LectureNotes2.pdf`.)

In some cases the beginning or the end of the transit falls outside the available data, and it becomes meaningless to apply a Kalman Smoother after removing the transit. In that case I replace the second Kalman Smoother by a line with slope extracted from data near the middle of the transit, and the highest intercept that allows the line to hit data either at the beginning or the end of the transit. An example of such an incomplete transit is shown in Figure 3. In this case, the second Kalman Smoother is replaced by the orange dashed line.

6 Depth estimation with explicit limb darkening

6.1 Transit simulation

My model that explicitly takes into account limb darkening simulates a planet moving in front of a star, and estimates the amount of light observed at every time step. Limb darkening means that the intensity of the star is highest at the center and is reduced toward the edge (the limb). The formula that I use for limb darkening is based on `https://www.astro.uvic.ca/~tatum/stellatm/atm6.pdf`. The exact formula used is:

$$intensity(x, u, v) = 1 - u \cdot ((1-v) \cdot (1 - \sqrt{1-x^2}) + v \cdot x^2),$$

where x is the distance to the center of the star, assuming that the star has radius 1, and u and v are parameters that depend on the star. Unfortunately, I made a sign error in my code and

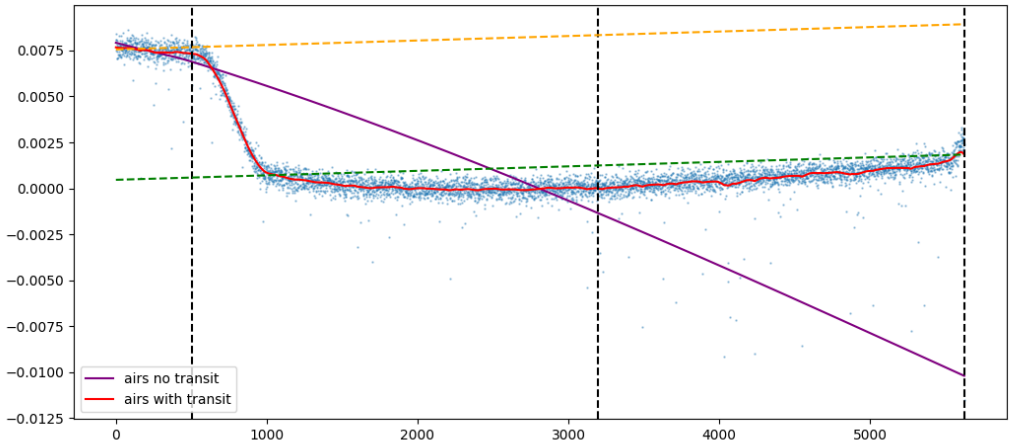

Figure 3: Example of incomplete transit for planet 1738121950.

changed the last plus into a minus, which meant that v behaved unexpectedly, so I gave up on directly optimizing v and instead treated it as a hyperparameter.

The model for constructing the transit uses six parameters:

- r: The radius of the planet, scaled such that the radius of the star is 1.
- u: The limb darkening parameter u described above.
- v: The limb darkening parameter v described above.
- d_{min}: The closest distance from the center of the planet to the center of the star during the transit.
- t_{min}: The time at which the planet is closest to the center of the star. The timeframe goes from 0 to 1.
- t_{length}: The transit starts when the planet first overlaps the star, and ends when the planet no longer overlaps the star. The transit length t_{length} is the time between these two points. Note that the transit length can be greater than 1 if the timeframe does not cover the entire transit.

The parameters are assumed to satisfy:

$$\begin{aligned} 0 &< r \\ 0 \leq u &\leq 1 \\ -1 \leq v &\leq 1 \\ 0 \leq d_{min} &< 1 + r \end{aligned}$$

Importantly, the transit is differentiable with respect to these six parameters, which means that the parameters can be optimized to produce the transit with the best fit for some training data.

