**Predictive modelling of sea debris around Maltese coastal waters**

**The Data**

**u:** The east-west component of the current velocity, representing movement in the eastward (positive u) or westward (negative u) direction. It's a measure of how fast the water is moving across the Earth's surface in the horizontal plane from west to east or vice versa.

**v:** The north-south component of the current velocity, representing movement in the northward (positive v) or southward (negative v) direction. Like ‘u’, it measures the speed of water moving in the vertical plane from north to south or the opposite.

Together, u and v components allow for the representation of the direction and speed of sea surface currents at a given point. By combining these two components, one can calculate the total velocity and direction of the current at that point.

The magnitude (or speed) of the current can be calculated from the u and v components using the Pythagorean theorem. If you consider u and v as the legs of a right triangle, the magnitude of the current (hypotenuse of the triangle) is the square root of the sum of the squares of u and v. The direction of the current, measured as an angle from the north (often in degrees or radians), can be calculated using the arctangent of v over u, which gives the angle θ. These are all calculated automatically in the Lagrangian model.  
  
**Dimensions:**

**time:** Hourly Intervals (24 hours)

**lat:** 52 latitude points, indicating the geographic scope in the north-south direction.

**lon:** 43 longitude points, indicating the geographic scope in the east-west direction.

**Original Boundaries:**

Latitude Range in the Dataset: 35.74470138549805 to 36.88019943237305

Longitude Range in the Dataset: 13.676799774169922 to 15.380399703979492

**Updated for Lagrangian (used in 7-day simulation):**lon\_min, lat\_min, lon\_max, lat\_max = 14.15, 35.79, 14.81, 36.3

**For simulation to prevent stuck boundaries (used in 7-day simulation):**

extent = [14.15, 14.71, 35.79, 36.2]

**Final polygon (red) for predictions:**

polygon\_coordinates = [

(14.6, 35.87),

(14.35, 36.01),

(14.35, 36.09),

(14.6, 36.09),

(14.6, 35.87)

]Add some information about the sea surface currents preprocessing file! (Remember that the nan values are not removed in this). This notebook is only to merge the data we have into a single file!

Add graphs for how the data is and explain why it was taken from 2020 till august 2023. (since there is a lot of missing data at the end of 2023). Actual Data starts at February 25th, 2020, since that is the first date of the data that was given.

* Would be a good idea to do something like:  
  A graph of a number of data

  Description automatically generated
* Radar locations: [Portus 3.0 (um.edu.mt)](https://portus.research.um.edu.mt/?p=14.161,35.911,10&basemap=Aerial&timestamp=1710926237)
* Missing (nans) data is likely due to data being close to the coast.

**Lagrangian Model Simulation**

* Mention pre-processing of the data, how the current data was merged and the length (7 days for brevity).
* Simulation can be for same length used to train AI model or for brevity just simulate for only 7 days.
* Originally the idea was to use and combine wind and current data, but we decided to only use the sea surface currents data. This decision was based on two factors. Firstly, sea surface debris is proven to be more affected by sea surface currents instead of wind as seen in *[https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0111913]… In this study, researchers analysed data collected during five expeditions across all major ocean basins between 2014 and 2015 using a variety of methods including visual surveys, net sampling, and acoustic imagery. They found that "the majority of microplastics were concentrated in subtropical gyres" and that "wind-driven processes played only a minor role in determining the spatial distribution of microplastics." This finding suggests that while wind does contribute to some extent, it has less significance compared to ocean currents when discussing the distribution of sea surface debris.* Although it must be said that the wind data would make this more accurate in a real-life scenario. Secondly, we decided to do this because it is very hard to build a custom behaviour kernel.
* Original simulation was for 3 years but after some consideration, this proved to be too long and pointless. This decision was originally taken since initially we wanted to make predictions for a month but after some consideration this was not possible to implement and therefore, we decided to simulate the Lagrangian model for a shorter period of 7 days and make a prediction for of 1 day instead.
* First step was to create a land-sea mask. This was done by using a shapefile, this was then rasterised, and land is represented by 1 and sea by 0. Then, it was converted to NetCDF format and added within the grid boundaries to match with the boundaries of the data set. Show outputted image here. Can also mention how shapefile was used as map of Malta in the visualisations.
* Important to mention that due to the radars only north part of the island there is data missing in some points. Here can also show a visualisation!
* Created the Lagrangian pipeline, and custom kernels (explain more in detail how they are deleted at boundaries and how the land sea mask reflects or deletes. Find reference for probability of being deleted, beached, or reflected).
* From simulations we noticed how particles were getting stuck at boundaries. This was found out to be since we were using all the dataset there was no data at the borders, so the particles were not affected. This was solved by using a bit smaller boundary (by 0.1).
* Originally, I was going to have multiple particles random but then it was more realistic to have around 50 (1 particle represents a group of debris) starting from very close proximity to simulate how clusters of debris move.
* Simulation run for period of dataset, dt (elaborate) saved every hour? Elaborate as well.
* When it comes to the missing data in the Lagrangian model, (find reference) the model was built in way of interpolating the data based on values around it so no preprocessing is necessary when it comes to this. (DOUBLE CHECK)
* Added interpolation to handle nans!
* [European Marine Observation and Data Network (EMODnet) (europa.eu)](https://emodnet.ec.europa.eu/en) for where particles start.
* One point along ship route
* See around Sikka l-Bajda
* 7 day (first Lagrangian) is dates from 1st Jan 2023 till 7th Jan 2023. This needs to be modified when doing evaluation to match a week’s data from the drifter data! Also needs to start from the same location.
* For some reason when we tried to interpolate the data (for the 7 days) we got very different visualisation results for the data with nans and data that was interpolated. We also got very similar results where the particle movement was always the same even if we tried different dates of data. Basically, the interpolated data (tried both linear and spline) gave us the same result regardless of the time frame of the data! This will be further proven by using the drifters for the evaluation… but for now the decision remains that nan values will not be removed for the Lagrangian Simulation! *The decision not to remove NaN values for the Lagrangian Simulation can be informed by the fact that interpolation, whether linear or spline, has been consistently homogenizing the data, leading to non-representative or misleading results regarding particle movements across different timeframes. This uniformity post-interpolation can mask the true variability and dynamic nature of the original data, which is critical in accurately simulating and understanding the drifters' behaviour and movement in varying oceanic conditions. Hence, retaining NaN values preserves the integrity of the original dataset, ensuring that the Lagrangian Simulation more accurately reflects the real-world conditions and variations that the drifters would encounter.*

