Applied and Computational Mechanics 19 (2025) XXX–YYY

Prediction of flutter onset by an LSTM neural network from measured time-variable responses of a randomly-tested airfoil using Lyapunov exponents for flutter classification

L. Pešek[a,*], W. Schumann[a]

[a] *Department of Dynamics and Vibration, Institute of Thermomechanics, Czech Academy of Sciences, Dolejškova 5, 182 00 Prague, Czech Republic*

Received 18 November 2024; accepted 2 September 2025

Abstract

Flutter, a self-excited oscillation due to energy transfer from the flow to the structure, can cause catastrophic failures in many aerospace structures if uncontrolled. Mostly, predictions of flutter states rely on model-based evaluations under restrictive conditions, such as constant Mach numbers and altitude, which are challenging to replicate outside laboratories. To counter this problem, we investigated flutter prediction using artificial intelligence, specifically long short-term memory (LSTM) neural networks on dynamically varied operational data to simulate real-world conditions. A novel test rig of wing model in a closed circular wind tunnel with controlled airflow velocity was used for flutter simulations under variable conditions. Hundreds of vibration records, captured at critical trigger levels, formed a robust dataset for flutter classification and prediction. Average divergence and Lyapunov largest exponent methods were used to classify stability and chaos in the system, which provided valuable input data for training artificial intelligence. Analysis of results demonstrated the efficacy of neural networks in rapidly identifying flutter onset, which could contribute to advancements in flutter monitoring airborne structures under diverse operational conditions.

Keywords: flutter, prediction, neural network, largest Lyapunov exponents

1. Introduction

Aeroelasticity is a field of engineering that deals with the interaction between aerodynamic forces and the structural dynamics of an object. This field focuses on the study of how aerodynamic loads affect the deformation, vibration, and overall behavior of structures, particularly those used in aerospace engineering, such as aircraft wings, rotor blades, and control surfaces. This interaction can lead to various phenomena, including flutter and divergence, which can affect the stability, performance, and safety of the object.

Engineers use advanced computational models, experimental testing in wind tunnels, and flight testing to design structures that can withstand aerodynamic loads without experiencing detrimental effects such as excessive deformation, vibration-induced fatigue, or structural failure. Nevertheless, in the practice some situations and operational conditions can arise that could lead to dangerous conditions in structural integrity and stability.

In the context of a wing, the overall behavior of the wing is affected by static and time-variable aerodynamic lift and drag forces. The static loads can lead to divergence-type instability and variable forces to flutter-type instability. In the latter case, under certain conditions due to phase lag of the aerodynamic forces on the movement of a structure, the flow gives energy

[*]Corresponding author. Tel.: +420 266 053 083, e-mail: pesek@it.cas.cz.
https://doi.org/10.24132/acm.2025.941

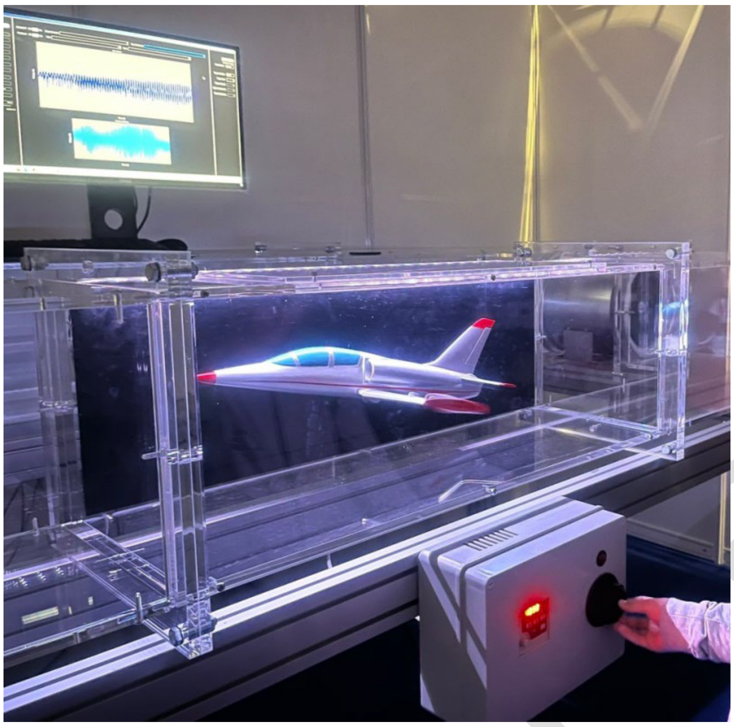


Fig. 1. Half-aircraft model exhibit with wing vibrating in a closed circle wind tunnel

to the structure, resulting in self-excited oscillations called flutter. These oscillations can lead to rapid and potentially catastrophic structural failures if not under control and surveillance. Therefore, predictions of these states are still a focus of vibration monitoring of the airborne objects. The predictions have been mostly made on model-based evaluation, e.g., [2, 8], and more recently, there has been a study of flutter speed prediction using deep learning [13]. The prediction models are quite complex but under several restrictions, e.g., constant Mach, altitude and mean air speed, which are not easy to fulfill outside of laboratory conditions.

In our approach, we wanted to study flutter prediction on randomly-generated data of wing response typical for variable operational conditions. We choose artificial intelligence (AI) with an LSTM neural network for rapid prediction of the onset of flutter under these assumptions. For testing this algorithm, we needed a large amount of data with proper classification of the flutter states for the training period. So, in the first part of the paper, the dynamic analysis of non-stationary response data by the average divergence and the largest Lyapunov exponent methods is carried out.

A new test rig developed at the Institute of Thermomechanics, Czech Academy of Sciences (IT CAS) provided the opportunity to measure and collect data of the dynamic behavior of an aircraft wing in a closed circular wind tunnel, see Fig. 1. During the Science Fair 2024 in Prague, where the test rig was demonstrated, visitors were able to control the change of airflow velocity in the tunnel up to a critical value, at which the wing started to flutter, by the manual regulation of the speed of the axial propeller. Consequently, each run was unique, which was desirable for our purpose of obtaining operational data. Vibration and fan data were monitored and measured in a loop and recorded when the vibration trigger level was reached. Hundreds of valuable records were collected. Based on these records, both the possibility of classifying the vibration signals by the largest Lyapunov exponent methods and use of neural network algorithms for fast flutter onset prediction were explored.