Figure 4 shows examples of the transits generated by my model for the same parameters, without and with limb darkening. Note that without limb darkening, the valley of the transit

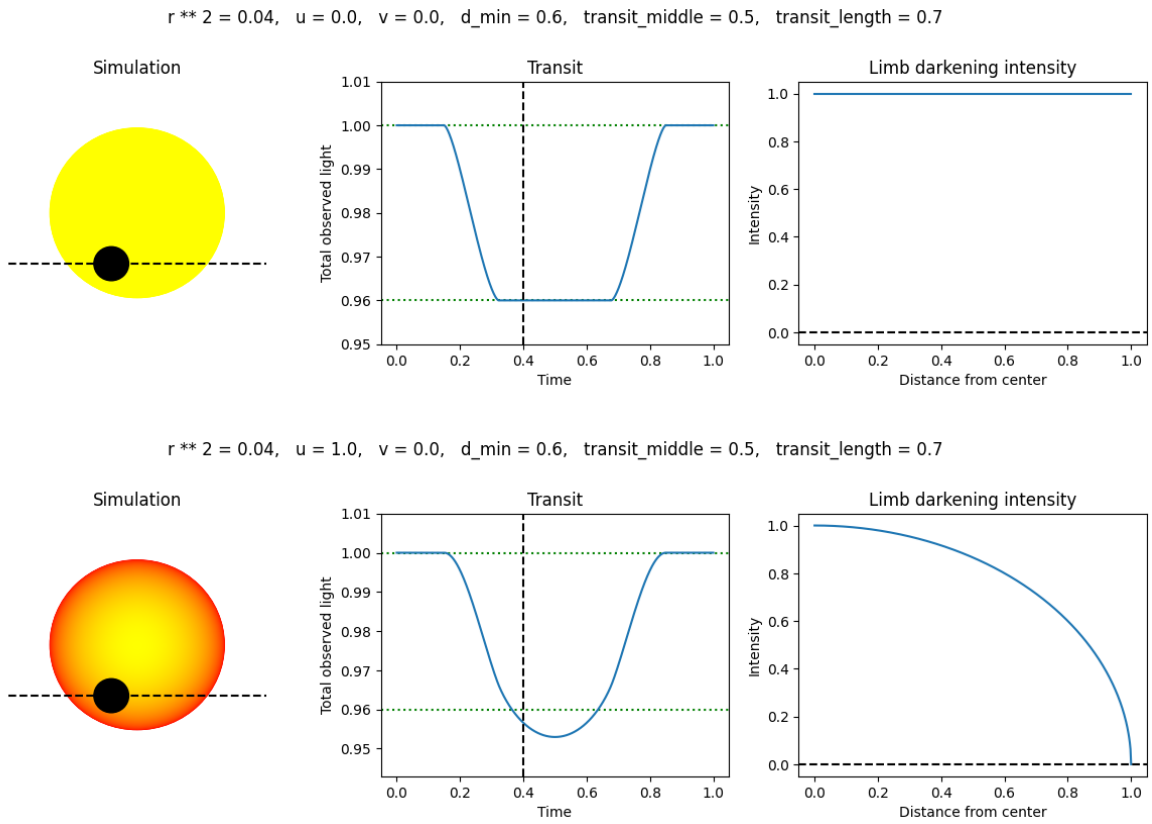

Figure 4: Example of a simulated transit without and with limb darkening.

is very flat, because the observed light mainly depends on whether the planet is fully inside the boundary of the star or not. With limb darkening, the observed light continues being reduced as the planet gets closer to the center of the star, because it then covers an area with higher intensity.

6.2 Parameter optimization

The parameters optimizing the fit of the simulated transits to the given signals are refined in several stages. We are given a time series for FGS1 and 282 time series for AIRS, defining a spectrum, as well as the transit depth, transit start, and transit end from the initial model described in Section 5. Our first goal is to compute transits for three time series:

1. FGS1.
2. The sum of AIRS over the spectrum.
3. The weighted sum of AIRS over the spectrum, with weights being the inverse of the mean of the signals, i.e., all 282 time series should have equal importance regardless of magnitude.

