UAV Classification with Mamba

Final Project Report, MLDS, Reichman University
Ayala Raanan, Jun 2024

1. Overview

The increasing use of unmanned aerial vehicles (UAVs) on the battlefield is forcing the world to develop diverse solutions for protection against such attacks. At the heart of any defense system is the need for a fast and reliable method for detecting and classifying flying objects. Common detection means include a combination of various optical cameras and radar technology, which is used to obtain the range and other properties of the targeted object. While detection and classification using images is one of the most researched topics in modern AI, this project takes a different approach. We leverage time-series data characterizing the object's trajectory to classify it. We use a Mamba generative model [1] to learn class behaviors and predict future trajectories as described in section 2. The classification results obtained by this method are compared to results by standard features-based baseline methods as described in section 6.

The data for this work is provided by Walaris, a company specializing in airspace awareness solutions. A stationary camera is used to track a flying object and classify it based on the image. A byproduct of the tracking is the direction of the object at each timestamp, given as a 3D unit vector pointing toward its location. The trajectory classification (as obtained from this project) can be weighted along with results from other methods for final decision-making. Moreover, in cases where the object's range is too great to enable sufficient resolution for image classification, trajectory classification may have a considerable advantage.

2. Proposed Methodology

Our task is to distinguish between the trajectories of four classes: UAV, bird, airplane, and static-object. We attempt to tackle it using the method of comparative forecasting models. For each class c of a flying object, we build a forecasting model M^c and train it to capture the physical and statistical behavior of that class. Given time-series data from the beginning of a trajectory $\{X^c_{t_i}\}_{i=0}^{l-1}$, we train the class model to predict the future trajectory $X^c_{t_l}$ (Eq.1). This is a **regression** task. With every new measurement at time t_l we can compare the incoming data to the model prediction and measure the error. The model is trained by minimizing the errors averaged along the trajectory. Eq.2 shows the class regression loss. Note that the new true data is fed into the model for future predictions in a streaming fashion. The trajectory X may be represented by a D dimensional vector, in which case we minimize the average error over the squared loss of all coordinates (with possible weights).

$$\tilde{X}^c_{t_l} = M^c(\{X^c_{t_i}\}_{i=0}^{l-1}) \tag{1}$$

$$\text{Loss} = \frac{1}{L}\sum_{l=1}^{L}(\tilde{X}^c_{t_l} - X^c_{t_l})^2 \tag{2}$$

Once we have a forecasting model for each class, we can use these models for **classification**. Given a trajectory of an unknown class c, each model $M^{\grave{c}}$ infers the future trajectory assuming the object belongs to its class. We predict the true class by comparing the errors between the observed trajectory and the predicted trajectory. We expect the true class model to produce the prediction with the least error (Eq.3). For example, when fed with the trajectory of a bird $\{X^{\text{bird}}_{t_i}\}_{i=0}^{l-1}$, we expect the bird class model to produce predictions with the least error compared to the other class models. Eq.3 shows classification based on comparing the prediction for a single data point in time. We can however average the errors over a period of time and gain more confidence in the prediction. The object is classified to the class with the least average error.

$$C_{\text{Predicted}} = \underset{\grave{c}}{\text{argmin}}(M^{\grave{c}}(\{X^c_{t_i}\}_{i=0}^{l-1}) - X^c_{t_l})^2 \tag{3}$$

2.1 Time Considerations For the application of defense against UAVs it is clear that an early-as-possible classification is essential. Throughout this work, the evaluation is carried out with this in mind, to be able to assess the accuracy at different intervals after first encounter. We expect the results to reflect the fact that evaluating a longer trajectory with more information should improve the classification results. For each method, we show the results for intervals of 5, 10, 15, 20, 25, and 30 seconds after first encounter.

2.2 Using Mamba Any prediction can be used with this methodology. We could employ either RNNs (particularly LSTM) or Transformer models. In the months preceding this work, the Mamba model was introduced as a promising new

alternative to state-of-the-art Transformers. This task provides a good opportunity to test its performance. Mamba may be particularly well-suited for this dataset because it originates from state space models (SSM), which are effective at learning the internal dynamics of a physical system, as detailed in section 7.1.

2.3 Method Pros and Cons This methodology is somewhat costly because we need to train a model for each class. However, it has several advantages. For instance, we can improve and retrain a class model separately when new data is collected for that class. With each newly trained forecasting model, we may find that combining it with the other class models yields better classification results.

The sequential inference aspect of this method is particularly intuitive for the UAV detection use case. We want the model to process new data points in real-time and improve the classification results. Choosing a model with linear compute, such as Mamba, helps keep the latency of the prediction low, which benefits the application. The specific advantages of Mamba are details in 7.7.

3. The Walaris Dataset

The dataset was provided by Walaris as a collection of time-series data from their tracking and detection product. It contains JSON files logging trajectories belonging to one of four classes: UAV, airplane, bird, and static-object. Each file records several variables by timestamp at an average of 25 frames per second (timestamp every 40 milliseconds). The variables include the components of the 3D unit vector pointing in the direction of the object, the estimated velocity, angular size, bounding box, camera field of view (FOV), pan/tilt parameters, and light domain.

The first data processing step was to convert the 3D unit vectors into the corresponding 2D vectors of *independent* azimuth and elevation (θ, ϕ). Note that the range to the object is not provided, which introduces a fundamental challenge to this work. In theory, a trajectory can look the same whether the object is flying fast and high or slow and low.

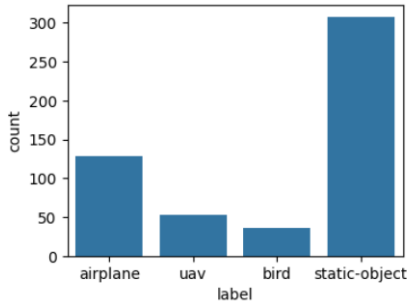

Figure 1: Number of files per class. The imbalance is compensated by sampling files of different classes with a custom overlap.

From exploratory data analysis (EDA), we learn a lot about the variables' spans as well as the biases in the dataset. For example, the span of the azimuth in a file can range from a hundredth of a degree to hundreds of degrees, presumably depending on the object's range and velocity. Characterizing the *local Std* for each variable indicates the scale of the variable in each file and helps us later on with normalization. This is done by calculating the Std over a rolling window in time and then taking the median of all values as a representative value. Fig. 2 shows the distribution of elevation local Std per class. Fig. 3 shows some typical azimuth and elevation projections (θ, ϕ) for each class.