2. Description of the test rig – wing model in tunnel

The half model of a trainer aircraft L39 Albatros at a scale of 1:21 (tapered wing shape: length 0.17 m, airfoil spans: root 0.125 m, tip 0.074 m) was modified for demonstrating aeroelastic coupling of a wing in airflow. Since this exhibit was only for demonstration purposes, several simplifications were made to the wing design:

- The simple closed circle wind tunnel was used. Due to the impaired wind tunnel aerodynamics, i.e., higher turbulence and pulsations, the flow was non-homogeneous, and therefore the gusts caused wing oscillations even at lower speeds.
- The maximum flow velocity was around 13 m/s. The velocity was regulated by the speed of the axial propeller.
- To prevent wing damage, the wing was elastically clamped into the torsional hinge made of flat springs. The dynamics of real wings is more complicated since the vibration consists both of torsional and flexural eigen-vibrations of the wing.

The flat springs of the hinge were designed so that the torsional eigenfrequency of the wing motion was about 12 Hz. The fluttering started at about 8 m/s, which corresponds to the non-dimensional criterion of reduced frequency 0.35–0.59.

3. Classification of the non-stationary signals by Lyapunov exponents and average divergences

The concept of evaluating Lyapunov exponents in phase space is particularly well suited for autonomous dynamic systems. In these systems, the behavior is determined solely by the system's initial conditions and the underlying dynamics. Tracking how the distance between a point in a state-space and its nearest neighbor evolves helps in understanding the sensitivity of the system to initial conditions. In an autonomous system, this tracking reveals how trajectories diverge or converge over time based on their initial conditions. In unstable systems, small differences in initial conditions lead to exponentially growing divergences, which is reflected in a positive Lyapunov exponent, while a negative or zero exponents indicate stability or periodicity. Fully capturing the behavior of the system means that the reconstructed state space is using an appropriate number of dimensions (embedding dimension) accurately reflecting all relevant dynamics of the original system without losing any critical information. This concept is central in understanding and analyzing the dynamics of a system, particularly in the context of chaos theory and nonlinear dynamics.

The largest Lyapunov exponents (LLEs) [5, 7] are designed for analyzing the stability and predictability of nonlinear dynamic systems from measured signals. It also involves phase space reconstruction, determining the nearest neighbors, and then calculating the divergence of nearby trajectories over time. Taking the logarithm of these divergences, then averaging them over a certain time interval, the logarithmic average divergences (LADs) and LLEs are evaluated by the most frequently used algorithms by Kantz [7] or Rosenstein [9].

In our study case of a wing in airflow with random flow velocity settings, we get non-stationary responses of the wing—not only due to the changes in the wing-flow interaction and the system behavior—but also due to time-variable external forces coming from the changing airflow in the channel. So, besides the wing-flow interaction, the input flow velocity and the aerodynamics of the channel also play important roles in the dynamic response of the wing. To reduce these external influences on the study of the wing-flow stability, we divided the time of the signal into short window intervals where the external changes were not large. We examined

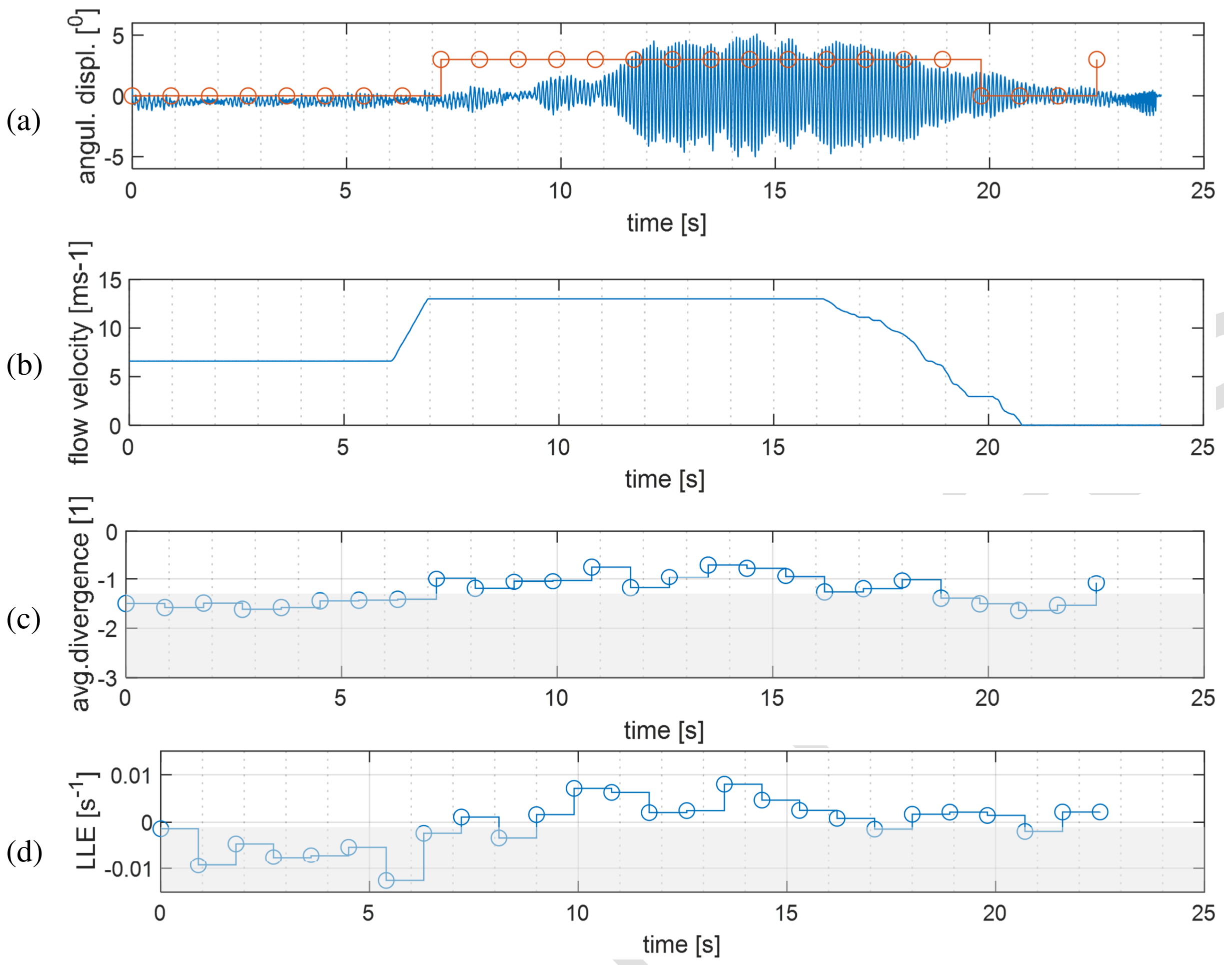


Fig. 2. Time characteristics (a)–(b) and processed window diagrams (c)–(d) for the record ”dataset_2024-06-1_12-36-52”

