

Industrial Productivity Solutions Guide

Adding a sixth sense to your
industrial machines



Introduction.....	3
About This Guide.....	4
Prerequisites for Following the Tutorial Steps in This Guide.....	5
The Impulse.....	6
On-Device Performance Estimation.....	7
Data.....	8
About the Dataset.....	9
Setting Up an Organization Dataset.....	10
External S3 Bucket.....	10
Upload Portal.....	13
Importing Data into the Project.....	13
CSV Wizard.....	13
Direct Ingestion from Device.....	14
Pre-Processing and Transformation.....	14
DSP — Feature Engineering at the Edge.....	16
Machine Learning Model.....	17
Training.....	18
Testing and Evaluation.....	18
Automated Machine Learning with EON Tuner.....	19
Creating the Impulse — Step-by-Step Guide.....	21
Step 0: Import Data to Your Project.....	21
Step 1: Select All the Blocks That Will be Part of Your Impulse.....	22
Processing Block.....	22
Learning Block.....	23
Step 2: Generate Features.....	23
Spectral Features.....	23
Flatten.....	24
Step 3: Train the Machine Learning Model.....	26
Step 4: Test the Machine Learning Model.....	27
Deploying the Impulse — Putting it All to the Edge.....	28
Prerequisites.....	30
EON Compiler.....	30
Download the C++ library and prepare the application.....	30
Build and Test the Application.....	32
Automation.....	34
Automating Data Import and Transformation — Organization Data Pipeline.....	35
Step 1 — Copy the Transformation Job as a Pipeline Step.....	36
Step 2 — Create the Automatic Pipeline.....	36
Automating Model Retraining and Deployment.....	38
Step 1 — Navigate to Data Source Configuration in the Project.....	38
Step 2 — Configure Pipeline Steps and Interval.....	39



Collaboration.....	40
Summary.....	41
F.A.Q.....	41
What are the minimum hardware requirements to run the Edge Impulse inferencing library on my embedded device?.....	42
What frameworks does Edge Impulse use to train the machine learning models?.....	42
What engine does Edge Impulse use to compile the Impulse?.....	42
Is there a downside to enabling the EON Compiler?.....	43
Can I use a model that has been trained elsewhere in Edge Impulse?.....	43
How does the Feature Explorer visualize data that has more than three dimensions?.....	43
What is the typical power consumption of the Impulse running on my device?.....	43
What is the .eim model format for Edge Impulse for Linux?.....	44
How is the labeling of the data performed?.....	44
About the Author.....	44



Introduction

Edge Impulse is the leading software platform that helps companies build and deploy real **machine learning** (ML) applications **at the edge**. Building production-grade and edge-ready ML pipelines with Edge Impulse helps enterprises unlock more value than ever by converging information technology (IT) and operational technology (OT) in the industrial space. We accelerate time to market, improve ML outcomes, and de-risk on-device deployment.

Edge Impulse (noted as **EI** throughout this document) offers a comprehensive solution that caters to various stages of a standard machine learning (ML) pipeline. It resembles the toolkit of a skilled architect designed for building skyscrapers. Just as an architect starts with a foundation, builds upwards, adjusts to challenges, and integrates new designs and technologies over time, EI provides the tools to start, adapt, and refine [ML solutions](#), ensuring they're **robust, current, and optimized for your data**.

With a focus on data collection, cleaning, feature extraction, model training, testing, and deployment, Edge Impulse ensures that users can access the necessary tools at each step. Users can efficiently manage the entire ML workflow, from data ingestion to model deployment, by seamlessly connecting these components through a unified software platform. Beyond its integrated approach, Edge Impulse provides [APIs and SDKs](#), enabling users to customize their workflows and integrate with external tools when needed.

Moreover, Edge Impulse acknowledges the importance of **collaboration**, incorporating collaboration tools while maintaining a strong focus on [security and compliance](#), which is particularly relevant for teams working in the industrial productivity domain. With these features, Edge Impulse empowers professionals of any ML expertise to build and deploy ML models to the edge effectively.

Upcoming sections of this guide delve deeper into the features and capabilities that Edge Impulse offers to aid understanding and provide insight on using it effectively for industrial machine health and productivity ML applications.