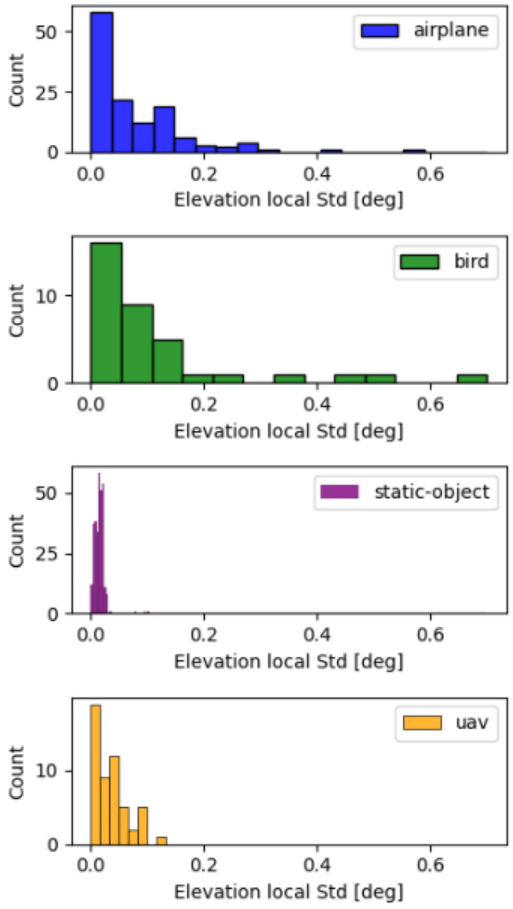

Figure 2: Elevation local Std distribution by class. The local Std is the median of the Std taken over a rolling window in time, indicating the local variations scale.

4. Preprocessing

Each data file contains a recording of arbitrary length. From each file, we extract samples of **known durations**, which are important for several reasons. Firstly, the samples we use for training must reflect the reasonable durations for which we require good performance from the model. Secondly, we want to evaluate performance for different sample durations and demonstrate that classification improves with longer sample durations, as expected.

For this project, we used samples of 5, 10, 15, 20, 25, and 30 seconds, with very few 60-second samples. The amount of overlap between the samples varies from class to class.

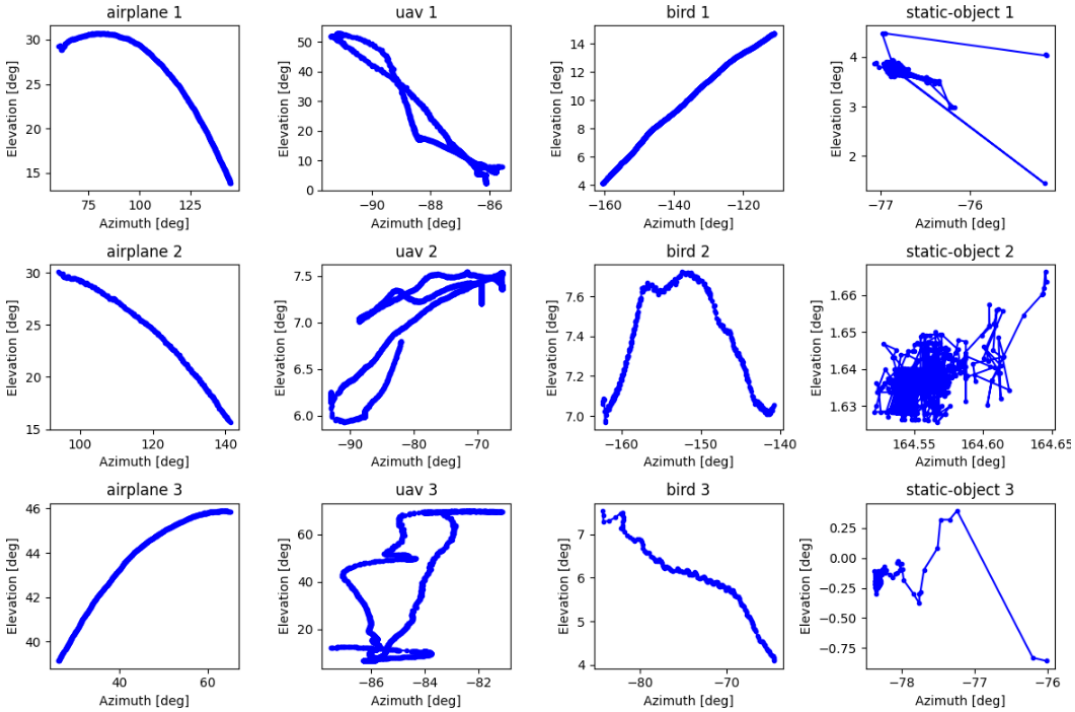

Figure 3: Typical azimuth and elevation trajectories

The overlap parameter was used to compensate for the imbalance of data per class. Files from classes with fewer data (e.g., bird) were sampled with much higher overlap than files from classes with many data points (e.g., static object). This produced a similar number of subsamples for each class in the training, validation, and test sets. Naturally, there are more short samples than long ones. Since we want to drive the model to an early detection, this inherent distribution should serve us well.

The raw samples consist of the timestamp, 3D direction vector, and the angular size field (both horizontal and vertical) which proved to be useful. The samples are preprocessed to match the input forms required by the models. We need two pipelines: one performing feature extraction for the use of the baseline feature-based methods, and one for the Mamba model. The first stages in both pipelines involve converting the 3D (x, y, z) unit vectors into 2D (θ, ϕ) vectors. The sample is then cleaned for outliers by median replacement and is offset to start from $(0, 0)$. Note that the test data was also cleaned in the same way. In implementation, this leads to an estimated latency of 80 milliseconds, but post analysis shows that this is worthwhile. The rest of the pipelines can be described as follows:

- Feature-based models: The 2D vectors undergo several transformations: derivation (velocity and acceleration), interpolation, smoothing, and PCA. We extract features from each transformation stage or their combination. These include extremum and median values, standard deviations, spans, ratios, correlations, and some special features detailed in section 6.1.
- Mamba model: The model needs three inputs from preprocessing for a sample of length L:
  1. The vector $\{\Delta t_i\}_{i=1}^{L}$ of time intervals between the samples (L)
  2. The D-dimensional trajectory representation $\{X_{t_i}\}_{i=1}^{L}$ $(L \times D)$
  3. The scale (local Std) of each dimension in the representation, used as weights during evaluation. (D)

For training Mamba, the samples were interpolated to intervals of the typical 40 milliseconds. Recall that we cut the samples to a known duration but with possibly different intervals and different lengths. Working in batches, However, requires samples of equal length. Interpolation solves this problem and greatly simplifies model training and evaluation. Note that the implementation still supports an input of arbitrary intervals for inference.

5. Dataset Splitting

The dataset was split into train and test sets with an 80:20 ratio. It is essential that the splitting is done according to files and not samples to avoid information leakage. Since one of the key challenges in the dataset is the varying scale, it was important for each set to preserve the distribution as best as possible and contain samples of both small and large scales. To that end, we used the local Std of the elevation as an indicator. The files in each class were **sorted** according to their elevation local Std, and then every fifth file was assigned to the test set. For the use of the Mamba model, the file preceding each fifth file in the sorting was assigned to the validation set.