**Lagrangian Model Evaluation**

* Use trajectories for drifters, start particle (drifter) with the same time, date, place and visualise.
* Compare trajectories with a particle/s.
* Drifters are not affected by wind that much so are good for evaluation of Lagrangian.

**AI Model**

* Original idea was to train a single model on 3 years and then predict 1 month in the future. This was unfeasible. The next test was to train a model on 7-day dataset and predict 1 day but making one model was hard due to the dimensionality of the data (4-D) and the predicted results were not accurate at all. Therefore, we decided to move to a smaller portion of the data (smaller boundaries), but longer period to predict the ‘u’ and ‘v’ components individually (for every pair of lon and lat) and then combine the predictions into one file to make the simulation from it.
* For this we decided to try and use 6 months of data and then also we decided we could test using 6 months but skipping a month (jan,mar,may…) because of the seasonal changes in the data.
* The prediction is for 24 hours. The prediction will be converted into NetCDF format so that we can simulate it using the Lagrangian model.
* First, we built a normal LSTM to make a single target prediction ‘u’. Then we tried the same LSTM for two target predictions ‘u’ and ‘v’. Then we tried to find the best model (LSTM,GRU…) and the best hyperparameters. Finally, we run a model for every pair and then merge into single output.
* Data extracted from 2023-01-01 till 2023-06-30 (6 months)
* Pre-processed data by dropping useless columns and removing rows where u and v are nan. First experiment we also dropped v so that data will be time and u to predict u.
* Removing nan values resulted in some points having way less data this could be seen on the visualisation.
* Number of pairs was reduced to 37…
* Because dropping nans resulted in a lot of missing data, we decided to use 1 year (22-01-01 till 22-12-31) worth of data to ensure there was enough data for every pair (since different pairs had different amount of data)
* Due to several nans the model input ended up only having on average around 4k inputs per lat and lon pair, this led to a very small validation and training loss value in the first epoch. To counter this we decided to use 2 years’ worth of data (2021 and 2022).
* The model for a single target prediction gave very good results. When the model was updated to have two targets (u & v) to predict, the results were not as good. This gave thought to have separate models for both variables… to ensure that the results are accurate as possible.
* After multiple tests, it was discovered that a sequence length (window\_size) of 1 gives the best results.
* We also tried to use ‘v’ and ‘u’ (both as features) and from this we got slightly better results, so we concluded that they have a better effect and thus we decided to shift focus on using 2 feature to predict a single feature.
* From multiple tests it was concluded that a learning rate of 0.001 performs the best.
* Run same tests but this time with GRU and it was concluded that GRU has overall better performance.
* Regularization techniques (e.g., dropout, L1/L2 regularization) was added help mitigate overfitting since small window sizes might lead to overfitting.
* Due to the Nans in the dataset we also tried to use interpolation to fill these values (spline) but it was concluded that the results were not good and worse than when we drop the nans. This is because… (give reason).
* Important to mention that all these tests where run on the same pair of lon and lat:

target\_lat = 36.03409957885742

target\_lon = 14.528599739074707

* Also tried LSTM with diff dropout and regularisation and got very similar results so we decided to go with the results that yielded the best overall results.
* Ideal batch size was 10 after some testing.
* It was decided that the prediction pipeline should change. Instead of creating a model to predict the next hour based on the previous hour, the new model takes the last 72 hours and makes 1 prediction for 24 hours… *for now I decided to fill in nans with mean value, but this might change after some testing. I am also thinking of adding rolling forecasting… basically completely changing the sequence and feature inputs!*
* After some consideration and testing, a LSTM was created that was able to make predictions based on the last 24 hours and then use rolling forecast to predict the next 24 hours. The model had reasonably good results (add results) and a discovery was made, since we are working with sea surface currents it will be very hard to predict the next day’s weather (even with rolling predictions) just because of the fact of how unpredictable it can be. For example, one day can, be calm and suddenly, not visible over couple hours the values are way higher. This could be studied further where other weather conditions could be used with the data and the training so that it is more evident if there is going to be a switch in the weather (add more useful input features).
* Changed to a longer window size of 72 days instead of 24 and got better results. Added interpolation to the input file since we need 72 and will not work if we drop nans.
* The new dates are:
* 2020/01/01 till 2023/08/01 🡪 The training data
* Input (72 hours from last sequence in test data) 🡪 2023/08/01 till 2023/08/03
* Actual (To compare) 🡪 2023/08/04.
* Explain exactly why I took (started from) the last sequence of the test set… although it can be any random day and it will work… we did this for brevity.
* Now experimenting with using GRU Layers instead of LSTM since they can be better for short sequence forecasts and take less time to train which will be useful when we make the pipeline for the 37 models.
* After some tests it was concluded that, although GRU’s take less time to train, the results were not as good as the tests that were done on the LSTM. GRUs simplify calculations compared to LSTMs, potentially speeding up training. However, this simplicity might limit their ability to model complex sequences with long-term dependencies as effectively as LSTMs, resulting in reduced performance on tasks where such intricacies are crucial.
* Since we are going to have to run an estimate of around 37 separate models to get a combined file that can be used for the Lagrangian simulation, the time that the models took to train was taken into consideration. Therefor a time of 20 mins was chosen to be the ideal time so that in total it will not take more than 12 hours to train the final multiple models. Early stopping was used to ensure that every model will have the best possible training and to prevent over fitting if the data for the specific point is less.
* It is for this reason that a learning rate of 0.001 was chosen instead of smaller rates like 0.0001 or 0.0005. This balance aims to reduce convergence time and improve results, as overly small learning rates may lead to slower progress and potentially getting stuck in less optimal minima.
* Created a loop pretty much does everything the same as the lstm we worked with before… the only difference is that it does this for every csv file corresponding to every pair of the coordinates.
* Even though there is a single model, it is re defined every loop. This ensures each dataset trains on a fresh model without previous weights. Declaring the model inside the loop ensures that a new instance of the model is created for each dataset. This approach is crucial when dealing with multiple datasets to avoid any data leakage or influence from previously trained models (clean slate training). It helps maintain the integrity of the learning process for each distinct dataset.
* Pretty much have a loop to extract the 3 days for the input per pair of coordinates and then the same thing but for the output 1 day for comparison.
* Important to mention that these have nans, so I had to interpolate since we need 72 time periods for the input to make a prediction and 24 to make a comparison… but it must be said that this means that the values are not that accurate.
* Mention why 70-15-15 split. Get reference.
* The final implementation does not use interpolation to fill in the nans, instead we just dropped the nans (find reference).
* The code was updated to use a more computationally expensive model architecture to allow for better predicted results. **Ignore what I said before about computational time!**
* **IMP ->** For Vis of red and blue create new code for shapefile visualisations instead of cartopy… to use in write up!
* Important to mention that nans were dropped, and the T-V-T Splits had the nans also dropped.

**AI Model Evaluation**

* Compare predictions (24 hours) to actual data of the same date of 24 hours.
* Can compare using evaluation metrics.
* Create two Lagrangian simulations for the actual 24 hours and for the predicted 24 hours and then have 2 visualisations on a single plot.
* To evaluate the accuracy of our models' predictions, we encountered a significant challenge. Our initial approach involved using data from August 1st to August 3th, 2023, as a basis for predicting values for August 4th 2023. However, the presence of Nans in the dataset presented complications, as these gaps in the data made it difficult to generate hourly predictions for the subsequent day. To address this issue, we adopted a strategy that involved preprocessing the data, specifically by applying spline interpolation to missing values to ensure a complete 72-hour dataset for the day preceding the prediction. This approach allowed us to proceed with making predictions, albeit with the caveat that the interpolated data may not perfectly reflect reality, potentially affecting the accuracy of our predictions. Consequently, this introduces a degree of uncertainty regarding the reliability of these predictions, suggesting that our evaluations might not be very accurate.
* Mention that the 3 days that the prediction was used is the last day of the test set and the next two dates. This was done so that the model will be more accurate since it uses close dates from the training even though it is unseen will give better predictions… Find a reference about this.
* The best thing to do is take an average of the error metrics for all the models. I can compare the whole combined predictions to the whole combined actual values, but this can be problematic since there is so much missing data.
* Evaluation can be done visually by using the Lagrangian Evaluation notebook to see a video simulation of the actual values and the predicted values overlapped.
* Evaluation:

A screenshot of a chat

Description automatically generated

*A white rectangular frame with black text

Description automatically generated*

* Link:

*https://pdf.sciencedirectassets.com/271506/1-s2.0-S0957417423X00311/1-s2.0-S095741742302835X/main.pdf?X-Amz-Security-Token=IQoJb3JpZ2luX2VjEMn%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FwEaCXVzLWVhc3QtMSJGMEQCICmkjh4y33TX3MKi0Q6Ibs2T9EBAcXgLeEGEmNn2flAwAiBLnrgtLSurB70S0rS97p54PMZojDfYCQGfu2xDelvRXyq8BQjR%2F%2F%2F%2F%2F%2F%2F%2F%2F%2F8BEAUaDDA1OTAwMzU0Njg2NSIM%2BCqSs92vR7G1%2FzleKpAF6DU2EsMPZ%2BNHU8SWbcrp28SpZjhk%2BXb%2FxqnwwN9fiFERh%2Bhl86GXxDQIcT4owffABCRXJ8f5pMMrtzaXLxQmT8LKRtkglyfPvvlrdyjXnbYfPZksyXYSAaphpQ6peQHsydZ7ILiAKb7agOCc3Lo%2BE%2FcljPF3s2OKWc7lOa%2BVj4IPKWsmUO7k7%2BOQM0yb3I7dBjv7k3Jbx%2FkcM42%2F4XVcEItTH6mUDZqEdz5HIpUS6Sv3iyrxwRMUyCgTG3T2blQpOoiisOrOl4H3KIhtdjT93JxpqglEDs3vvj8gF45dkxcLssipY7mqUDhz6Hj8MNVNRlejSyCaMj6fXZBA5M%2FSunfFXPLtaw4HIOVG%2BphW8wT4HAKiN2nLnuuChC7xEPSIN6XSlHhie1BjfEOAh3iB8F56TKySwNj8ytL9U3l5zmLDrsGx3qDQ8L1zxHcqzMfaK6epGcxZb%2FKt1K5uv%2F8z0YMywDk0WNQqy2XbTpPqngBhJ9oFxaJN%2FOyRYGCz1mSkTyf7s6wav99PWoGCqutOH%2BNG2gCSG22cQEAMHldk2EJOTmFvey7ct6x2xdispWl24V0DpKOAUajyK02OJdOlI55BayYeTUR7UCsRAjj1qpvHHCoeZ2VG2Rk%2BJkBbFmJfJ1ryhvplmnlpz6jk9MMN6g%2BMf1VU8kthpbCCnwOmSaK3XaTNGqW4jFXmqDOVDC87lN9Bw3uNRxOZfIun0MEx5xRlOYGfCWZUFSmgo0pTayUMZo%2FCq6gNYIrXWpWc8Gge0igQux6ETOkWDkb2%2FXvldFWMeCihEfiUvvPQAtF9N0rl9evhCwM4XYsKBgREG9oCAxbGSRU9BwX5oiZmVhfKL6K9yjLY9Bmkz2ZKgiP50E4wsuvfrwY6sgE%2FLQNCzYFQ3ROEjVcCCvv4eydmxqVYqKSl4DTSIQ9A7K2TPwrpxvytLLRCnQCTT3TnXewhHz7BUfiyrVwnR3BlUt9eI1qjzhsltM3ln847jhcNgsiDPaA%2F14p%2FLdk%2BqK1z4IC8AjMNJCGr17L2sP4Qj3tnHIyiqtRKgjivUYWEKFKD6sv1fTwVS9eTDWkYsAwE%2Bb7%2BQwvB%2FG0Vtq8aczNP0eylcMq3tGHzEDmpJTWl0WC4&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20240318T092052Z&X-Amz-SignedHeaders=host&X-Amz-Expires=300&X-Amz-Credential=ASIAQ3PHCVTY4JDEOBPI%2F20240318%2Fus-east-1%2Fs3%2Faws4\_request&X-Amz-Signature=4ca1c4bfa715221ef8cfb966ce1e7f4129ad756973502734338a74d683485f7a&hash=7a99fecf61bb1c75b2d8bdad0342a0edf0be07a83127f452e5bd9a941b721c54&host=68042c943591013ac2b2430a89b270f6af2c76d8dfd086a07176afe7c76c2c61&pii=S095741742302835X&tid=spdf-7448cde1-2e68-4d14-86b4-228f2342caea&sid=e8c849b99b905645e379f9f7f4d461c53bd8gxrqb&type=client&tsoh=d3d3LnNjaWVuY2VkaXJlY3QuY29t&ua=021157515756515551&rr=86642637894411aa&cc=mt* -> Link where similar (MAE, RMSE) evaluation is done. Plus, GRU vs LSTM.