the wing dynamics by LLEs in these intervals where the external conditions were relatively stable. As a result, we could observe the changes of LLEs over the entire time interval of the signal.

The concept of dividing a time series into windows and analyzing each segment in order to study the dynamics (e.g., by Lyapunov exponents) can be found in various works on nonlinear time series analysis [9].

Two records, i.e., ”dataset_212024-06-1_12-36-52” and ”dataset_2024-05-31_15-41-21”, are shown as examples in Figs. 2 and 3, respectively. The number of samples was 25 402 with a sample rate of 1 000 Hz.

The diagrams at the top of Figs. 2 and 3 display time characteristics of the wing torsional displacement measured by strain gauges and also the flow velocity in the channel, evaluated from the control signal of the propeller converter. At the bottom are LAD and LLE diagrams evaluated from the angular displacements of the wing. For evaluation, the time series of the data was split into 26 windows (duration 1 s) with a 10% overlap.

Consequently, the diagrams (c) and (d) of Figs. 2 and 3 are stair diagrams. The shaded areas in diagram (c) reflect the interval with lower LAD values and in (d) a negative value zone. For early tracking of the onset of flutter by LLE, the threshold for the flutter condition was shifted from zero to -0.1. The MATLAB function *lyapunovExponent.m* [9] based on the Rosenstein

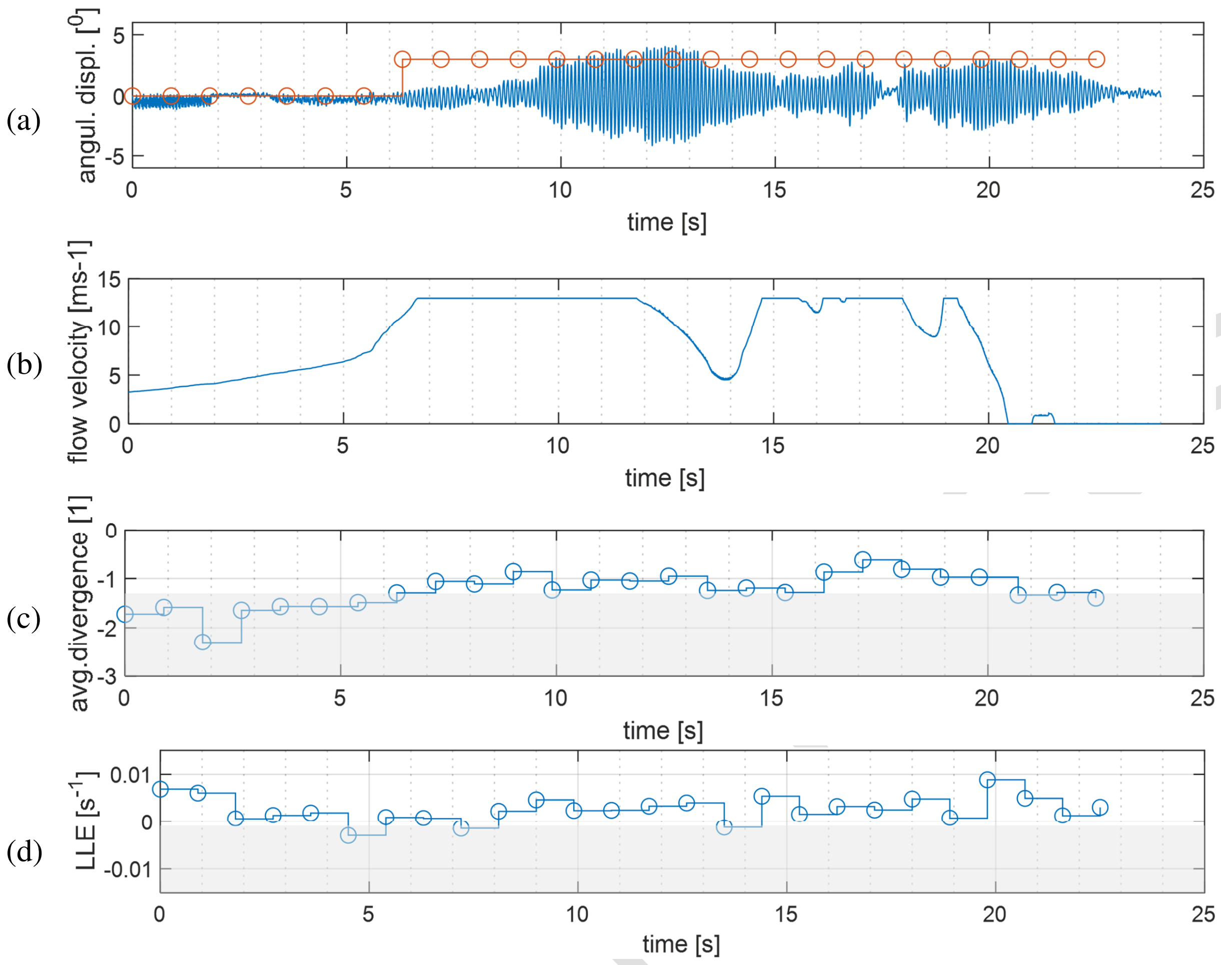


Fig. 3. Time characteristics (a)–(b) and processed window diagrams (c)–(d) for the record ”dataset_2024-05-31_15-41-21”

method was used for the evaluation of both characteristics. Similar results were achieved also by the programs *Lyap_r* and *Lyap_k* of the library TISEAN [5].