About This Guide

This guide serves as a reference example of using the Edge Impulse platform for building an edge machine learning (ML) application to enable an industrial predictive maintenance use case.

It is designed as a tutorial, providing descriptions and examples for each part of building an Edge ML application, namely:

- **Data Collection, Cleaning, and Transformation**
- **Digital Signal Processing (DSP)** — i.e., Feature Engineering
- **ML Model Construction** — how to choose, train, test and tune
- **Deployment** of the resulting inference library on any target device

In addition to the Edge ML context for the abovementioned parts, the guide introduces a notion of an [Impulse](#) — which Edge Impulse defines as a processing pipeline including DSP and ML model. This is followed by the [Data](#) section that provides a real-world example dataset to demonstrate how to work with data in the Edge Impulse platform and illustrate an example industrial use case.

The section [Creating the Impulse — Step by Step Guide](#) follows with a sequential explanation of all the actions necessary to create a complete Edge ML pipeline — from importing a dataset to training and evaluating a machine learning model. An **impulse** is the fundamental part of the Edge Impulse platform — it represents a pipeline consisting of feature engineering and a machine learning model that is deployed to the edge device.

The section [Deploying the Impulse — Putting it All to the Edge](#) provides instructions on deploying the resulting ML inference pipeline to an edge device. You are encouraged to follow through — all the resources, including a sample dataset, are [publically available](#) and referenced throughout this guide.

Everything described in the guide (and more) can also be performed programmatically through our [APIs or Python SDK](#).

In addition, links to relevant articles in our comprehensive [documentation portal](#) are provided for an in-depth understanding. There, one can find more information about each of the steps and features covered in this guide, alongside other helpful information.



Prerequisites for Following the Tutorial Steps in This Guide

- Clone a GitHub repository
github.com/edgeimpulse/industrial-solution-guide
It contains the code that is demonstrated for some of the features (section **[Pre-Processing and Transformation](#)**), as well as a copy and a link to the publically accessible dataset used in this guide
- Become a part of an Edge Impulse organization — if you are not a paying customer yet you can get access to an organization as part of enterprise trial
studio.edgeimpulse.com/trial-signup
- Set up an AWS S3 bucket (to be able to work with organizational dataset features)
- Create a project in the Edge Impulse platform, or copy a [pre-made project](#) that already has everything set up.



The Impulse

The Edge Impulse platform allows one to create an **Impulse** — a machine learning pipeline that will eventually run on the target device. A combination of an impulse and a dataset constitutes a [project](#). An impulse consists of “blocks” representing the steps in the ML pipeline:

- **Input data block:** Creating a training dataset and selecting applicable input parameters, such as window size and the sampling frequency for time series, or resizing resolution for images
- **Processing block:** Selecting the DSP algorithm, tuning the algorithm parameters, and generating features from input data
- **Learning block:** Selecting and tuning the machine learning model, and **training / retraining the model** using the features generated by the DSP block.