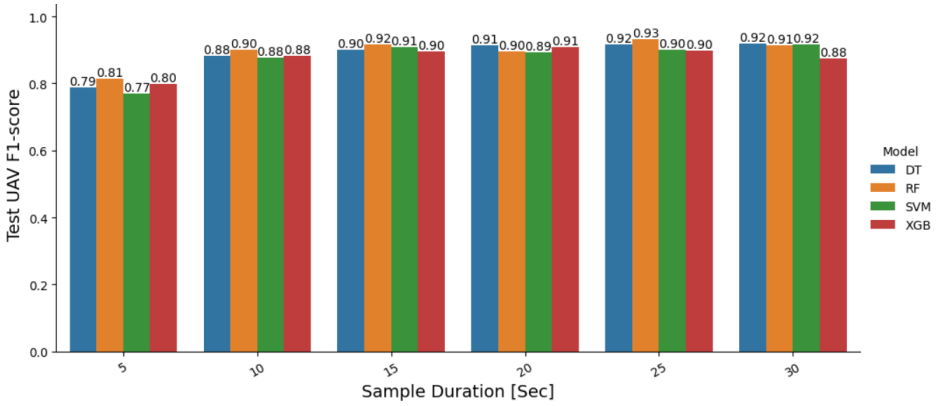

Figure 4: Baseline methods performance, per sample duration. The performance improves with time, especially between 5 to 15 seconds.

The baseline feature-based methods (RF, SVM, etc.) used 5-fold cross-validation during training and did not use a specific validation set.

6. Baseline Methods

In the first stage, the data was used to obtain results with standard feature-based classification methods (Decision Tree, Random Forest, XGBoost, SVM). Significant time was devoted to feature extraction and hyperparameter tuning.

6.1 Feature Extraction As described, we performed various transformations on the data series samples, including smoothing, interpolation, derivation, and more. From each transformation, we extracted appropriate features. The features were largely divided into two groups: scale-dependent and scale-independent. Scale-dependent features include everything that depends on the explicit values of the trajectory, such as maximal elevation, mean angular velocity, and azimuth Std. These features would change significantly if the same trajectory was observed from a greater or shorter distance. On the other hand, there is a group of scale-independent features that would not change, or would change very little, with distance. These include various ratios and correlations. We will expand on a few that were crafted specifically with the trajectory classification task in mind:

- Geometric curvature quantiles - The motivation for this feature comes from the observation that UAV trajectories often contain loops or highly curved paths. One way to quantify curvature is to calculate the ratio between the distance along the trajectory path and the Euclidean distance between two points on the path. Straight lines should have ratios close to 1, while loops should produce very high ratios as the Euclidean distance becomes small. To characterize the trajectory curvature, the ratio is calculated for all pairs of data points on the trajectory (excluding pairs below a minimal interval between them to filter out noise). After collecting the ratios from all pairs, we take the statistical 10-quantiles as features. We expect highly curved paths to have very high values at high quantiles (0.8, 0.9). For a smooth trajectory (e.g., an airplane) we expect the 0.5 quantile to have values slightly higher than 1. These features proved to be dominant in feature-based classification.
- Another feature is crafted to characterize the distribution of velocities with respect to the median velocity. This feature helps determine whether the velocity stays steady throughout the trajectory (as expected for airplanes) or varies significantly (as with birds and UAVs). The velocities in the trajectory are normalized by their mean value, after which we extract the distribution's 10-quantiles.
- In the data visualization during EDA, it was evident that the velocity profile of UAVs has some very distinctive patterns during ascent or descent. We model similar filters and use the correlation to the normalized speed to try and capture such events (appendix, Fig.12).

6.2 Models and Training The base models we used were: Decision Tree, Random Forest, and SVM from the scikit-learn library, and XGBoost from the XGBoost library. Training was done with 5-fold cross-validation, where the average classification metrics across all folds were recorded (precision, recall, F1-score). Internal tools for class balancing were used for all models.

For each model, a grid search of the main hyperparameters was conducted. We used the UAV F1-score to choose the best hyperparameters. After selecting the best hyperparameters, the model was trained on the entire training set and evaluated on the test set for final results.

Initially, a separate model was built for each sample duration instead of a unified model that could classify all durations. The rationale was that it might be more difficult for a model to fit both short and long samples. While conditioning the model on the duration has some advantages, there are two main disadvantages. The first is that each model has fewer samples to train on. The second is that such models are more difficult to manage in a product setting. A unified model can be used continuously over time. The results presented in Fig.4 are from the unified model trained on all sample lengths.

6.3 Results The performance of the feature based models was surprisingly good. For example, the F1-score for Random Forest ranges from 0.81 for 5 seconds samples to 0.93 for 25 seconds samples. Further analysis used the RF model to compare the performance of range-dependent versus range-independent features, showing a significant advantage for the latter (appendix, Fig.13). The decision tree results were analyzed to identify the most useful features at the top of the tree. These were most often the geometric curvature quantiles, the normalized velocity quantiles, and the angular size median.

7. Mamba

Mamba is a new foundation model introduced by Albert Gu and Tri Dao [1]. It is part of recent efforts to develop subquadratic architectures that can compete in performance with the quadratic Transformer models and the attention mechanism. The Mamba model is built upon state space models (SSMs) and, more specifically, on structured state space models (S4).

SSMs are a popular way to represent the behavior of a linear system with multiple variables. Their implementation as neural networks involves a simple linear RNN with no activation layers. This linearity property makes them well-behaved during backpropagation and easy to train. Furthermore, their sequence-to-sequence operation can be executed as a single convolution, without the realization of high-dimensional hidden states.

S4 is an important improvement made by Albert Gu, Karan Goel, and Christopher Ré [2]. In their paper, the authors show how structuring the internal parameters of the model using HiPPO matrices [3] can maximize the compression of the sequence history into current hidden states, thereby boosting performance.

Although SSMs and S4 demonstrate very good results, they still underperform Transformers on many tasks, particularly in important discrete modalities such as language. Gu and Dao claim that the root cause of this underperformance is their lack of content-based reasoning. Once the model parameters are learned, the SSM step becomes independent of the input. The innovation of the Mamba model is making the model parameters a function of the input. While the SSM operations remain linear, this approach allows the model to propagate or forget information based on the current inputs, similar to LSTM gating.

Unfortunately, having the parameters depend on the input, limits the model execution to sequential operations and eliminates the efficient convolution option of the SSM. Although the operation of Transformers is quadratic in input length, they have the advantage of being parallelizable. Sequential operation cannot be parallelized, so a hardware-aware algorithm was introduced to make Mamba computationally efficient. The guiding principle is the clever use of GPU memory hierarchy. Intermediate calculations are sent to the fast SRAM when it is essential to materialize the high-dimensional hidden states of the model, and the results in a lower dimension are sent back to GPU high-bandwidth memory.