From the different flow velocity diagrams (Figs. 2b and 3b), we can see the different runs of velocities and different responses of the wing in both cases. Red rectangle domains (Figs. 2a and 3a) indicate time intervals where flutter (including its onset and reverberation) was observed.

From the evaluation of the LLE (Fig. 2d), it can be seen that the flutter was quite precisely detected both at the beginning and at the end of the record. The exponent at 9 s turned from negative to positive and is linked to the instability and chaos in the system. However, the LLE values are variable since they are sensitive to uncertainties due to the time-variable conditions in the channel. Besides that, in some cases, LLEs oscillate between positive and negative values even for small amplitudes of the wing response (Fig. 3d), probably due to signal noise. Therefore, for evaluation of flutter states, the LADs provide meaningful information about the system dynamics and signal amplitudes. When the signal amplitudes increase, the LAD values increase, so for tracking flutter state of time-varying and noisy data, besides inspection of the LLEs, the LAD values are also needed. As shown in Fig. 3c–d at the interval of 0.9 s, there are several positive LLEs, but low levels of LADs; therefore, it was not classified as flutter. The limit value of LAD_limit $= -1.3$ was estimated for our case empirically through observation of the average divergence (AD) characteristics.

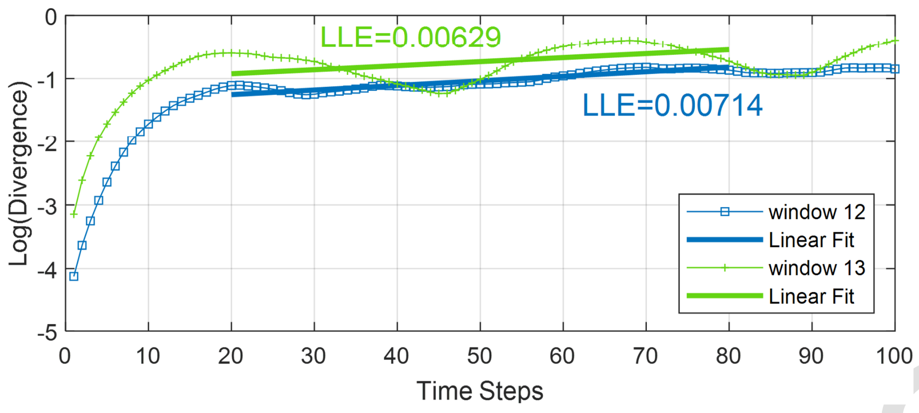


Fig. 4. LAD characteristics versus time steps with the linear fits (blue – window No. 12, green – window No. 13)

Parameters of the LLEs functions were: time delay $= 1$, embedding dimension $= 2$, number of time steps $= 100$, average interval $= (20, 80)$. The examples of time LAD characteristics of two time windows (No. 12 and 13) of the record "dataset_2024-05-31_15-41-21" can be seen in Fig. 4. The linear fits of LADs were used for LLE estimations. The logarithmic course of LAD over time for the case of window No. 12 is typical to evaluate the LLE. The LAD course of window No. 13 is wavier due to time-variable external forces in the channel, contributing to the errors and uncertainties of the LLE evaluation.

4. Neural networking model description

Given the quality of data collection that is possible, plus the relative simplicity of the usual data (linear wing displacement, wind velocity, angle of attack), it seems likely that a neural network should be able to produce accurate predictions within a time frame that would allow automated preventive measures to be taken, such as reduction of speed. This basic problem was handled as a binary classification – flutter or not. The model outputs the probabilities of both cases.

Time series forecasting with LSTM [6] was the selected algorithm for flutter prediction. The model is depicted in Fig. 5. A flowchart showing all operations for a single LSTM cell is shown in Fig. 6.

During the processing of a single LSTM cell, the following three values pass through the following gates on their way to a new cell state and a hidden state:

1. In the **forget gate**, it is decided what current and previous information is retained. This includes the hidden status from the previous pass and the current input. These values are passed into a sigmoid ($\boldsymbol{\sigma}$) function, outputting values in the range of $(0, 1)$. A value of zero means that previous information can be forgotten because there is possibly new, more important information. A value of one indicates that the previous information is to be preserved. The results from this are multiplied by the current cell state so that knowledge that is no longer needed is forgotten since it is multiplied by zero and thus drops out.
2. In the **input gate**, it is decided how valuable the current input is to solve the task. The current input is multiplied by the hidden state and the weight matrix of the last run. All information that appears important in the input gate is then added to the cell state to form the new cell state $c(t)$. This new cell state will be used as the current state of the long-term memory in the next run.

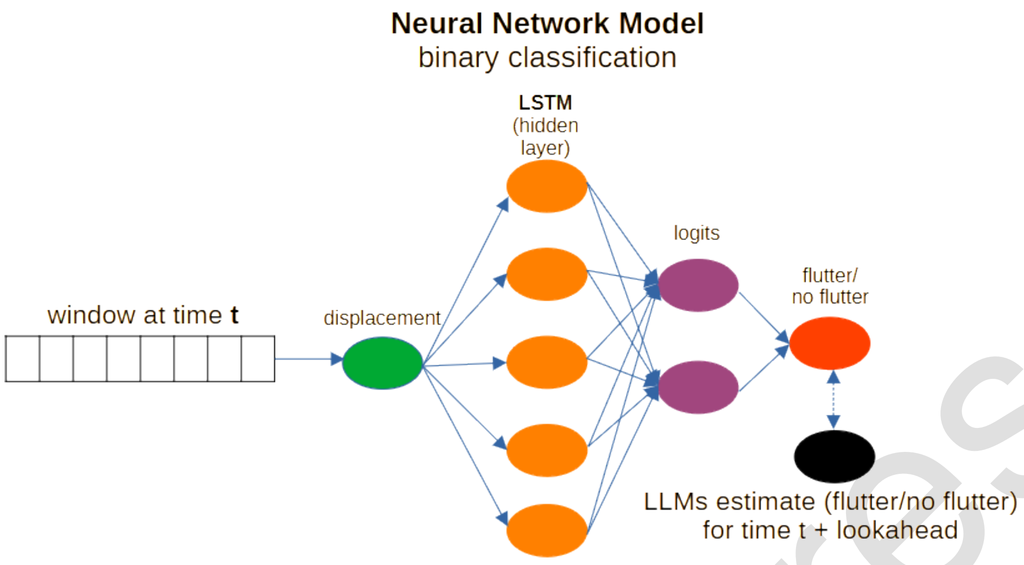


Fig. 5. The neural network model: the green nodes represent the input, the five orange nodes represent the single hidden LSTM layer, the purple nodes represent the probabilities of flutter and no flutter, the red node is a single number – 0 indicating no flutter/1 indicating flutter, and the black node represents the LLMs estimate for flutter at time t + lookahead

3. In the **output gate**, the output of the LSTM model is calculated in the hidden state. The sigmoid function decides what information can come through the output gate and then the cell state is multiplied after it is activated with the tanh function.