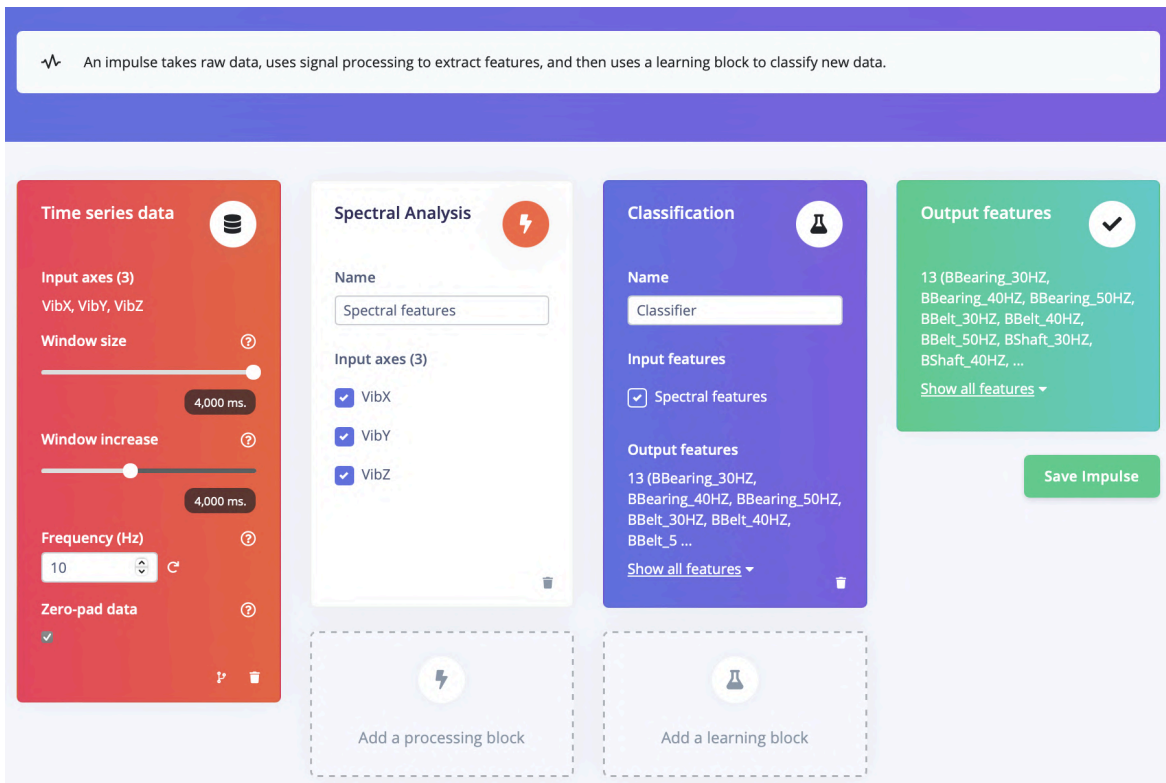


Figure 1: Overview of the impulse

The Edge Impulse platform provides all the infrastructure necessary for each step of Impulse creation, including **CPU and GPU compute** for model training and DSP feature generation, AutoML tools such as [EON Tuner](#), as well as **graphs and visualizations** for evaluating the performance, and other advanced features.

For each block type, Edge Impulse has already developed a large set of [processing algorithms](#) and [ML models](#) that can be selected for composing an Impulse.

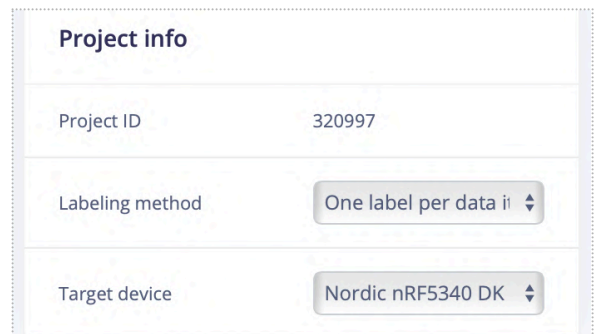
Users can extend the set of blocks available to compose an impulse in the platform. This can be achieved via [Custom Processing Blocks](#) and [Custom Learning Blocks](#). Custom blocks are helpful if the user has an existing DSP algorithm implementation or an ML model architecture created outside of Edge Impulse that is specific to some sensor data or a particular use case.

Once the impulse is created, the whole pipeline can be deployed to a target device. The **Deployment** step allows to generate a C++ library that contains the version of the impulse highly optimized specifically for **inference** on the target device or gateway that is used in your project.

Further sections cover each step of creation and deployment separately.

On-Device Performance Estimation

As mentioned above, each block of the resulting impulse will eventually run on the edge device. Therefore it is valuable to be able to have an idea about how each step of the pipeline will perform once deployed as early as possible to avoid spending time on algorithms that might not fit the selected device.



Project info	
Project ID	320997
Labeling method	One label per data i ▾
Target device	Nordic nRF5340 DK ▾

Figure 2: Device selection at project Dashboard

To be able to see these estimations, select the target device in the **“Dashboard”** section of your project (**Project Info** pane). Now, any time there is any modification to processing and learning blocks the live performance metrics estimations will be updated. Metrics include latency, memory usage, and storage requirements and are visible on the respective blocks’ pages. A total estimation of the whole impulse will also be provided on the deployment page at the final step of the impulse creation.