The r^2 from the first and last transits are used directly in the linear regression for producing the final depth for the submission. The second transit is included because simply summing AIRS across the spectrum has a lower noise than the other two time series, and basing the computation of d_{min}, t_{min}, and t_{length} on the least noisy data produces the best result.

The transits are optimized jointly, so that they share parameters d_{min}, t_{min}, and t_{length}. v is a frozen hyperparameter, and r and u are optimized independently for the three time series. I minimize a weighted sum of the mean squared errors from the three time series, with a higher weight on the second time series. The transit itself is a graph between 0 and 1, so in order match the signal, it is scaled by a polynomial f whose coefficients are computed with linear regression. These coefficients are computed independently for each of the three time series, and they are not treated as parameters, i.e., the transit is defined by its parameters, and the polynomial f is then chosen to minimize the mean squared error against the signal. Additionally, I apply L2 regularization to the coefficients of f for terms with degree at least 2.

To initialize the optimization, t_{min} and t_{length} are derived from the given transit start and transit end. r is defined by the given depth, and u and d_{min} are set relatively arbitrarily to typical such values. The mean squared error is minimized with Adam, with $\beta_1 = 0.8$ and a cosine learning rate schedule with warm-up. We perform three stages of optimization, each for 100 iterations, with an increasing degree of freedom for the polynomial f. The reason that f is not just unconstrained is that it would then adjust to the signal dip, before the appropriate depth was reflected in the parameters. For the first stage we also only use data from inside the transit, since the degree of f is limited to 1. The optimization parameters for the three stages are as follows. Each stage starts with the parameters from the previous stage.

Stage	Learning rate	Polynomial degree	Regularization	Transit data only
1	0.011	1	Not relevant	Yes
2	0.006	4	3000	No
3	0.001	4	75	No

Next, we perform similar computations for all 282 time series in the AIRS spectrum. The initial parameters are taken from the third time series above and broadcasted to all 282 time series. The parameters for d_{min}, t_{min}, and t_{length} remain frozen throughout the rest of the computation. We will also always be using all the data, unlike the first stage above, and the polynomial degree and regularization will remain 4 and 75, respectively. For the fourth stage, u also remains frozen. Thus, we compute r and the polynomial f independently for each time series. The fourth stage again consists of 100 iterations, and it uses the following learning rate.

Stage	Learning rate	Iterations
4	0.007	100

Next, we want to incorporate the *gain drift* into the computation. This part is based on last year's first place solution: `https://www.kaggle.com/competitions/ariel-data-challenge-2024/writeups/c-number-daiwakun-1st-place-solution`. They observed that the data generated by the ExoSim2 simulator used a gain drift factor $1+f(t)\cdot g(\lambda)$, where f is a polynomial of time, similar to what I used above, and g is a polynomial of the wavelength. They used polynomials of degree four, and so will I. At time t and wavelength λ, the overall prediction is then of the form:

$$y_{pred}(t,\lambda) \;=\; I(\lambda)\cdot transit(t, r_\lambda, u_\lambda, v, d_{min}, t_{min}, t_{length})\cdot(1+f(t)\cdot g(\lambda)),$$

where $I(\lambda)$ is the intensity of the star at wavelength λ, $transit(t, r_\lambda, u_\lambda, v, d_{min}, t_{min}, t_{length})$ is the transit factor at time t and wavelength λ, and $(1+f(t)\cdot g(\lambda))$ is the gain drift.

In order to stabilize the computation, I alternate between optimizing the coefficients of f and g and optimizing the vectors r and u. When optimizing r and u, the intensity $I(\lambda)$ is derived independently for each λ with linear regression, as it was done for f in the first three stages, i.e., $I(\lambda)$ is treated as a polynomial of degree zero. When optimizing f and g, the intensity and the transit are fixed from the previous stage, and the coefficients of g are again derived with

linear regression. Thus, we never treat coefficients of g as true parameters. I perform three such alternations with the following learning rates.