The PyTorch Lightning framework was chosen for software development. Python is regarded as being good for development, since it is high-level and powerful, yet most of the computation is done within its support routines and can be coded with more efficient languages, such as C/C++, which provide the foundation. Python version 3.11 was adopted as the starting version due to its efficiency and speed improvements, particularly those related to the stack frame [3]. Python 3.12 was then used due to enhanced format string capabilities.

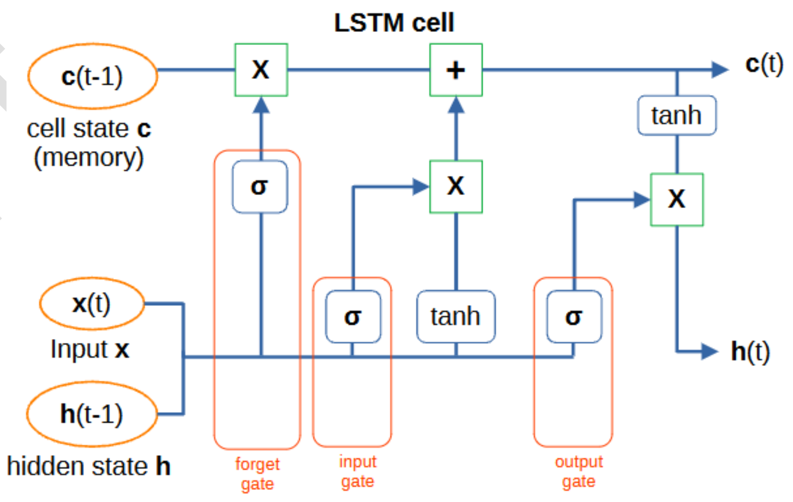


Fig. 6. Flowchart of a single LSTM cell. The plus sign and 'x' indicate matrix addition and multiplication, respectively

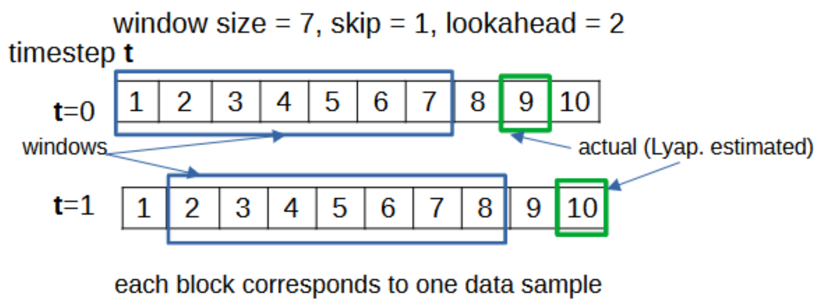


Fig. 7. Simplified sliding window protocol drawing with prediction

The flutter prediction model's Python class is derived from the class ”LightningModule” in order to use the framework. Similarly, the data module is derived from ”LightningDataModule”. The PyTorch linear classifier [10] was used. The Adam optimization algorithm to update network weights iteratively was used due to it being relatively straightforward and efficient. The cross-entropy criterion was selected to compute the loss, using weighted mean reduction [11]. As it is usually beneficial for the learning rate to be reduced as training progresses in order to improve the consistency of loss and to reach lower loss and higher accuracy, learning rate scheduling was used during training. The most successful technique was found to be Lightning's ”reduce learning rate on plateau”, which automatically reduces the learning rate by a specified factor (here, 0.1) if a loss stabilizes over some number of steps (here, 10). A more rigid reduction in learning rate in regular steps performed almost as well. Different data normalization techniques were evaluated. The most effective was a logarithmic normalization proposed by Smolík et al. [12]. The Scikit Learn package provides the class ”StandardScaler” with the method ”fit_transform” also showed improvement over not normalizing.

A sliding window protocol scheme was implemented. This allows for only the last N samples to be used for predictions at a given time step. A specified number of time steps can be skipped between windows to reduce processing and improve responsiveness in real time. For example, a factor of two instead of one will reduce the number of windows to be processed by half. A ”lookahead” defines the time period into the future the prediction is attempted. Each of these must be consistent during training and in real time. Varying these affects accuracy and performance. A sampling rate parameter to discard unnecessary samples was not implemented, although it would be useful in cases where performance was inadequate or the actual sampling rate was higher than necessary. The final training with a sample rate of 1 Hz was based on a window of 600 (0.6 s) and prediction ”lookahead” of 2 000 (2 s) with no window skipping.

Fig. 7 shows a drawing illustrating the first two time steps of a sliding window, assuming a window size of seven, a skip of one and a lookahead of two.

With LSTM predictions, for a given time step, the window containing the displacements for all time steps in the window is fed into the model, and the flutter/no flutter value for the time step plus the lookahead is provided in parallel as a target.

With skip $= 1$, as used in the final model, each time step has a unique effect.

5. Data preparation, training and tuning stages

Data is stored in 32-bit float variables, which is a generally accepted convention for neural networking. The data were collected during the Science Fair 2024 and consisted of the flow velocity and the wing displacement. Only the wing displacement is input to the model. The character of the input varied widely, since the velocity was determined by the visitors of the exposition – mostly students. The resulting variation of samples improved the quality of learning.

The initial samples were MATLAB files **.mat* with corresponding results of the Lyapunov exponent analysis, indicating either flutter or no flutter designated in windows covering multiple time steps. A custom Python class was to encapsulate the preprocessing and input of samples into Lightning datasets for this specific application.