Figures 3a and 3b show estimation for DSP and ML model blocks of an impulse for the target selected for this project — [Nordic nRF5340 DK](#).

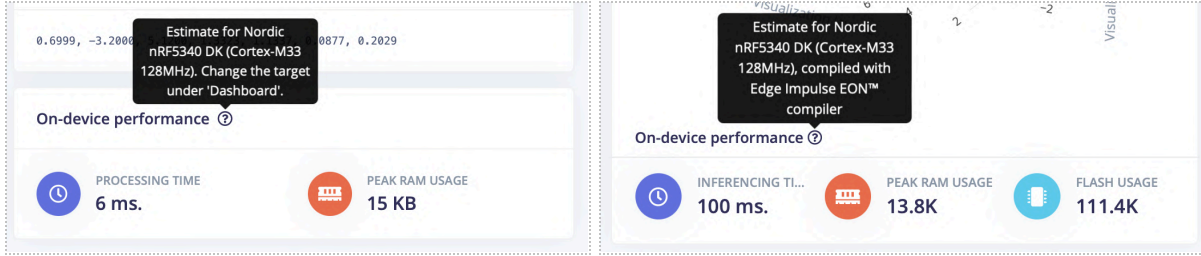


Figure 3a: Performance metrics estimation for DSP algorithm in a processing block
 Figure 3b: Performance metrics estimation for ML model in a learning block

Data

The process of creating a machine learning model begins with data. The data may originate from **various devices or other sources** (prototype devices being developed vs industrial-grade reference devices), have **different formats** (excel sheets, images, CSV, JSON, etc...), and are stored in various places (researchers' computers, Dropbox folders, Google Cloud Storage, S3 buckets, etc...).

Data in Edge Impulse can exist in two places: **Organization** and **Project**.

Edge Impulse's **Organizational Data** component (part of the data acquisition pipeline) is designed to import data from various sources and plays a role similar to the feature store of an organization. Data can be organized into different datasets that can be reused across various projects and transformation experiments. This facilitates the collection of diverse, real-world data (an in-house digital asset for your organization) to train robust models.

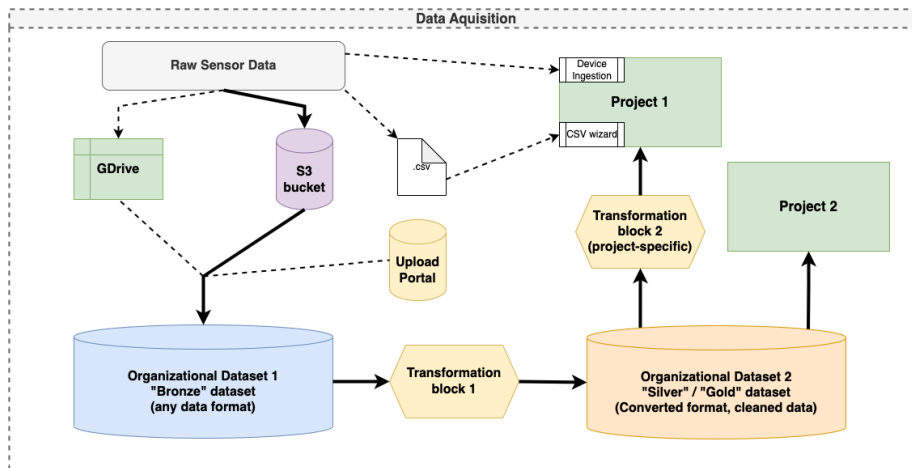


Figure 4: Diagram of data flow paths from device through the Edge Impulse platform

Additionally, users can collect and store training and test data directly in a **Project**, alongside their model and deployment code. Rather than relying on prebuilt datasets or requiring users to construct their own data-gathering technology, Edge Impulse offers a variety of data [ingestion methods](#).

Organizational data should be the default way to start an Edge ML project. However, for quicker experiments, it is okay to use project data directly.

Figure 4 illustrates an example setup of dataflow in an organization, showcasing different ways the data can be imported into the project. Data features are described in more detail in the section [Data](#).