7.1 Mamba S4 Core Operation SSM models draw inspiration from classical state space models [4] but can be regarded simply as linear RNNs (or convolutions). We model the behavior of a continuous system through time using equations 4a and 4b, where the parameters A, B, and C map a 1-dimensional input $x(t)$ to an output $y(t)$ through a hidden state $h(t)$. While $x(t), y(t)$, and $h(t)$ are functions in equations (4), we can perform a discretization step (6) and transform the equations to 5a and 5b. The discretization step uses a time-like parameter Δ to approximate the system's state at an interval Δ later. Different approximation formulations will result in slightly different discretization steps. The resulting equations are interpretable as linear RNNs with four parameters ($\bar{A}, \bar{B}, C, \Delta$ Where $\bar{A}$ and $\bar{B}$ are computed from A, B, and Δ). This is the form of the general SSM. To make this a structured SSM, structure is imposed on matrix A, initializing it as a HiPPO-like matrix. For further simplification of the computation, A is constructed as diagonal.

$$(4a) \quad h^{'}(t) = Ah(t) + Bx(t) \qquad (5a) \quad h_t = \bar{A}h_{t+1} + \bar{B}x_t$$
$$(4b) \quad y(t) = Ch(t) \qquad (5b) \quad y_t = Ch_t$$

$$(6a) \quad \bar{A} = exp(\Delta A)$$
$$(6b) \quad \bar{B} = (\Delta A)^{-1}(exp(\Delta A) - I)\Delta B$$

7.2 Selective SSMs The equation system 4 - 6 is linear time invariant (LTI) with input-independent parameters. Mamba introduces input dependency to allow the network to respond differently to changing inputs. The dependence of parameters B, C and Δ is formulated as simple linear projections (Eq.7).

$$(7a) \quad B = Linear_N(x) \quad (7b) \quad C = Linear_N(x)$$
$$(7c) \quad \Delta = Broadcast_D(Linear_1(x))$$

$\bar{A}$ only depends on the input through Δ due to the discretization step. A itself is input-independent which is very important to note in our specific application (see discussion in section 7.6). Fig.14 in the appendix summarizes the changes in S4 which are required for selectivity. Selectivity means that the model can change its response according to the input. For example, it can choose to ignore the new input by damping $\bar{B}$ and making the hidden state the dominant term in the RNN step. Contrarily it can boost $\bar{B}$ and make the new input more dominant and forget the history of the sequence.

7.3 Selective SSM Block Architecture The selective SSM architecture is presented in Fig. 5 (taken from [1]). It shows the processing of an input x_t with $D = 5$, using a hidden state h of size $N = 4$, outputting y_t of $D = 5$.

7.4 The Mamba block The Mamba block (Fig.6, [1]), wrapping the SSM, is inspired by two other popular blocks: H3 [5] and gated MLP. Similarly to gated MLP, the input is first linearly projected to an expanded dimension ($\times 2$). This is done twice, where the secondary branch goes through an activation layer and is used for gating the SSM output. In the main branch, the expanded dimension enters a convolution layer followed by an activation layer before going into the

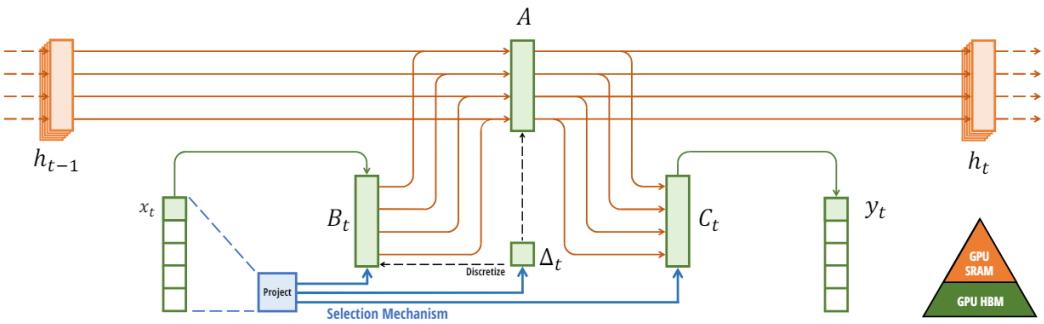

Figure 5: SSM Block from [1]

SSM. This process can be thought of as a way to distill the 'singular vectors' of the system. Once inside the SSM, each input dimension is processed independently of the others. At the end of the SSM, we have the output gate, after which a final linear projection produces the output.

7.5 Complete Architecture The Mamba block is wrapped into a residual block with an optional normalization block (LayerNorm) prior to the Mamba block. The complete Mamba architecture comprises stacked residual blocks with a Mamba core.

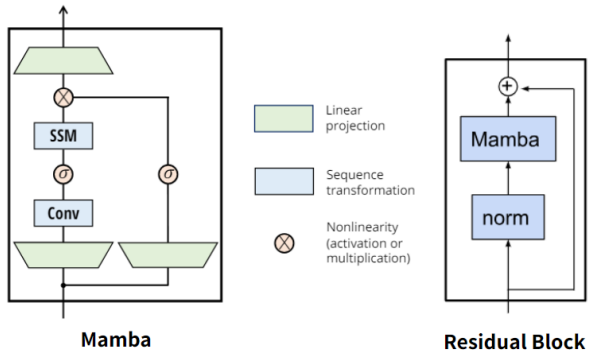

Figure 6: Complete Architecture from [1]

7.6 Implementation and time considerations The code for this project is based on an open-source Github repository [6][7]. While it retains all the core functionality of the Mamba model, it does not include the hardware-aware improvements that make it highly efficient for running on extremely large datasets. Significant changes were made to the code for this specific project, which will be detailed here.

Our trajectory data includes explicit timestamps for each data point, which is natural for a physical system and aligns with the original formulation of SSMs, where $\Delta = \Delta t$ is used for the discretization step. However, in discrete modalities such as language, there is no explicit Δ from token to token. For this reason, in the general case, Δ is parameterized or inferred from the input. Since we already have an accurate $\Delta = \Delta t$, it makes more sense to use it directly. Proper adjustments were made to the code to implement this, and Eq. 7c is not used in our case.

Reminded that $\bar{A}$ is input-dependent only through Δ, the result of using an explicit $\Delta = \Delta t$ is that $\bar{A}$ will not depend on the input x_t but only on Δt (which is a partial input). $\bar{A}$ will still change to emulate the time intervals between inputs, but it will not depend fully on the input.

In addition, since we have numeric inputs in our data and not tokens, the embedding layer in the code was removed. The chosen representation for our data points is of $D = 8$ and includes the azimuth and elevation, the derived angular velocities and accelerations, and the angular horizontal and vertical sizes. The sign of the velocities and accelerations was kept, as it is presumably more natural for a physical system with linear equations to handle than absolute values.

7.7 Mamba Advantages Mamba was able to surpass Transformer models performance on a variety of tasks including language, DNA and audio modeling [1]. It also has some advantages that make it especially attractive:

- Mamba memory and compute resources are linear in sequence length ($O(L)$).
- Implemented as a sequential model, inference for a new data point is done in $O(1)$
- Unlike Transformers, it is not limited to a context window size.
- The trained model is scalable in time and should be able to handle out-of-distribution timescales and missing timestamps. Since Mamba uses a time parameter Δ, a trained model can adjust its outputs according to the real-time intervals (whether they are inferred from the input or part of the input). This is a very useful property for physical applications.

8. Evaluation

8.1 Scale Considerations When training the forecasting models, we are confronted again with the problem of the missing range. The loss function will average over samples of very different scales. Since this is a regression problem, we expect the absolute values of the errors to be large when the object is close and small when it is far away. Nevertheless, we want the model to perform well for all ranges. To address this, we produced the local standard deviation (local Std) values for each sample (and each parameter) during

preprocessing. These values are used to normalize each error that is added to the loss function. Additionally, this normalization ensures equal importance to the different trajectory parameters, which have different scales (position, velocity, acceleration, and angular size).