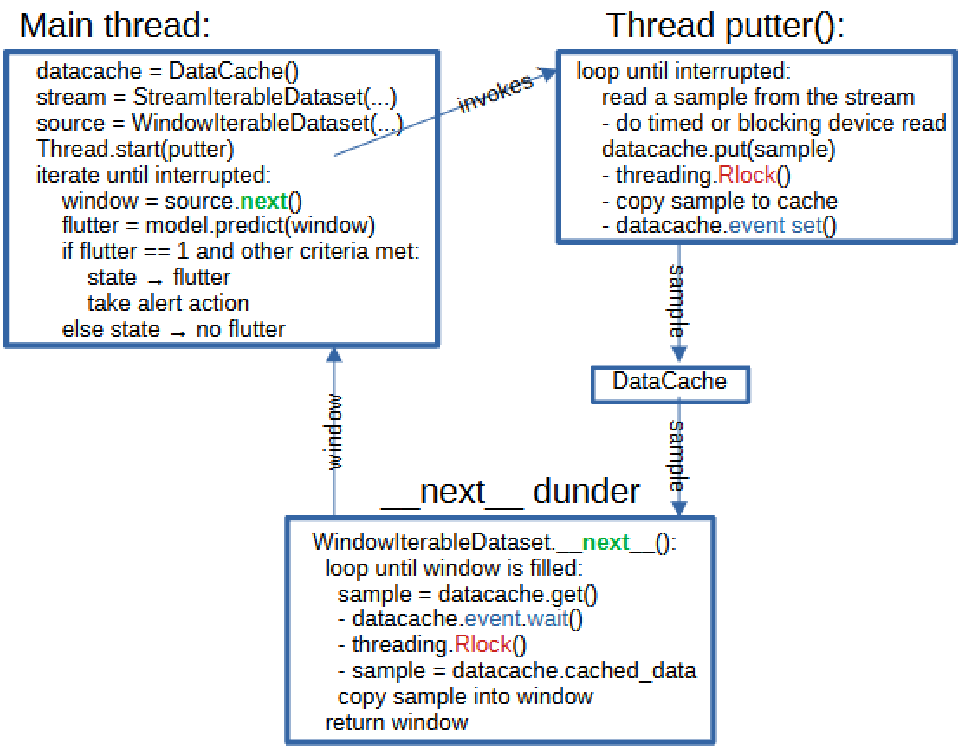
In Lightning, the training data is stored in a file referred to as the ”checkpoint file”. Applying training to existing training data, enabling learning over time, the checkpoint file is provided to the trainer before evaluating each sample, and updated after the sample has been learned. Each training cycle is an ”epoch”, meaning that the sample has passed through the neural network one time. Multiple epochs are typically required to get a result with the least-possible loss, and a maximum number of epochs must be specified to prevent the training to continue indefinitely. As training sessions are finished, the trainer saves the checkpoint file with the best results for the session, writing the last epoch number to the checkpoint file. To resume training, the maximum epoch must be adjusted by increasing it by the epoch number from the checkpoint file. The initial value for maximum epochs was 50.

There are other important considerations during tuning, training and testing:

- A key point in feeding training data into the model is that the input (here, angular displacement) is provided in parallel with the LLEs estimated flutter, except that the LLEs estimates are shifted into the future with respect to the input. For example, if the displacement was read at time t, the corresponding prediction is not taken from time t, but $t +$ lookahead, which in this case was 2 000 (2 s). The input/classifications pairings are fed into the model with this offset.
- Care is taken such that the windowing parameters remain identical throughout training, as well as the hyperparameters and normalization technique and other algorithms.
- The typical approach to split training and test data during training was done, so that a fraction of the data is not used to train, but to validate the training during the training phase. The data collected on the first three days was used for training and data from the last day was used strictly for testing.
- Tuning objectives prioritized accuracy and also responsiveness since the prediction will be done in real time. It was found that a lightweight depth of one layer with five hidden layers was sufficient. This can be attributed to the relative simplicity of the input data.
- The initial learning rate of 0.01 was selected since it quickly minimizes loss, and as more samples were input, the learning rate scheduler reduced it to 0.001, which proved to be a better starting point for increased accuracy as learning progresses. With the learning rate scheduler algorithm ”reduce learning rate on plateau”, the factor was set to 0.1 and patience to 5.
- An early stopping function was used to avoid lengthy training when nothing new was being learned. For early stopping, validation loss was monitored and patience was set to five epochs.

6. Real time prediction

Since a real-time interface was not available for extensive testing, a comparable scheme involving a separate thread streaming sample data at real-time rates was developed. A flowchart of



```
datacache = DataCache()
stream = StreamIterableDataset(...)
source = WindowIterableDataset(...)
Thread.start(putter)
iterate until interrupted:
    window = source.next()
    flutter = model.predict(window)
    if flutter == 1 and other criteria met:
        state → flutter
        take alert action
    else state → no flutter
```

```
loop until interrupted:
    read a sample from the stream
    - do timed or blocking device read
    datacache.put(sample)
    - threading.Rlock()
    - copy sample to cache
    - datacache.event set()
```

```
WindowIterableDataset.__next__():
  loop until window is filled:
    sample = datacache.get()
    - datacache.event.wait()
    - threading.Rlock()
    - sample = datacache.cached_data
    copy sample into window
  return window
```

Fig. 8. Chart describing data stream processing in Python pseudocode, emphasizing iteration (green), locking (red) and events (blue)

data stream processing in Python pseudocode is in Fig. 8. As efficiency is critical and Python's efficiency in many circumstances is questionable, the most efficient data passing mechanisms were sought. A variable can be shared in memory between threads in Python, so data does not have to be transmitted, piped or queued—only synchronized—and the shared data is actually a Python variable of any data type. Thread synchronization is done using an "Event object", and locking uses an "Rlock object" – both from Python's threading class. Upon writing a sample to a shared variable, the writing thread sets an event that the reading thread waits for. The primitive nature of locking and events suggests that high performance is possible, and a coding solution using them was found in [1] under named class "DataCache".