About the Dataset

Building a dataset is a key part of the ML process. One of the key values that Edge Impulse provides are its straightforward yet powerful tools for data collection, data labeling, and transformation. Please consult our docs outline [how to begin creating your own datasets in Edge Impulse](#).

In this guide, we will showcase the features of Edge Impulse by using an open dataset for **sensorless AC motor drive diagnostics**. The dataset can be accessed [here](#), as well as from the [GitHub repository](#) accompanying this guide.

This dataset consists of samples of measured electric current during the operation of a sensorless AC motor drive. Samples are recorded for 11 sessions of motor operation. In each session, the drive had various intact and defective components. This results in 11 different sample classes, where each sample belongs to one of 11 classes.

Each class consists of samples gathered under different operating speeds, load moments, and load forces. The current signals are measured with a current probe and an oscilloscope on two phases.

The dataset is presented as a directory with **11 subdirectories**, each corresponding to **one condition class**. For each class, there are eight samples or time series recordings of two axes of alternating current (AC). Each of these recordings is a 9000ms array of time series AC measurements sampled at 100000 Hz. The recordings are in a generic **.txt** format that is often used when collecting data in industrial settings. Subsection [Pre-Processing and Transformation](#) shows a



process of converting this format to CSV, which is more commonly used for time series data with ML tasks.

Throughout this guide, this dataset will be used to build a machine learning pipeline to classify the motor failure based on the AC data. It can then be deployed to the edge device — for this application, it could be an MCU that is attached to the motor drive and has an AC sensor as an input.

Setting Up an Organization Dataset

An organization dataset can be imported from several locations, including an AWS S3 bucket/Google Cloud Storage that might contain raw data.

External S3 Bucket

To link an S3 bucket to an organization, select the “**Data**” tab in the organization view, select “**Buckets**” view, and click “+ Add new bucket” in the top right corner.

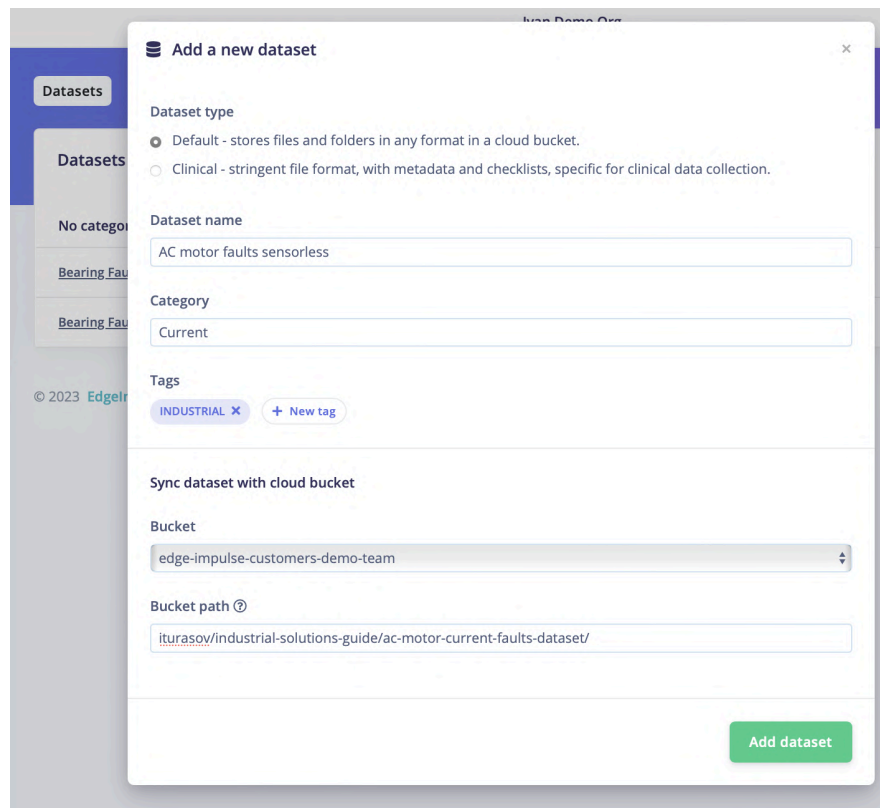
Fill in the bucket details. E.g., s3.eu-west-1.amazonaws.com/my-raw-data-bucket.