* **Evaluation IMP** ‘v’ is north to south, and radars are aligned north to south. ‘v’ is parallel to the radars, this is why ‘v’ is less accurate.
* We decided not to interpolate the values since temporal resolution and continuity are crucial. Given the potential for misrepresenting temporal trends and dynamics — especially when large gaps or non-random distributions of missing data are present — avoiding interpolation ensures that the analysis remains grounded in the actual observed data, avoiding the introduction of artificial trends or errors.

**Evaluation (In General)**

* **Lagrangian -> AI Model -> Lagrangian Model** 😉
* This was chosen since we can mention: since from first step we learned how things worked and this gave motivation for research. Also, can be mentioned in Methodology since from the Lagrangian we learned how things work and what exactly we needed.
* We have LSTM vs GRU for comparison evaluation.
* Mention how we started with running tests on one point and then we evaluated which model architecture, features and how many targets were best… show results from end of this document. Then say when these hyperparameters and architectures were chosen for both LSTM and GRU we ran the full implementation and then will compare at the end.
* Important to remember that objectives have number of subtitles.
* Finally, to compare GRU vs LSTM use the mean and standard deviation of the results!
* Optionally, can run to predict another date other than the 4th of August.
* Drifter data should only be used for the Lagrangian. This is most ideal although can be risky since I am only using currents for the simulation and drifters would be affected by other phenomena (a bit, they are mostly affected with currents).
* Mention and show comparison Lagrangian evaluation notebook (overlapping)
* Test set visualisations can also be used but double check they are the same…
* Can mention/do that the predicted results are better and more realistic than the actual data for the simulation, since the actual data has a lot of missing data (nans) and is hard to interpolate. This needs to be confirmed with trajectories!
* One thing we can do is that we make a Lagrangian simulation of the data with the same location and dates as the drifters (does not have to be 2023). Then if it matches we can say that the Lagrangian simulation is accurate (due to the previous experiment), so we can assume that the simulation for the 4th of august (Predicted vs Actual) is correct and just evaluate on the AI models.

**Future Work**

* Incorporate wind into the simulation and the prediction to make the model more accurate in real life scenarios.
* Jellyfish/plankton movement, search and rescue, oil spills.
* Can move onto ensemble learning (elaborate).
* Individual models with shared learning, train separate models for individual (lat & lon), then leverage insights from clustering for better models. Can also experiment with clustering common data together and making predictions per cluster. (Elaborate).
* Can make area bigger and have more than the 37 total models. Also, can have a more complex model like a transformer.
* Get more weather data that will influence surface debris movement and create a method with more features for a more realistic and accurate prediction! These can also be predicted as new targets and the Lagrangian to be updated to use this new data.
* Model that predicts the missing values in the datasets.
* Have a system where it is constantly being trained with most recent data.
* Fill in the missing values in the datasets
* Can be used as a better visualisation overall since there is no nans
* Website with future predictions and better visualisations

**Miscellaneous**

* Put Adam in Acknowledgments.
* REDP Application ID: ICT-2023-00203
* Mention Rubber Duck Incident.
* Tidal height was ignored (Reference – Godin 1983)
* Nan values where ignored (elaborate)
* Mention that there exists ARIMA for example for same situation, this will emphasize how different and why we took a different approach.
* Mention that… Originally the idea was to create a webpage implementation to showcase the Lagrangian model and the prediction. After some consideration it was determined that it is better to focus on the physics and the academic component since the web page was more software development and therefore the idea was dropped.
* Important to make pipeline pseudocode!

**Models Evaluation**

* Mention **outlier**, which performed better, how ‘u’ is better and then an explanation.

**LSTM U**

Metric Mean Standard Deviation

MAE 0.14125946248567997 0.2265831724081127

MSE 0.11692609396936171 0.5132824073724478

RMSE 0.1795733102624883 0.29099745739565114

**LSTM V**

Metric Mean Standard Deviation

MAE 0.14369923815926253 0.13416186874255337

MSE 0.06405462684653777 0.10900010672303224

RMSE 0.18299755039882487 0.17483284415282882

**GRU U**

Metric Mean Standard Deviation

MAE 0.14848978520755404 0.22170805613768949

MSE 0.11588880936806335 0.5033877277710723

RMSE 0.18693390462543497 0.2845075125010799

**GRU V**

Metric Mean Standard Deviation

MAE 0.14472329790354027 0.1376257720831358

MSE 0.06558009878579078 0.11214920062969254

RMSE 0.18376353052668595 0.17835656319339224

**Advanced Pseudocode**

1. Main Data Frame
2. Remove Nans from Main Data Frame
3. *Save main Data Frame top physical Storage (EXTRA)*
4. Geospatial Visualization of all the points
5. Polygon Geospatial Visualization
6. Save to Final Data Frame (Points in the polygon)
7. Save Final Data Frame to physical Storage