Note that the local Std can be calculated during inference using the initial part of the trajectory before evaluation has started (sec. 8.2). In fact, the local Std is not needed during inference because of the lack of range; since we are dealing with a single instance, the scaling will be the same for all models. However, we should still use it during inference to weigh the different trajectory parameters appropriately.

8.2 Evaluation Options As discussed in Section 2, classification is done by comparing the regression errors of each forecasting model on the test samples. For better performance, we average the error over a period of time during which we compare the predictions to the actual trajectory. There are still different approaches we can choose and hyperparameters we can optimize while evaluating the regression error. Fig.7 shows the degrees of freedom that are available to us:

1. Initial Context Interval: We cannot expect the model to produce high-quality predictions without sufficient context from the beginning of the trajectory. Therefore, we may want to start evaluating the model outputs only after a minimal duration of the trajectory has been recorded. Starting too early might produce large errors that could outweigh the later, finer errors when averaged together.
2. Evaluation Period: This is the period of time over which we average the prediction errors. In a product setting, after the initial context duration has passed, we can evaluate new inputs and average over the period. The evaluation period may increase until we gain sufficient confidence in our classification, or until a time limit is reached.
3. The number of prediction steps k: So far, we have only discussed predicting one step ahead. Given the initial trajectory, the model predicts the most recent data point, for which we calculate the regression error. Another approach could be to let the model predict k steps ahead by constantly feeding the last model prediction as its next input. This evaluation is tricky since we must average the errors over the k steps, starting with **every** entry in the evaluation period. To do this, we must handle alternative predictions in parallel.

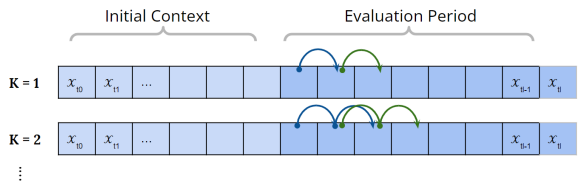

Figure 7: Evaluation degrees of freedom. We can choose the length of the initial context sequence and the number of propagated prediction steps k.

9 Training the Forecasting Models

With the proposed methodology, we must train a forecasting model for each class separately. Each class has its own training and validation sets. The loss function (Eq.2) is minimized for the training set, while evaluation on a validation set is performed to avoid overfitting. This evaluation was done using a constant evaluation period of 3 seconds at the end of all validation samples. We monitor the performance on samples of different duration in parallel. We chose to use early stopping when the average regression error on the 10-seconds samples stops decreasing. The parameters for the trained model with the smallest regression error was saved.

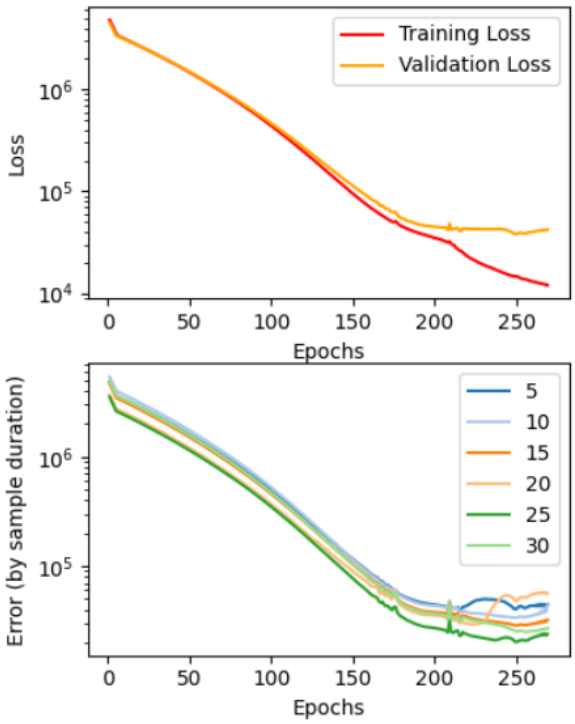

Figure 8: UAV forecasting model training. The saved model is of the epoch with best evaluation results on 10-seconds samples.

9.1 Hyperparameters Within each class, we train different models by changing the following hyperparameters: number of Mamba layers (1 or 2), hidden state dimension (16 or 32), batch size (16 or 32), and learning rate (0.1, 0.05, 0.01, 0.005). To speed up convergence, we often used a 0.1 learning rate for a warm-up period of up to ten epochs. The expansion factor and convolution size were left as recommended in the Mamba paper.

10 Combining Models for Classification

Each forecasting model produces different regression errors for the validation set. Next, we need to choose a forecasting model **representative** from each class and combine them to perform classification. But how do we choose the repre-

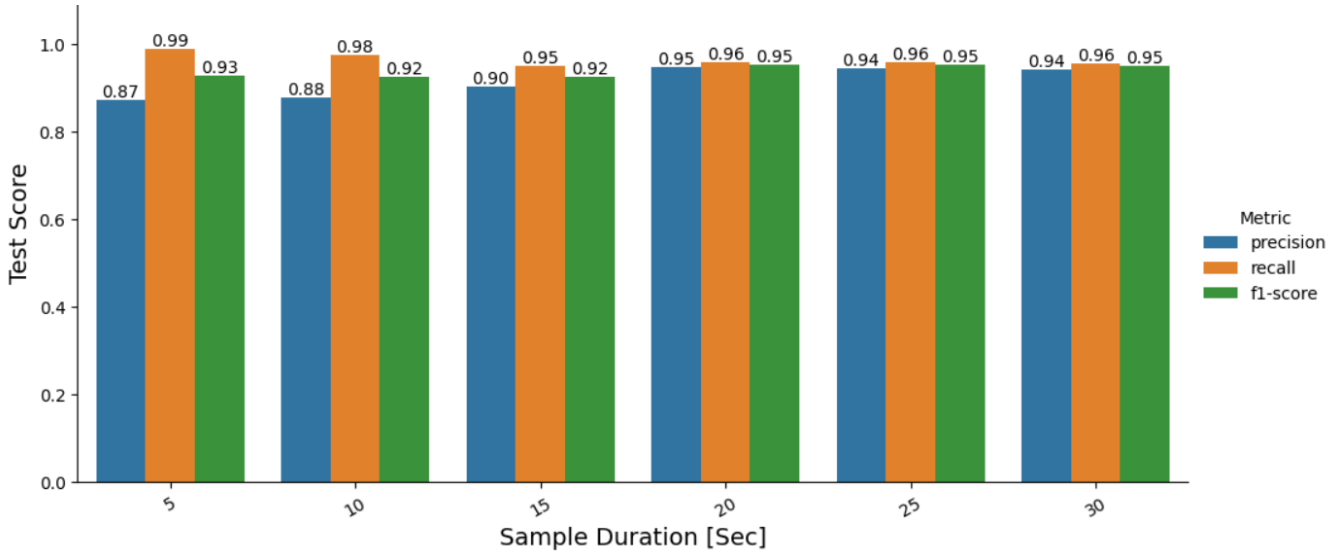

Figure 9: Mamba performance on the UAV class per sample duration. The combination of forecasting models for classification is optimized for best performance over the UAV class for short duration (5 sec), using the validation set.

sentative model for each class? We compared two plausible methods:

1. The naive approach would be to take the model which minimizes the regression error on the class's validation set. After all, such a model has probably captures the class behavior and should lead to good classification.
2. Alternatively, we can evaluate model combinations directly on the classification task and predict the **combination** with the best results. This can be done by re-using the validation set. Suppose we trained several possible forecasting models for each class. We can try each one in combination with other class forecasting models. We choose a metric by which we want to select the best combination (e.g., the UAV class f1-score) and choose the combination that maximizes this metric **on the validation set**. Finally, we can use the chosen combination to classify the test set and evaluate the results.