The screenshot shows the 'Add a new storage bucket' form in the Edge Impulse interface. The form is titled 'Add a new storage bucket' and is open in the 'Buckets' view of the 'Ivan Demo Org' organization. The form asks 'Where is your bucket hosted?' with a dropdown menu set to 'Amazon S3'. Below this, there are fields for 'Amazon S3 access credentials': 'Bucket' (my-raw-data-bucket), 'Region' (eu-west-1), 'Access key' (ACCESS-KEY), 'Endpoint' (https://s3.eu-west-1.amazonaws.com), 'Secret key' (masked with dots), and 'Prefix to check connectivity (optional)' (path/that/you/have/access/to). A green 'Add bucket' button is at the bottom right.

Figure 5: S3 bucket setup form

Then fill in “my-raw-data-bucket” in the “**Bucket**” field, and “eu-west-1” in “**Region.**” Fill in the bucket access key and secret key, and press “**Add bucket.**”

Now, when the S3 bucket is connected, it can be used to create an **Organization Dataset** from one of the directories that the bucket has.



The screenshot shows a modal window titled "Add a new dataset" with the following fields and options:

- Dataset type:** Radio buttons for "Default - stores files and folders in any format in a cloud bucket." (selected) and "Clinical - stringent file format, with metadata and checklists, specific for clinical data collection."
- Dataset name:** Text input field containing "AC motor faults sensorless".
- Category:** Text input field containing "Current".
- Tags:** A tag labeled "INDUSTRIAL" with a close button (x) and a "+ New tag" button.
- Sync dataset with cloud bucket:** A checked checkbox.
- Bucket:** A dropdown menu showing "edge-impulse-customers-demo-team".
- Bucket path:** Text input field containing "iturasov/industrial-solutions-guide/ac-motor-current-faults-dataset/".
- Buttons:** A green "Add dataset" button at the bottom right.

Figure 6: Organization Dataset setup form

Locate the “**Datasets**” view and click “**+ Add new dataset.**” Select “clinical” type (this dataset type works well with automated pipelines covered later in this guide), give the dataset a name, and provide the bucket path where the data is located. The bucket can have several folders, each containing a different dataset, and can be mapped to the Edge Impulse datasets of the bucket structure.

Click “**Add dataset.**” After the job is complete, a dataset explorer will appear that contains the same file structure as the directory previously linked to the bucket. Now, this dataset can be used to clean, transform, and import data across the Edge Impulse platform.

The status of the buckets' connections can always be checked with a health indicator, as shown by the green indicator on the "Buckets" tab.