LOOP

1. Loop through Final Data Frame and for every pair of coordinates, save individual Data Frame and put all the Data Frames into a folder

LOOP

1. Loop through the folder and create sequences
2. Split every sequence into Train/Val/Test sets
3. Train on same model (Every Individual Data Frame)
4. Store the result of every model in physical storage in a folder

LOOP

1. Loop through all the coordinates and create input (3 days) and the actual output (1 day) and store all on physical storage.

LOOP

1. Loop through input 3 day Data Frame and make a prediction for every single corresponding pair of coordinates for a total of 24 hours
2. Evaluate every prediction
3. Merge predicted values into a single file
4. Save file to physical storage
5. **Repeat for ‘v’**
6. *Evaluate the whole file [EXTRA]*
7. Merge ‘u’ and ‘v’ predictions
8. Simulate Lagrangian with original data (same 24 hours)
9. Simulate Lagrangian with predicted data
10. Compare visualizations.

**Latest Results**

model = Sequential([ #35+ mins

    LSTM(256, input\_shape=(win\_length, num\_features), return\_sequences=True),

    Dropout(0.3),

    LSTM(128, return\_sequences=True),

    Dropout(0.3),

    LSTM(128, return\_sequences=True),

    Dropout(0.3),

    LSTM(64),

    Dense(64, activation='relu'),

    Dropout(0.3),

    Dense(32, activation='relu'),

    Dropout(0.2),

    Dense(1)

])

Mean Absolute Error (MAE): 0.04815140675522497

Mean Squared Error (MSE): 0.004785946268366612

Root Mean Squared Error (RMSE): 0.06918053388321466

# Define the LSTM model

model = Sequential([ # 8 mins

    LSTM(128, input\_shape=(win\_length, num\_features), return\_sequences=True),

    LSTM(128, return\_sequences=True),

    Dropout(0.3),

    LSTM(64),

    Dropout(0.3),

    Dense(1, activation='leaky\_relu')

])

Mean Absolute Error (MAE): 0.06943381787629392

Mean Squared Error (MSE): 0.006316376392918817

Root Mean Squared Error (RMSE): 0.07947563395732

model = Sequential([ #15 mins

    LSTM(256, input\_shape=(win\_length, num\_features), return\_sequences=True),

    Dropout(0.3),

    LSTM(128, return\_sequences=True),

    Dropout(0.3),

    LSTM(64),

    Dropout(0.2),

    Dense(1)

])

Mean Absolute Error (MAE): 0.058881258369162755

Mean Squared Error (MSE): 0.004260453256319127

Root Mean Squared Error (RMSE): 0.06527214763066347

model = Sequential([ # 33 mins

    LSTM(256, input\_shape=(win\_length, num\_features), return\_sequences=True),

    Dropout(0.3),

    LSTM(128, return\_sequences=True),

    Dropout(0.3),

    LSTM(128, return\_sequences=True),

    Dropout(0.3),

    LSTM(64),

    Dense(64, activation='relu'),

    Dropout(0.2),

    Dense(1)

])

Mean Absolute Error (MAE): 0.05014801504637721

Mean Squared Error (MSE): 0.004105850056959454

Root Mean Squared Error (RMSE): 0.0640769073610724

    model = Sequential([ # 20 mins

    GRU(256, input\_shape=(win\_length, num\_features), return\_sequences=True),

    Dropout(0.3),

    GRU(128, return\_sequences=True),

    Dropout(0.3),

    GRU(128, return\_sequences=True),

    Dropout(0.3),

    GRU(64),

    Dense(64, activation='relu'),

    Dropout(0.2),

    Dense(1)

])

Mean Absolute Error (MAE): 0.05095688652320586

Mean Squared Error (MSE): 0.005465899094394342

Root Mean Squared Error (RMSE): 0.07393171913593205

**FINAL LSTM MODEL**

# 8 Batch size:  
model = Sequential([ # 1hr

    LSTM(256, input\_shape=(win\_length, num\_features), return\_sequences=True),

    Dropout(0.3),

    LSTM(128, return\_sequences=True),

    Dropout(0.3),

    LSTM(64),

    Dropout(0.2),

    Dense(1)

])

Mean Absolute Error (MAE): 0.05117638073347091

Mean Squared Error (MSE): 0.005773326056766139

Root Mean Squared Error (RMSE): 0.07598240623174643

# 32 Batch size:  
model = Sequential([ # 13 mins

    LSTM(256, input\_shape=(win\_length, num\_features), return\_sequences=True),

    Dropout(0.3),

    LSTM(128, return\_sequences=True),

    Dropout(0.3),

    LSTM(64),

    Dropout(0.2),

    Dense(1)