Stage	Optimized parameters	Learning rate	Iterations
5	f	200	120
6	r and u	0.005	50
7	f	0.1	60
8	r and u	0.002	50
9	f	0.03	60
10	r and u	0.001	50

I regularize the vector u during these last stages by adding $200 \cdot \text{var}(\text{diff}(u))/\text{mean}(u)$ to the error, where $\text{var}(\text{diff}(u))$ is the variance of one-step differences of u, and $\text{mean}(u)$ is the mean of u. This is done to ensure that u changes smoothly over the spectrum. r remains noisy, however, at this stage. I also use L2 regularization for f and g. For f regularization is applied to terms with degree at least 2, and for g regularization is applied to terms with degree at least 1.

6.3 Spectrum smoothing and final depth estimation

The depth, i.e., r^2, provided to the submission is obtained with simple linear regression on the r^2 parameters produced in the previous section. For planets with multiple instances, the average depth is used for the submission. Before applying linear regression, the AIRS spectrum is smoothed with a Kalman Smoother in the same way as described in Section 5. Recall from Section 4 that parts of the spectrum are missing, in case there were important missing pixels. This issue is fixed by the Kalman Smoother, since it works even with missing values.

The linear regression for AIRS uses the features:

1. r^2 from the third transit, weighted sum of AIRS, after stage 3 in Section 6.2, broadcasted to the entire spectrum.
2. The mean of the smoothed r^2 for the spectrum, broadcasted to the entire spectrum.
3. The difference between the smoothed r^2 for the spectrum and its mean.
4. Possibly a feature derived with principal component analysis (PCA) as described in Section 8.

The linear regression for FGS1 uses only a single feature:

1. r^2 from the first transit, FGS1, after stage 3 in Section 6.2.

The training data contained outliers that reduced the quality of the solution. This was handled simply by omitting the 18 planets with worst mean squared error from the linear regression.

7 Computation of uncertainty

The standard deviation component of the solution is computed with two independent Neural networks, one for AIRS and one for FGS1. I use 11 features for AIRS and 7 features for FGS1. The features are described below.

The neural networks are essentially multilayer perceptrons (MLPs), but with only a single hidden layer. Instead of using a single activation function, the hidden layer computes two

vectors in parallel, applies a sigmoid activation function to one and relu to the other, and takes the product of the two vectors before applying dropout. I also use batch normalization for both vectors. The output is transformed with one plus an elu activation function and scaled by a parameter. The code is simple enough that it is worth showing in its entirety in Figure 5.

The inputs for the MLPs are normalized with a StandardScaler, i.e., I subtract the mean and divide by the standard deviation. The inputs are clipped by the MLP itself to simplify hyperparameter optimization. Hyperparameter optimization was performed with manual experimentation. I initially planned to use Optuna, but I found satisfactory parameters before I got around to it. I used five fold cross validation with three repetitions. I selected features in the same way.

For AIRS, the features are clipped at plus-minus 7 standard deviations, the hidden dimension and dropout are 256 and 0.4, and the log_scale parameter is -9. For FGS1, the features are clipped at plus-minus 5 standard deviations, the hidden dimension and dropout are 32 and 0.1, and the log_scale parameter is -8. In both cases I use an Adam optimizer with weight decay. For AIRS I trained for 5 epochs, and for FGS1 I trained for 150 epochs. Note that there was much more training data for AIRS due to working with a spectrum, which explains why it worked better with fewer epochs.