We found that method 2 produced much better results.

10 Results and Discussion

Fig.9 shows the test results for the best performing model. The model is the result of optimization on the validation set for the highest UAV class F1-score for samples of 5 seconds. With this model we get an F1-score of 0.93 for 5-sec samples and 0.95 for 20-sec and above samples. The optimization means that we use the freedom to choose the models' combination based on the best results for the 5-sec samples belonging to the validation set. We also optimize the **evaluation period** which is chosen to start at 0.5 sec for 5 sec samples. We can see similar results for a different combination optimized for 30-sec samples in the appendix (Fig.15). The UAV class F1-score reaches 0.96, but the results on the shorter samples are significantly lower (0.89 for 5-sec). When averaged over all sample durations, the results optimized for 5-sec do better.

While this combination does well on the UAV class (for which it was optimized), the other classes do not do as well. Fig.10 shows the confusion matrix for the 303 test samples of 5-sec duration. We find that the airplane class has particularly low recall (0.2) while the static-object precision is only 0.67. Looking back at the baseline methods, their result is generally more balanced. Table 1 shows the comparison for some examples.

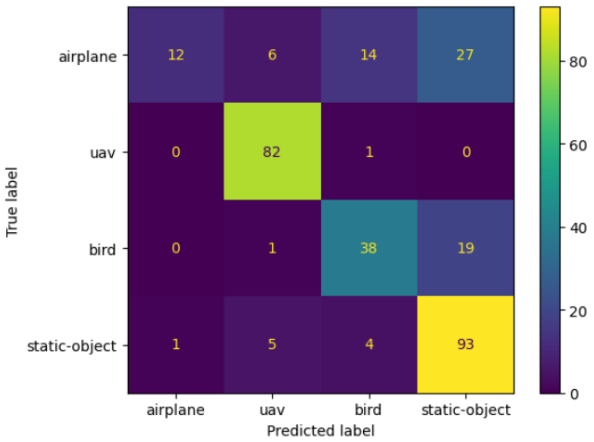

Figure 10: Confusion matrix for Mamba's test results

We hypothesis that this behavior is the result of the scale handling in each model. For example, notice that the static-object class naturally has a very small scale for all of its samples, which are in fact noise. It is likely to produce predictions with small changes. Other classes may have samples of smaller or larger scales. If the other classes were not able to properly generalize for samples of small scale, the static object might make better predictions (even by not pre-

Model	Class	Precision	Recall	F1-Score
Mamba	UAV	0.87	0.98	**0.93**
	Airplane	0.92	0.2	0.33
	Bird	0.67	0.66	0.66
	Static-Object	0.67	0.9	0.77
RF	UAV	0.89	0.75	0.81
	Airplane	0.73	0.9	**0.8**
	Bird	0.84	0.83	**0.84**
	Static-Object	0.95	0.95	**0.95**

Table 1: Comparing Mamba with Random Forest on **all** class-performance for L=5 sec samples. While Mamba was optimized for the UAV class and shows the best results, the chosen classification combination does not do as well as the baseline methods for other classes.

dicting any change). This can be the reason that so many of the airplane and bird samples were classified as static objects, hereby lowering their recall. If this is the case, the way to solve this would be either to train better models for the airplane and bird classes or handle the scaling differently.

10.1 k-steps prediction As explained in 8.2, we have the option to predict more than one step ahead by feeding the model with its own predictions instead of inputting the new available inputs. In order to do this we implemented a recurrent mode of the model (which can be run for inference in real-time), allowing the user to control the hidden states and the inputs. It was theorised that predicting k steps ahead, which also propagates the prediction error k steps ahead, can sometimes lead to a better prediction. For example, we might expect that when a UAV model is given an airplane trajectory, it would predict trajectory with a different curvature. We ran such an experiment where for each timestamp in the evaluation, a k steps prediction was produces and evaluated. We compared the results with predicting k = 1, 2, 3, 4, 5, 10 and 20 steps. The general trend we see is that when k increases, the precision increases, the recall decreases, and the F1-score decreases (Fig.11). For the shorter samples, the result with k = 1 was significantly better. Therefore, on average, it is not recommended to evaluate more than 1 step ahead.

10.2 Real-Time Δt As mentioned in section 4, the test samples were also lightly cleaned and interpolated. These operations are not hard to perform in real-time and cost only a small latency. However, one of the stated advantages of the Mamba model is that it is time adaptive. This means that even though the model was trained on interpolated data, it should be able to achieve good results for arbitrary time intervals (of similar size). To test this we processed the validation and test files again, this time without interpolation, which meant working with samples of various lengths and a batch size of 1. We ran the same evaluation process with the system's raw interval Δt vectors. We found that the results were degraded by about 2-3% with respect to the interpolated test samples. The F1-score for 5-sec samples was 0.87 and climbs up to 0.93 for 30-sec samples (Fig.16 in the appendix). The recall values for the UAV class are close to 1 for all sample lengths.

	Sample Duration [sec]					
Model	5	10	15	20	25	30
DT	0.79	0.88	0.9	0.91	0.92	0.92
RF	0.81	0.9	**0.92**	0.9	0.93	0.91
SVM	0.77	0.88	0.91	0.89	0.9	0.92
XGB	0.8	0.88	0.9	0.91	0.9	0.88
Mamba	**0.93**	**0.92**	**0.92**	**0.95**	**0.95**	**0.95**

Table 2: Mamba demonstrates the best results on the UAV class, with a 12% F1-score improvement on 5-second samples

11. Conclusions and suggested improvements

Table 2 compares the results of the different experiments. The feature-based baseline methods do very well with the extracted features. However, Mamba was able to outperform them, particularly on short intervals of 5 and 10 seconds. We adopt a method for using the validation set to optimize the combination of class forecasting models used for classification. We implement a recurrent mode that can perform inference in $O(1)$ and propagate predictions as input. We find that evaluation over a 1-step prediction performs better than looking k-steps ahead. We examine the effect of using real-time intervals instead of interpolated data. The model still performs well, but with some degradation.