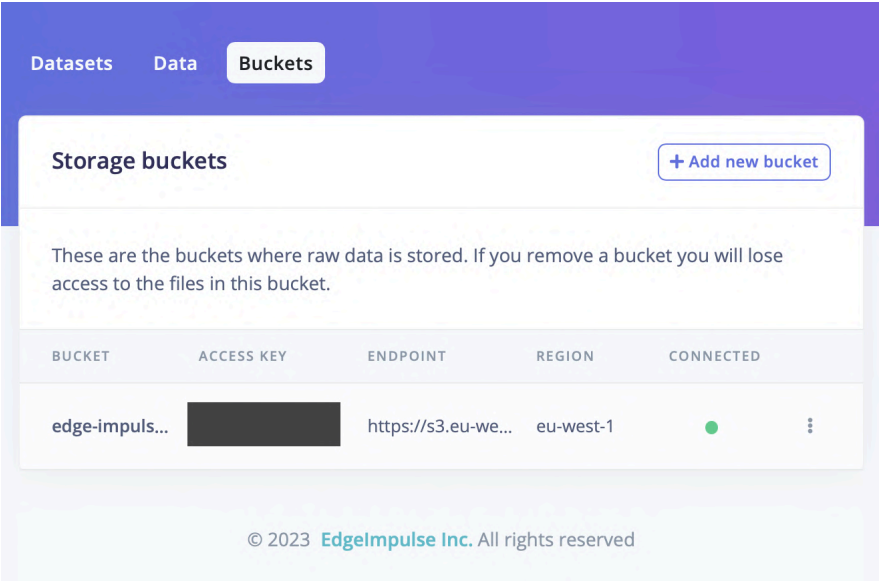


Figure 7: Storage buckets list after a bucket is successfully added

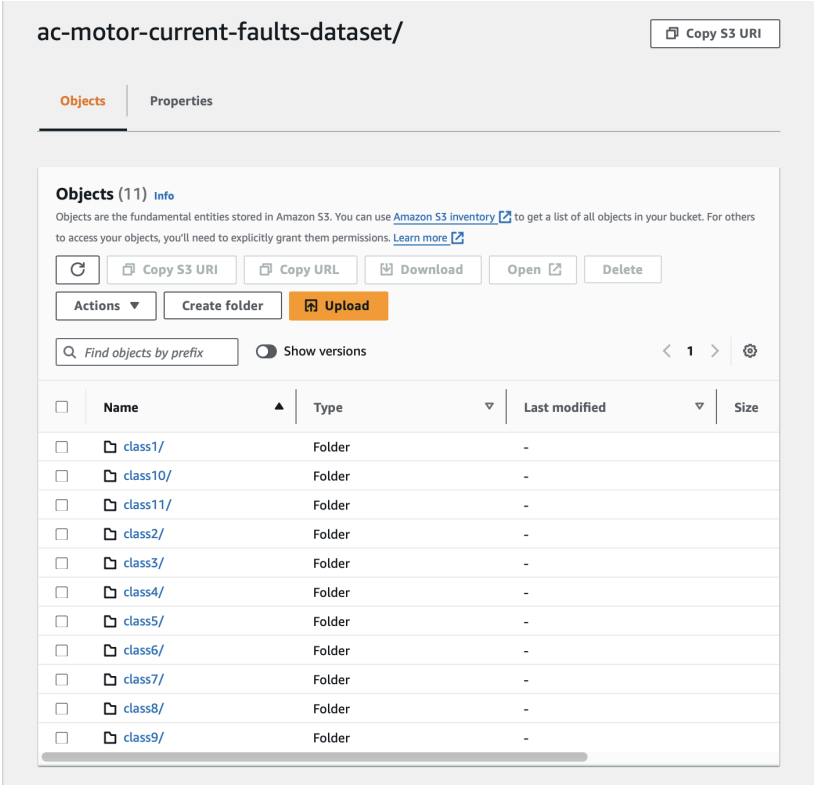


Figure 8: Directory structure in the AWS S3 bucket

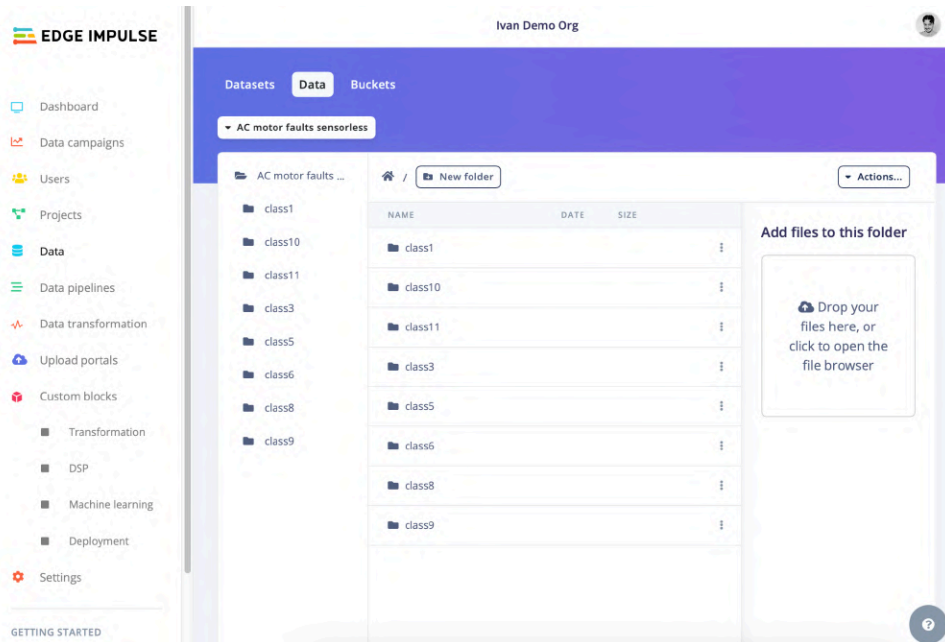


Figure 9: Directory structure in an AWS bucket, connected to Edge Impulse

Upload Portal

An [Upload Portal](#) is a secure way to allow external parties to upload data to your datasets. It provides an easy user interface for adding data without giving access to the content of the dataset or the ability to delete any files. Data uploaded through the portal can be stored on-premise or in your own cloud infrastructure. Upload portals are particularly useful for collecting data from external sources directly into your storage buckets, facilitating the use of this data in Edge Impulse for further processing.

Importing Data into the Project

Sometimes it is necessary to add test data to only the current project being worked on without going through a feature store and transformation pipeline. This may be useful to quickly test a small subset of samples to test some initial hypothesis. Below are the options to achieve this.

CSV Wizard

The most common format for importing time-series data is CSV. In case the samples stored on the machine are already in CSV format, a template can be defined by which all the rows will be imported by using one representative sample and setting up the column semantics. This feature is called [CSV Wizard](#) and can be accessed from the data upload page of a project.

Direct Ingestion from Device

Two most common ways to upload data to the Edge Impulse platform directly from an edge target device are:

1. Using the [Data Forwarder](#): This method is easy to use if the device can output the data it samples to the serial output. Connect the Edge Impulse CLI tool to the serial port where the computer is receiving the device output, and the samples will be forwarded to the Edge Impulse ingestion service and arrive in the project.
2. Using the [Edge Impulse C ingestion SDK](#): This method enables the user to program the device firmware to send the samples to the Edge Impulse ingestion service itself. This is an advanced method that is applicable if the device can directly connect to the internet, but it can also unlock use cases such as active learning to continue collecting raw data samples from devices that are deployed in the field.

Pre-Processing and Transformation