I use 11 features for AIRS, the most important feature being the limb darkening coefficient u. The features are based on the output of the limb darkening model described in Section 6. Instead of directly using the output of the depth estimation model, I use five fold cross validation with three repetitions to generate three times as much training data with out-of-sample properties. The features are as follows:

1. The predicted value of r^2 for the planet. If there are multiple instances of the planet, then the mean of such predictions.
2. The absolute value of the difference between feature 1 and the predicted r^2 for the particular instance, i.e., if there is only one instance for a planet, then this feature is zero.
3. The predicted value of the limb darkening coefficient u.
4. The predicted value of the transit length t_{length}.
5. The predicted minimum distance d_{min}.
6. The multiplicity of the planet, i.e., how many instances of data that are available.
7. The standard deviation of the predictions for r^2 for the spectrum.
8. The index of the data point in the spectrum.
9. For a particular index in the spectrum, the standard deviation of the predictions for r^2 for that index, but ignoring outliers.
10. A measure for the non-linearity of the polynomials for the gain drift: The sum over the outer product of the absolute values of coefficients from f with degree at least 2 and the absolute values of coefficients from g with degree at least 1, normalized by the mean of the raw signal.
11. The predicted spectrum: The difference between the predicted value of r^2 for individual indices and the mean over the entire spectrum.

```python
import flax.linen as nn
import jax.numpy as jnp
from typing import Tuple

class MLP(nn.Module):

   input_clipping_threshold: float
   n_hidden: Tuple[int]
   dropout: Tuple[float]
   log_scale: float

   @nn.compact
   def __call__(self, x, train: bool):
      x = x.astype(jnp.float64)
      x = jnp.clip(x, -self.input_clipping_threshold, self.input_clipping_threshold)

      for n_hidden, dropout in zip(self.n_hidden, self.dropout):
         x_sigmoid = nn.BatchNorm(use_running_average=(not train))(x)
         x_sigmoid = nn.Dense(features=n_hidden, use_bias=True)(x_sigmoid)
         x_sigmoid = nn.sigmoid(x_sigmoid)

         x = nn.Dense(features=n_hidden, use_bias=True)(x)
         x = nn.BatchNorm(use_running_average=(not train))(x)
         x = nn.relu(x)

         x = x * x_sigmoid

         x = nn.Dropout(rate=dropout, deterministic=not train)(x)

      x = nn.Dense(features=1, use_bias=True)(x)[..., 0]
      x = 1 + nn.elu(x)

      log_scale = self.param('log_scale no_weight_decay',
          nn.initializers.constant(self.log_scale), (), jnp.float32)

      x = jnp.exp(log_scale) * x

      return x
```

Figure 5: Modified multilayer perceptron.

For FGS1 I use the same features as for AIRS, computed with data from FGS1, but I only use the 7 first features. Note that the 7th feature describes the standard deviation of the predicted spectrum, which is computed independently of FGS1, but it was still helpful to include it as a feature for FGS1.

8 Principal component analysis (PCA)

Applying principal component analysis (PCA) to the ARIEL target spectrum revealed that a small number of eigenvectors explained most of the variance. This is because certain molecules are more likely to make up the atmosphere of a planet. The idea of using PCA came from last year's second place solution: `https://www.kaggle.com/competitions/ariel-data-challenge-2024/writeups/jeroen-cottaar-2nd-place-solution-pure-bayesian-in`. By target spectrum, I mean the matrix obtained by subtracting for each row of the target matrix the mean of that row. I only do this for the AIRS part of the data. I chose to use 7 components, which explained more than 99.5% of the variance.

For each AIRS spectrum obtained from the limb darkening model from Section 6, I fit a linear regression model on the 7 most significant eigenvectors to the spectrum of interest, i.e., I find the best way to approximate the spectrum with spectra from the original target matrix. I then use this approximation as a replacement for the given spectrum, the idea being that the original target data is more accurate than the estimates produced by the limb darkening model.

It is possible that the planet in question is not well represented by the training data. In that case relying on the PCA spectrum replacement would produce worse results. I therefore check whether the Euclidean distance of the original spectrum and the replacement spectrum is below the threshold 0.0015. The threshold is chosen such that about 90% of the training data itself would use PCA replacement. I use the PCA replacement spectrum if that is the case, otherwise I only use the original spectrum. To make this distinction I train two entirely separate models on all the data: One with and one without the PCA spectrum.

