User Guide
Low code, high performance embedded AI with MATLAB and Arm IP Explorer

Version 1.0

Brenda Zhuang, Akshay Rajhans, Eric Sondhi

September 23, 2024

Contents

1 Introduction

The intersection of AI and embedded systems represents a frontier of technological innovation. With the exponential growth in IoT devices and the advancement of AI models, there is a pressing need for professionals who can effectively deploy AI in resource-constrained environments. The goal of this tutorial is to present a low-code end-to-end workflow in an interactive hands-on format.

The tutorial focuses on design, optimization, and deployment of AI algorithms on Arm processors, typically used in powerconscious embedded devices. Using MATLAB, participants will learn to start from a high-level algorithmic design, optimize it, and auto-generate optimized C code. Arm IP Explorer complements this by offering the detailed insights into the performance metrics. This synergy allows for easy fine-tuning of applications to achieve performant solutions using reliable benchmarks.

The tutorial will engage in practical exercises, such as applying the end-to-end MATLAB workflow on curated dataset and collecting cycle-accurate runtime metrics from simulations in the Arm IP Explorer

2 System Requirement

To participate in the workshop, you need:

1. A laptop
2. Google Chrome browser
3. A MathWorks account
4. Arm account

You will be provided with a temporary MATLAB workshop license that will give you access to all products used in the workshop, as well as the workshop exercise files. If you don't have an Arm account, you need to create one to get access to Arm IP Explorer and use the material for this event.

3 Understand the dataset: Viberation and Electrical Signals in Motors

We will first explore the role of datasets in the context of machine learning applications for industrial motors. Specifically, we will focus on datasets used for monitoring and diagnosing the health of motor systems.

Vibration and electrical signals provide a wealth of information about the operational state of motors. By analyzing these datasets, we can identify patterns and anomalies that indicate potential issues, such as mechanical wear, imbalance, or electrical faults. This understanding enables predictive maintenance strategies, reducing unplanned downtime and enhancing the operational efficiency of industrial systems.

In this tutorial, we aim to detect failures in an AC induction motor. We will explore several data analysis methods to gain insights into preprocessing techniques, including filtering noise and normalizing data, to prepare for analysis. We will discover how to extract meaningful features from raw data and use them to train machine learning models for fault detection and predictive maintenance.

3.1 Smaller dataset

The experimental workbench consists of a three-phase induction motor coupled with a direct-current machine, which works as a generator simulating the load torque, connected by a shaft containing a rotating torque meter. For more details about the workbench setup, see IEEE Broken Rotor Bar Database. The data set contains electrical and mechanical signals from experiments on three-phase induction motors. Electrical and vibration signals were collected from the system under the health and loading conditions with ten experiments in the full dataset. In this tutorial, we will use smaller dataset that contains only two experiments per operating condition.

3.2 Datastore

In the context of data management and machine learning, particularly when using environments like MATLAB, the concepts of datastore and fileEnsembleDatastore are important for handling large datasets efficiently.

A datastore is a high-level abstraction for accessing large collections of data that are too big to fit into memory at once. It allows you to work with data incrementally, loading only a portion of the data into memory as needed. This is particularly useful for large datasets such as collections of images, large text files, or extensive tabular data.

These abstractions are crucial for workflows involving big data and complex simulations, enabling efficient data handling and processing.

```
ens = fileEnsembleDatastore(location ,'.mat');
ens.ReadFcn = @readMemberData;
ens.WriteToMemberFcn = @writeMemberData;
```

Now we can easily access the data using read interface.

```
T = read(ens);
```

4 Diagnostic Feature Designer app to extract features

The Diagnostic Feature Designer app is a powerful tool in MATLAB designed to facilitate the extraction and analysis of features from time-series data, particularly for applications in predictive maintenance and fault diagnosis. This app provides an interactive environment where engineers and data scientists can explore, preprocess, and extract meaningful features from datasets that typically include sensor data, such as vibration, temperature, or pressure readings.

You can then import the file ensemble datastore into the app by clicking the New Session button and selecting the ens variable from the list. Once preliminary data is read by the app, make sure to uncheck the Append data to file ensemble option if you do not want to write new features or data back to the ensemble member files. When you are working with the smaller data set in this example, generating new features directly in memory is faster, so uncheck the option.

Select the time-domain signals to import in the tree view of the New Session dialog. Here, you only use the synthetic signals, so uncheck Ia and *Vib_acpi*. Keep the envelope of the radial vibration speed on the driven side *Vib_acpi_env* checked; you do not need to modify it otherwise.

The Auto Features option is a fast way of creating a large number of features from available signals and ranking them according to their power in separating various health classes. For the imported time-domain signal *Vib_acpi_env*, apply the Auto Features option by selecting the signal in the Data Browser tree and clicking Auto Features on the Feature Designer tab.

5 Train the models in the Learner App

Begin by using the Diagnostic Feature Designer app to extract relevant features from your dataset. This involves preprocessing your data, selecting segments of interest, and computing various features that can help differentiate between different classes or states in your data.

Once you have extracted and selected your features, export them from the Diagnostic Feature Designer app. This typically involves saving the features and their associated labels to a format that can be imported into the Classification Learner app, such as a MATLAB table or a CSV file. Launch the Classification Learner app from MATLAB's Apps tab. This app provides a user-friendly interface for training, validating, and comparing different classification models.

Import the feature dataset into the Classification Learner app. Ensure that your data includes both the features and the corresponding class labels necessary for supervised learning.

Choose a classification model to train. For this workflow, select a decision tree model, which is a popular choice for its interpretability and effectiveness in handling complex datasets.

6 Troubleshooting

Provide solutions to common problems and FAQs.

7 Appendix

Include any additional information or resources.

8 References