])

Mean Absolute Error (MAE): 0.058487600016612636

Mean Squared Error (MSE): 0.004140952992224818

Root Mean Squared Error (RMSE): 0.0643502369243876

# 64 batch size: **CHOSEN MODEL**

model = Sequential([ # 20 mins

    LSTM(256, input\_shape=(win\_length, num\_features), return\_sequences=True),

    Dropout(0.3),

    LSTM(128, return\_sequences=True),

    Dropout(0.3),

    LSTM(64),

    Dropout(0.2),

    Dense(1)

])

Mean Absolute Error (MAE): 0.050986669725115216

Mean Squared Error (MSE): 0.003993544460290849

Root Mean Squared Error (RMSE): 0.06319449707285317

# 128 batch size:  
model = Sequential([ # 17 mins

    LSTM(256, input\_shape=(win\_length, num\_features), return\_sequences=True),

    Dropout(0.3),

    LSTM(128, return\_sequences=True),

    Dropout(0.3),

    LSTM(64),

    Dropout(0.2),

    Dense(1)

])

Mean Absolute Error (MAE): 0.05668386128265458

Mean Squared Error (MSE): 0.00645752369881714

Root Mean Squared Error (RMSE): 0.08035871887242317

# Batch size: 64 Learning Rate: 0.0005 Patiance: 3

model = Sequential([ #30 mins

    LSTM(256, input\_shape=(win\_length, num\_features), return\_sequences=True),

    Dropout(0.3),

    LSTM(128, return\_sequences=True),

    Dropout(0.3),

    LSTM(64),

    Dropout(0.2),

    Dense(1)

])

Mean Absolute Error (MAE): 0.05490543067000056

Mean Squared Error (MSE): 0.00562192764358487

Root Mean Squared Error (RMSE): 0.07497951482628352

**Results (OLD)**

LSTM Single Target U (2 years):

|  |  |
| --- | --- |
| Test MSE | 0.0024190667630645227 |
| Test R-squared | 0.8156669780585895 |
| Test Explained Variance | 0.816099923155643 |
| Test MAE | 0.021644903820475958 |

LSTM Single Target V (2 years):

|  |  |
| --- | --- |
| Test MSE | 0.0026758533046433593 |
| Test MAE | 0.032873101353487616 |
| Test R-squared | 0.796099829302783 |
| Test Explained Variance | 0.8053555550911925 |

LSTM Single Target U Simple (3 years):

|  |  |
| --- | --- |
| Test MSE for 'u' component | 0.0018392425899589357 |
| Test MAE for 'u' component | 0.028609757062382415 |
| Test R-squared for 'u' component | 0.6981919036542692 |
| Test Explained Variance for 'u' component | 0.6981958637429437 |

LSTM Single Target V (3 years):

|  |  |
| --- | --- |
| Test MSE for 'v' component | 0.0034144461552393186 |
| Test MAE for 'v' component | 0.0397711281401556 |
| Test R-squared for 'v' component | 0.7681236805037378 |
| Test Explained Variance for 'v' component | 0.7688760337222249 |

LSTM Single Target U (24 Sequence):

|  |  |
| --- | --- |
| Test MSE for 'u' component | 0.0036544685574883087 |
| Test MAE for 'u' component | 0.04077346492489306 |
| Test R-squared for 'u' component | 0.7214374535512165 |
| Test Explained Variance for 'u' component | 0.7528849249465257 |

LSTM Single Target U (12 Sequence):

|  |  |
| --- | --- |
| Test MSE for 'u' component | 0.0029143808739715525 |
| Test MAE for 'u' component | 0.034527283585515066 |
| Test R-squared for 'u' component | 0.7777920622305681 |
| Test Explained Variance for 'u' component | 0.7784308998641449 |

LSTM Single Target U (1 Sequence):

|  |  |
| --- | --- |
| Test MSE for 'u' component | 0.0026107904748576068 |
| Test MAE for 'u' component | 0.03225322956613726 |
| Test R-squared for 'u' component | 0.8011538230713655 |
| Test Explained Variance for 'u' component | 0.8101298180496459 |

LSTM Single Target U (0.001 LR):

|  |  |
| --- | --- |
| Test MSE for 'u' component | 0.002342229529907343 |
| Test MAE for 'u' component | 0.029819278266926652 |
| Test R-squared for 'u' component | 0.8216082860740367 |
| Test Explained Variance for 'u' component | 0.8216656680894614 |

LSTM Single Target U (with ‘v’ as feature):

|  |  |
| --- | --- |
| Test MSE for 'u' component | 0.002516725745286363 |
| Test MAE for 'u' component | 0.03264639806964096 |
| Test R-squared for 'u' component | 0.808318094597248 |
| Test Explained Variance for 'u' component | 0.82340737786933 |