A separate thread reads from the source and uses the DataCache "put" method to provide samples to the main thread at the rate of the original sampling. This thread could be modified to do take samples from a device in parallel to the neural network processing. Custom iterable classes to manage the stream and window processing were created. A Python iterable class is advantageous since it can be iterated with many different Python techniques. In the main thread, a corresponding object reads from the DataCache object, composes the new window of data as a "PyTorch Tensor", which is basically an array, and returns the window. In this window reader thread, the composition and return of windows is done mainly in the Python class "__next__ dunder" method, which accumulates samples from the driving thread and returns a window for every iteration. A Python dunder method is not invoked directly, but is used to provide programming when logic a built-in operation is not provided by default – in this case, iteration to the next window in the stream is done by the __next__ method, requiring custom coding. The input then is run through the neural network using the model's predict method.

As DataCache was implemented, there is no notification that data is not lost if it is not read before the next sampling interval. A useful enhancement would be for DataCache to monitor the frequency of sample loss and to address that if it exceeds a certain level.

A requirement for this mechanism to work efficiently is a dedicated device or operating system with real-time clock granularity. Also, PEP 703 is a welcome enhancement planned for Python 3.13, since threading for multiple-CPU computers would be more efficient [4].

7. Results evaluation

Predictions for the Science Fair data required some scrutiny due to a high percentage of false negatives, i.e., time steps that were predicted to exhibit no flutter, but were reported as exhibiting flutter according to the LLEs estimates, as well as false positives, conversely. Plotting angular displacement against the AI-predicted flutter showed plausible reasons for the discrepancies. This evaluation is described below and depicted in Figs. 9–10. The physical units of measured signals and LLEs are the same as in Figs. 2–3.

An example of the LLEs estimates vs. AI-predicted flutter states for one selected sample (AIveletrh_2024-06-01_14-41-14) is shown in Fig. 9. The wing displacement is represented by blue, purple represents flutter logit values (non-dimensional), and velocity is represented by brown. A logit is a non-normalized value representing a prediction of an AI model. If the logit value at a time is positive, flutter is more likely than not, and as the absolute value of the logit increases, the predicted likelihood increases. Although the logit is a raw, dimensionless value, it is useful to plot, since it shows how certain the AI is about its prediction.

Offset at about $y = -7$ for better visibility are green and orange square waves showing the LLEs flutter/non-flutter estimates (green) versus the AI-predicted state (orange). When the green line goes high, the LLEs estimation of flutter begins at that timestep; when the orange line goes high, AI predicts flutter. The blue line indicating the displacement shows flutter at the times when the absolute values are relatively high.

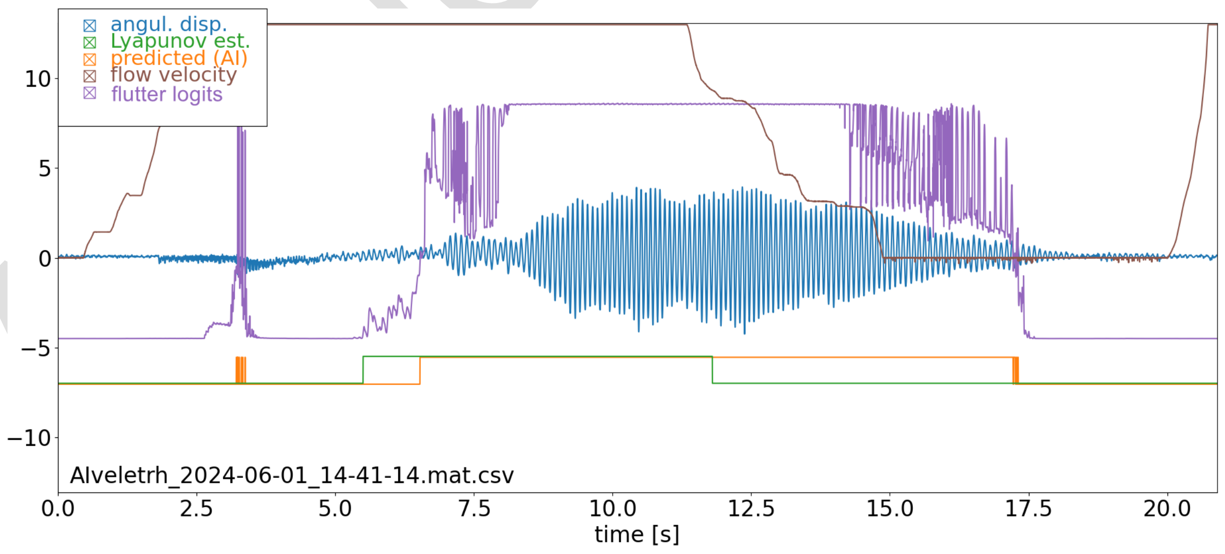


Fig. 9. Results of flutter prediction of the sample vs. the LLEs estimate: the wing displacement (blue), the flutter logit values (purple), flow velocity (brown), the AI-predicted flutter (orange), the LLEs estimate of flutter (green)

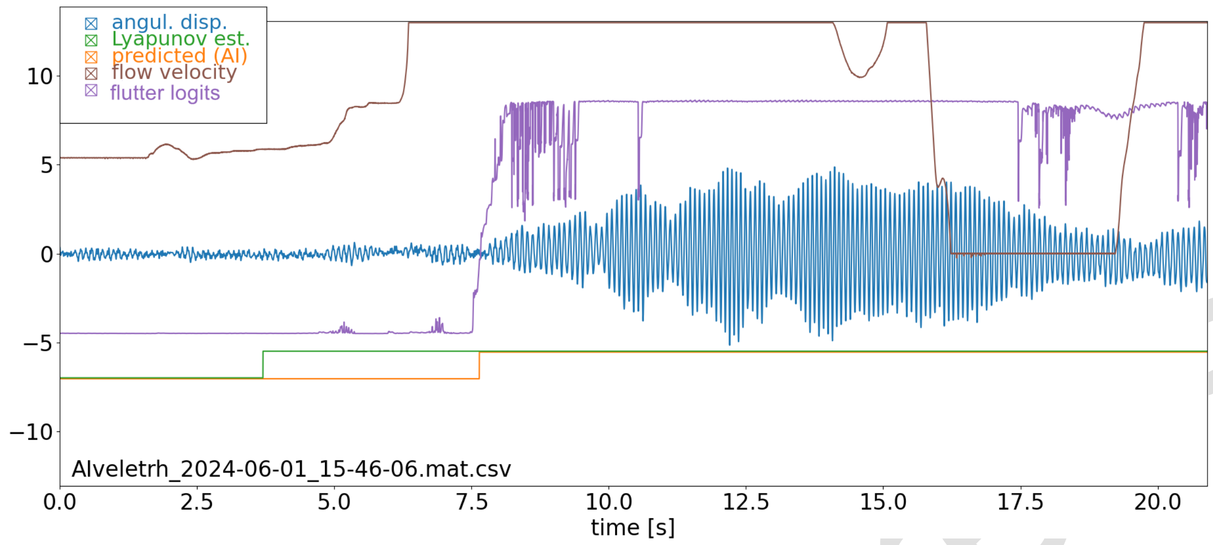


Fig. 10. Results of flutter prediction of the sample vs. the LLEs estimate: the wing displacement (blue), the flutter logit values (purple), flow velocity (brown), the AI-predicted flutter (orange), the LLEs estimate of flutter (green)