When using the PCA replacement spectrum, it does not technically replace the original spectrum. It appears as an additional feature in the linear regression model for depth estimation. The PCA spectrum generally dominates the original spectrum, however, being assigned about 20 times as high a weight.

9 Ensembling

Both with and without PCA I train five models and average the result, so in total I train ten models with different seeds. This number could have been much larger while staying within the time limit, but I did not experiment with this aspect.

10 Computation time

I processed data and trained models locally with an RTX 5090. In total, the running time of a Kaggle notebook with a P100 was about twice as long as with my RTX 5090. I expected a higher speedup with a significantly faster GPU, but the code has not been properly optimized (the electricity consumption was very low), and calculations are generally performed with 64 bit precision. The time spent on the individual steps with an RTX 5090 was:

- Preprocessing (Section 4): 17 minutes.
- Initial depth estimation (Section 5): 1 minute.

- Depth estimation with limb darkening (Section 6): 3 hours and 35 minutes.
- Model training: 11 minutes.

11 Reproducibility

During the competition, I noticed that the standard deviations produced by Kaggle notebooks differed from those produced locally. I thought this was due to using different batch sizes, but even with the same batch sizes there was a difference. Since the end of the competition, there was an automatic update of Ubuntu that broke my Nvidia drivers, and I had to reinstall the drivers. Now there is a slight difference between the standard deviations produced locally by the same code before and after the driver update, so the issue with Kaggle notebooks may also have been driver related.

I was careful to keep track of code and seed changes, since I had trouble with reproducibility in the *Jane Street Real-Time Market Data Forecasting* competition, but I did not keep track of drivers. The new driver is nvidia-driver-580-open. I believe my previous driver was nvidia-driver-575-open, which is no longer available via ubuntu-drivers. I also did a full purge of Nvidia software on my system in general. All in all, I decided that it was not worth trying to replicate my previous setup.

I removed some unused code for the final version, but I verified that this did not change the produced standard deviations.

12 Conclusion

The limb darkening model from Section 6 produces good depth estimates by itself, and one can imagine further extensions of the model that include additional physics simulations. The way the transit is computed with a discrete integral is very memory inefficient, and it is the performance bottleneck for my submission. It is a small piece of code, and it should be possible to significantly speed up this computation, maybe even by writing a small custom kernel.

The downside of the limb darkening model is that it is very specific to this problem, and achieving high quality results is quite tedious. General purpose models are more likely to be optimized and to be well understood by the research community, so using a very bespoke model is a definite disadvantage. Perhaps the real underlying research question here is to develop optimization techniques that perform well on a wide range of bespoke models. In order for that to be a successful area of research it would require that a large number of bespoke models were made easily available for research. That is clearly outside the scope of the Ariel Challenge, however.

It is a definite downside to my approach that the depth estimate is computed entirely independently of the standard deviation. I would expect that improvements can be made on this front, possibly be adding a correction that is computed together with the standard deviation. I briefly tried this, but it caused overfitting, so I gave up on the idea. I would also expect that it should be possible to make a better model for standard deviation by itself. I generally did not put much time into this area.

13 Are we alone in the universe?

The Fermi Paradox asks "Why are there no signs of intelligence elsewhere in the universe?" The opportunities for life to appear are literally astronomical. I would argue, however, that

astronomical numbers are quite small in this context, since they are only polynomial in nature. The probability of intelligent life appearing may very well be exponential in nature, for example if an initial critical step is for a sequence of molecules to randomly come together in a specific order. Even with slower growing functions than exponentials, the inverse of the probability of intelligent life may dwarf any number in the astronomical range. Thus, it may be plausible that the answer to the question "Are we alone in an infinite universe?" is no, while the answer to "Are we alone in the observable universe?" is yes with very high probability. This all assumes that life appears spontaneously. In a distant future it seems plausible that humanity will seed other planets with life. Perhaps life on Earth itself originates from such seeds planted by other civilizations or higher entities. Just something to keep in mind: If life was seeded by ancient civilizations, how should that affect the way that we search for it?