There are still many variations that we could consider in order to improve the results:

- The trajectory data does not include a range to the object, which makes scaling the variables more challenging. In this work, we have chosen not to scale the inputs going into the network, but to weight the outputs in the loss function according to a local Std indicator. However, the architecture already contains a normalization layer, that can be either removed or changed, and we can also try normalizing the inputs, dividing them by the local Std as a first step.
- There are still hyperparameters that we have not exhausted in our search. For example, working with a different expansion factor, or convolution size, training with L1 loss instead of L2, trying different optimizers, etc.
- The model can be changed such that the $\bar{A}$ parameter would depend directly on the input and not only through Δ. Such a solution should still preserve $\bar{A}$ initialization as a HiPPO matrix and so a multiplicative factor $\tilde{\Delta}$ may be a good direction.

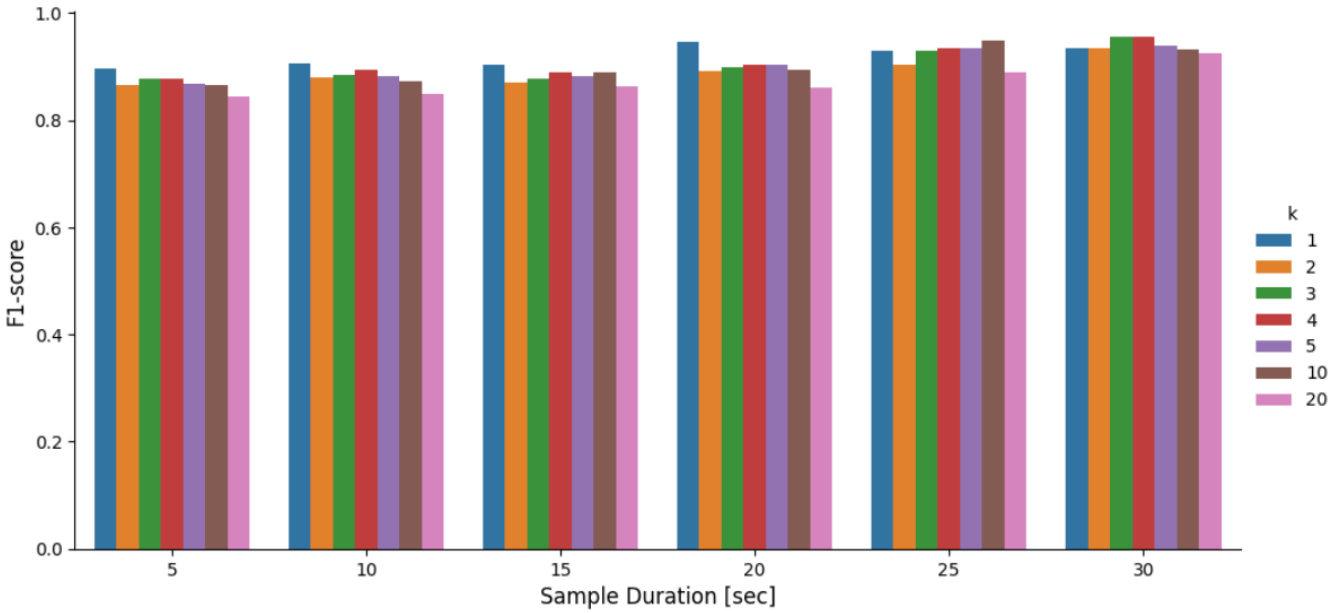

Figure 11: With k-steps prediction the model's output is used as input k times to predict k steps into the future. The results show that averaging over 1 step is almost always better than k steps, especially for the shorter samples.

- We can explore using a different input representation from what we have chosen (angles, angular velocity, acceleration and angular size). Perhaps we can use only some of the dimensions or assign them weights.

Acknowledgements

I would like to thank Prof. Alon Kipnis for his guidance and for our productive discussions. I would also like to thank the Walaris team: Anil Sönmez, Peter Gall and Amit Pandita for working hard on providing the data and for their helpful questions and feedback.

References

[1] GU, Albert; DAO, Tri. Mamba: Linear-time sequence modeling with selective state spaces. arXiv preprint arXiv:2312.00752, 2023.
[2] GU, Albert; GOEL, Karan; RÉ, Christopher. Efficiently modeling long sequences with structured state spaces. arXiv preprint arXiv:2111.00396, 2021.
[3] GU, Albert, et al. Hippo: Recurrent memory with optimal polynomial projections. Advances in neural information processing systems, 2020, 33: 1474-1487.
[4] KALMAN, Rudolph Emil. A new approach to linear filtering and prediction problems. 1960.
[5] FU, Daniel Y., et al. Hungry hungry hippos: Towards language modeling with state space models. arXiv preprint arXiv:2212.14052, 2022.
[6] Pei, R. (2024). @PeaBrane: mamba-tiny. GitHub.
[7] Ma, J.Z (2023). @johnma2006: mamba-minimal .Github.

Appendix

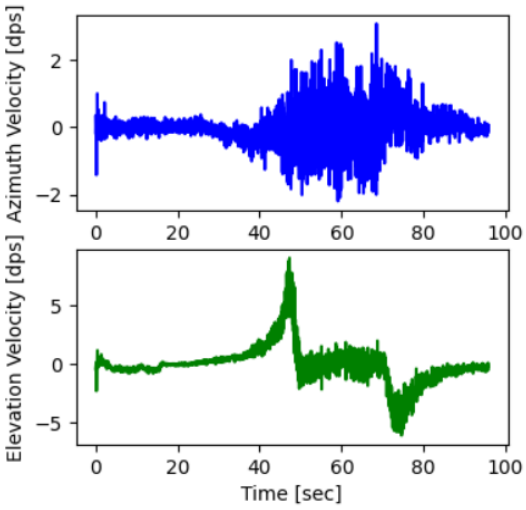

Figure 12: UAV angular velocity profile. We detect a distinctive signature for the elevation derivative during ascent and descent. Specific filters are designed to find a correlation to the pattern.