**From here on always 2 features and a single target**

LSTM Single Target U (with adaptive LR starting at 0.001):

|  |  |
| --- | --- |
| Test MSE for 'u' component | 0.0023592781916186665 |
| Test MAE for 'u' component | 0.03052777880046811 |
| Test R-squared for 'u' component | 0.8203098053128676 |
| Test Explained Variance for 'u' component | 0.8203161559186333 |

LSTM Single Target U complex (3 years):

|  |  |
| --- | --- |
| Test MSE for 'u' component | 0.001898580755587542 |
| Test MAE for 'u' component | 0.029220607353491038 |
| Test R-squared for 'u' component | 0.6886662720026921 |
| Test Explained Variance for 'u' component | 0.6945530437051486 |

GRU Single Target U (3 years, LR=0.0001):

|  |  |
| --- | --- |
| Test MSE for 'u' component | 0.0017845362995307333 |
| Test MAE for 'u' component | 0.02757052929739459 |
| Test R-squared for 'u' component | 0.7073675495528291 |
| Test Explained Variance for 'u' component | 0.7073919490494525 |

GRU Single Target U (2 years, LR=0.0001):

|  |  |
| --- | --- |
| Test MSE for 'u' component | 0.0024379899291823042 |
| Test MAE for 'u' component | 0.03051282151001504 |
| Test R-squared for 'u' component | 0.8143148669044942 |
| Test Explained Variance for 'u' component | 0.8168623228711309 |

GRU Single Target U Less complex(2 years, LR=0.001):

|  |  |
| --- | --- |
| Test MSE for 'u' component | 0.0022538887932932 |
| Test MAE for 'u' component | 0.02898659598288193 |
| Test R-squared for 'u' component | 0.8283365999360444 |
| Test Explained Variance for 'u' component | 0.8283392773560619 |

LSTM Single Target U Less complex (Interpolation, single feature):

|  |  |
| --- | --- |
| Test MSE for 'u' component | 0.007416813195800896 |
| Test MAE for 'u' component | 0.04529537497181993 |
| Test R-squared for 'u' component | 0.519584878029119 |
| Test Explained Variance for 'u' component | 0.5203720938677607 |

LSTM Single Target U Higher dropout 0.3 (Adding onto best model):

|  |  |
| --- | --- |
| Test MSE for 'u' component | 0.0024214458955741304 |
| Test MAE for 'u' component | 0.03134766232455302 |
| Test R-squared for 'u' component | 0.8155749135706839 |
| Test Explained Variance for 'u' component | 0.8209414886247651 |

Batch size = 8

|  |  |
| --- | --- |
| Test MSE for 'u' component | 0.002179373371083336 |
| Test MAE for 'u' component | 0.02803178147733633 |
| Test R-squared for 'u' component | 0.8340119335069861 |
| Test Explained Variance for 'u' component | 0.834027284219115 |

Batch size = 16

|  |  |
| --- | --- |
| Test MSE for 'u' component | 0.0023517817452135712 |
| Test MAE for 'u' component | 0.03077673077260262 |
| Test R-squared for 'u' component | 0.820880758716658 |
| Test Explained Variance for 'u' component | 0.8208942235949539 |

Batch size = 32

|  |  |
| --- | --- |
| Test MSE for 'u' component | 0.002271727808319735 |
| Test MAE for 'u' component | 0.028695319344842898 |
| Test R-squared for 'u' component | 0.8269779233312538 |
| Test Explained Variance for 'u' component | 0.8291700086264638 |

Batch size = 64

Very similar to normal

Batch size = 128

Very similar to normal

Batch size = 256

Very similar to normal

LSTM Double Target U,V (2 years):

|  |  |
| --- | --- |
| Test MSE for 'u' component | 0.004009208428052819 |
| Test MSE for 'v' component | 0.008702654008220605 |
| Test MAE for 'u' component | 0.04196935631070439 |
| Test MAE for 'v' component | 0.0579934010020833 |
| Test R-squared for 'u' component | 0.6944980947116437 |
| Test R-squared for 'v' component | 0.40923099614634095 |
| Test Explained Variance for 'u' component | 0.722836121476568 |
| Test Explained Variance for 'v' component | 0.43345408386505246 |

LSTM Double Target U,V (3 years):

|  |  |
| --- | --- |
| Test MSE for 'u' component | 0.0022882551865674927 |
| Test MSE for 'v' component | 0.005073492373509212 |
| Test MAE for 'u' component | 0.03262867246795131 |
| Test MAE for 'v' component | 0.05051134610980628 |
| Test R-squared for 'u' component | 0.6245117715403712 |
| Test R-squared for 'v' component | 0.655457229349919 |
| Test Explained Variance for 'u' component | 0.6271222424285632 |
| Test Explained Variance for 'v' component | 0.6628467139706005 |