The main objective is for the AI to predict the onset of flutter. There are very brief flutter predictions in response to strong interference of the measured signal by the fan converter; these signal noise interferences can be dismissed from the flutter prediction due to their very short duration or may disappear with more training samples exhibiting the interference.

In this case, it can be seen that the AI predicted flutter just before the flutter became evident in the displacements. The LLEs estimate came slightly after the actual flutter onset. It is interesting to note that the AI predicted flutter much more closely than the LLEs estimates. Seeing this pattern in other samples strongly suggests that the AI is able to improve its predictions despite imprecise cues from the LLEs estimates, which have some inherent imprecision due to the fact that they are windowed. Similarly, the drop in flutter was predicted more accurately than the Lyapunov estimate and can be seen as a correction made possible through learning.

Fig. 10 was taken from a similar sample exhibiting normal running vibrations followed by an onset of flutter (AIveletrh_2024-06-01_15-46-06). In this sample, the onset of flutter was predicted much more precisely than in the Lyapunov estimate. This is another example of what was generally observed in samples displaying a leading non-flutter period: the prediction (orange) was very close to the flutter observed in the displacement (blue). Again, despite inaccuracies or misleading LLEs estimates, the AI delivered precise predictions of the onset of flutter.

The flutter logit value (purple) is useful for diagnosing situations in which flutter is predicted, but not so convincingly. This can be useful for signaling a warning state that could draw attention to the condition so that preventative action might be taken. There is a corresponding non-flutter estimate that was not indicated in the graphs because it is generally a mirror image of the flutter prediction. The final prediction is generated by a ”binary accuracy” function that accepts the two logits as input – one representing flutter, the other representing the non-flutter state.

8. Conclusions

The advanced methods, such as the largest Lyapunov exponents and long short-term memory, were studied for classification and prediction of flutter states of the airflow wing. Time-varying wing responses with flutter states measured in the IT CAS wind tunnel built for the Science Fair 2024 exposure were used as test samples.

The results of these methods are presented and discussed in this paper. Regarding the LLE method, which is used to classify stability and chaos in the system, it was found that the flutter was quite precisely detected for some records just by inspection of the LLE values. However, the LLE values are variable since they are sensitive to uncertainties due to the time-variable conditions in the channel. In some records, LLEs oscillated between positive and negative values even for the small amplitudes of the wing response due to noised signal. Therefore, for the tracking of flutter states from time-varying and noisy data, besides the LLEs, ADs values also needed to be examined. The LLE, together with LAD characteristics, can contribute to a better classification of complex real-world systems that work under multiple influences over time. It can be also a useful tool for automatically classifying flutter states of a coupled system for further processing by AI methods.

The LSTM method after training, anomalies, i.e., false positives and negatives, began to reflect actual flutter even more closely than values provided by the Lyapunov estimations. The AI model was sufficiently simple so that it was able to process data at real-time rates while maintaining a high level of accuracy. The synchronization available in Python appears to be sufficiently fast to be used in more complex real-time applications such as aircraft wing flutter prediction.

Acknowledgement

This research is supported by the project of Czech Science Foundation No. 24-12144S ”Investigation of 3D flow structures and their effects on aeroelastic stability of turbine-blade cascades using experiment and deep learning approach”.

References

[1] Bruen, T., Handling real time data in Python, 2019, available from https://medium.com/@teebr/handling-real-time-data-in-python-54ca97a40b62.

[2] De Troyer, T., Zouari, R., Guillaume, P., Mevel, L., A new frequency-domain flutter speed prediction algorithm using a simplified linear aeroelastic model, Proceedings of the International Conference on Noise and Vibration Engineering ISMA, Leuven, 2008, pp. 1 197–1 206.

[3] Faster Runtime, Python 3.11 release notes, 2022, available from https://docs.python.org/3/whatsnew/3.11.html.

[4] Gross, S., PEP 703 – Making the global interpreter lock optional in CPython, 2023, available from https://peps.python.org/pep-0703/#the-gil-makes-many-types-of-parallelism-difficult-to-express.

[5] Hegger, R., Kantz, H., Schreiber, T., Practical implementation of nonlinear time series methods: The TISEAN package, Chaos 9 (2) (1999) 413–435. https://doi.org/10.1063/1.166424

[6] Hochreiter, S., Schmidhuber, J. Long Short Term Memory, Wikidata Q98967430, 1995.

[7] Kantz, H., Schreiber, T., Nonlinear time series analysis, Cambridge University Press, 2004.

[8] Kim, T., Flutter prediction methodology based on dynamic eigen decomposition and frequency-domain stability, Journal of Fluids and Structures 86 (2019) 354–367. https://doi.org/10.1016/j.jfluidstructs.2019.01.022

[9] Manual for predictive maintenance toolbox, MATLAB 2019b, 2019.

[10] PyTorch documentation for the linear classifier nn.linear, 2024, available from https://docs.pytorch.org/docs/stable/generated/torch.nn.Linear.html.

[11] PyTorch documentation on CrossEntropyLoss, 2024, available from https://docs.pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html.

[12] Smolík, L., Rendl, J., Bulín, L., Employing nonlinear transformation of datasets to train neural networks, Humusoft Technical Computing Camp, 2024.

[13] Wang, Y.-R., Wang, Y.-J., Flutter speed prediction by using deep learning, Advances in Mechanical Engineering 13 (11) (2021) 1–15. https://doi.org/10.1177/16878140211062275