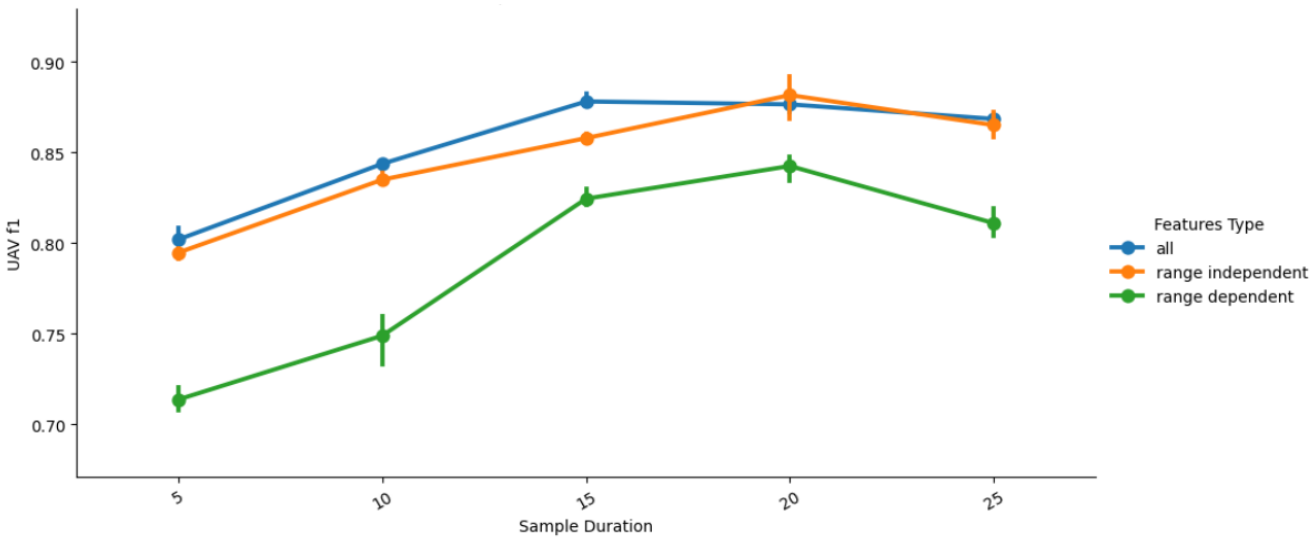

Figure 13: A comparison of range-dependent vs. range-independent groups for classification using RF. We see the advantage of the range-independent features. Note that these results belong to an older split, not compatible with the rest of the project. This split was random and not scale-balanced.

Algorithm 1 SSM (S4)

Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$

$\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
$\boldsymbol{C} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
$\Delta : (\mathtt{D}) \leftarrow \tau_\Delta(\text{Parameter})$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{D}, \mathtt{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-invariant: recurrence or convolution
return y

Algorithm 2 SSM + Selection (S6)

Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$

$\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_B(x)$
$\boldsymbol{C} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_C(x)$
$\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{B}, \mathtt{L}, \mathtt{D}, \mathtt{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
return y

Figure 14: Taken from the Mamba paper [1]. The additions in red on the right summarize the changes in upgrading S4 to the input-dependent S6 (Mamba). The model parameters $\bar{A}, B, C$ and Δ gain the dimension of batch size B and sequence length L, since they now depend on each input.

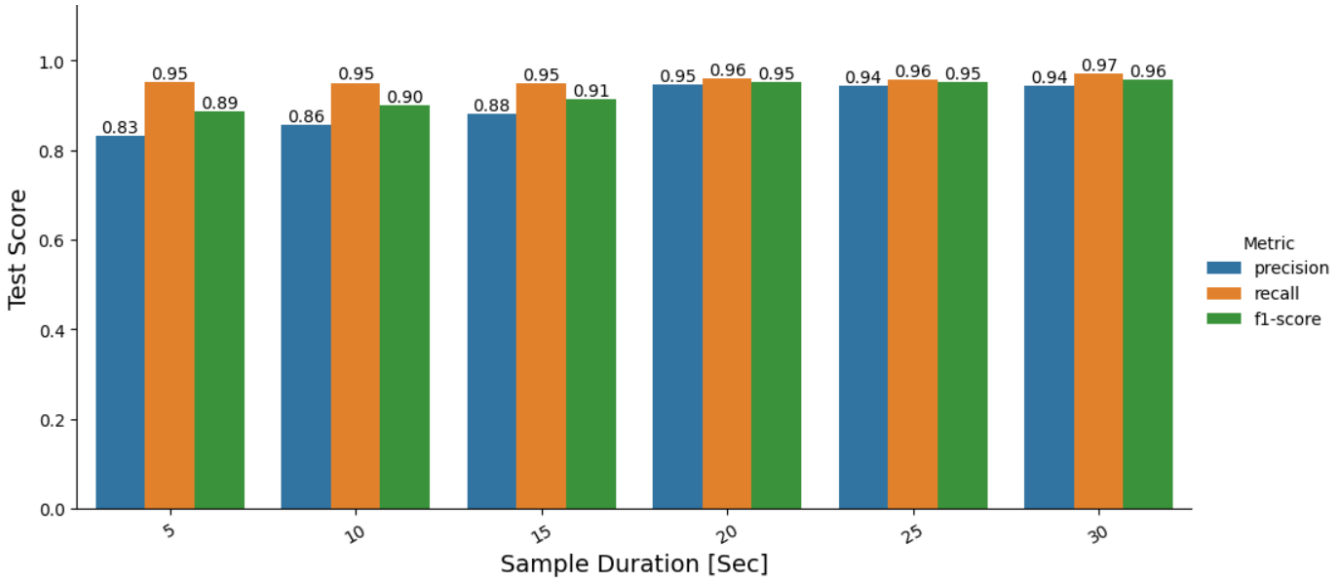

Figure 15: Mamba performance with optimization for 30-seconds samples. While the F1-score reaches 0.96, the result for the shorter samples is degraded by up to 4%

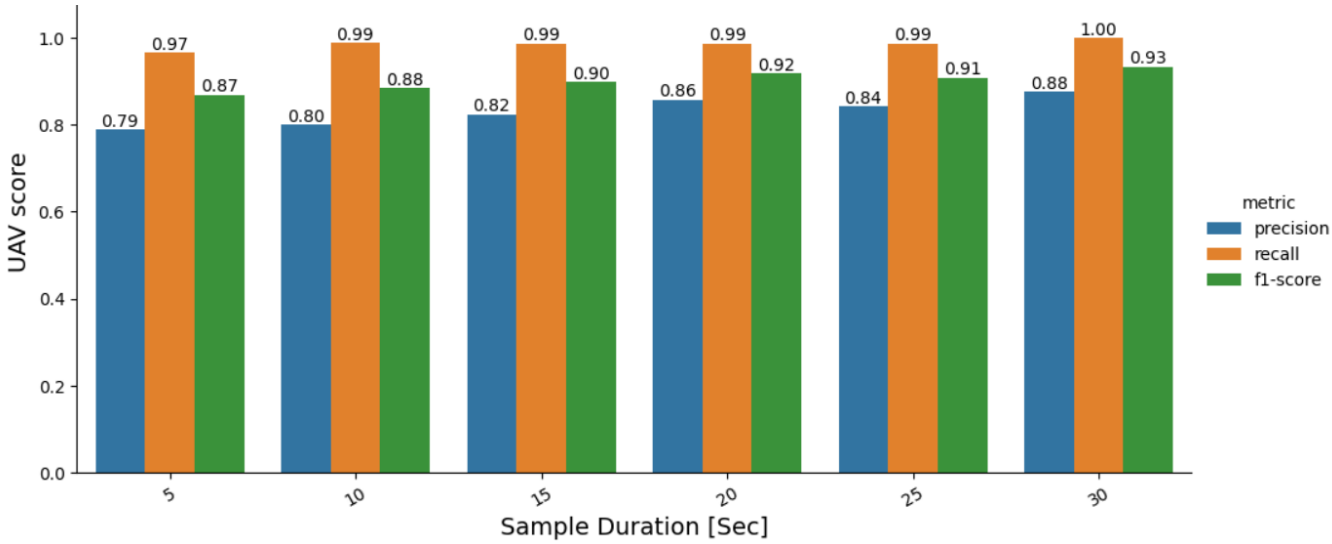

Figure 16: Mamba performance when using the raw non-interpolated data. The F1-score drops by 2-3%. Note the very high recall score.