[Transformation blocks](#) are a very flexible tool that can be leveraged as part of the organizational features that the Edge Impulse platform offers. They can be used for most advanced data transformation use cases. Transformation Blocks can take raw data from the organizational datasets and convert the data into files that can be loaded into an Edge Impulse project/another organizational dataset.

Transformation blocks can be used as part of automated **organization pipelines** and **project pipelines** to automate the processes. Transformation blocks can fetch external datasets, augment/create variants of the data samples, extract metadata from config files, create helper graphs, align and interpolate measurements across sensors, remove duplicate entries, and more.

Transformation blocks can be written in **any programming language** that can be executed in a containerized environment, such as Docker. When deployed, they run on the Edge Impulse platform infrastructure. The most common language used for building a transformation block is **Python**.



Here is an example of a transformation block used to transform the sensorless AC dataset to a JSON format that the Edge Impulse project can work with. The [Edge Impulse CLI](#) tools are required to set up and upload the transformation block to the organization.

```

industrial-solution-guide/transform-ac-motor-fault-detection-data on main [?] v
ia 3.10.9 on v20.10.22 via industrial-solution-guide via base took 36.3s
→ edge-impulse-blocks init
Edge Impulse Blocks v1.22.0
(node:7098) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Pl
ease use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
? In which organization do you want to create this block? Ivan Demo Org
Attaching block to organization 'Ivan Demo Org'
? Choose a type of block Transformation block
? Enter the name of your block ac-sensorless-txt-to-ei-json
? Enter the description of your block This block takes the files in the format pu
blished in ine "Dataset for Sensorless Drive Diagnosis" and transforms it into EI
json format
? What type of data does this block operate on? Data item (--in-directory passed
into the block)
? Which buckets do you want to mount into this block (will be mounted under /mnt/
s3fs/BUCKET_NAME, you can change these mount points in the Studio)?
Your new block 'ac-sensorless-txt-to-ei-json' has been created in '/Users/ivan/ei
-solutions/industrial-solution-guide/transform-ac-motor-fault-detection-data'.
When you have finished building your transform block, run 'edge-impulse-blocks pu
sh' to update the block in Edge Impulse.

```

Figure 10: Uploading a transformation block through Edge Impulse CLI

The source files for this custom block are located in the [repository](#) accompanying this guide.

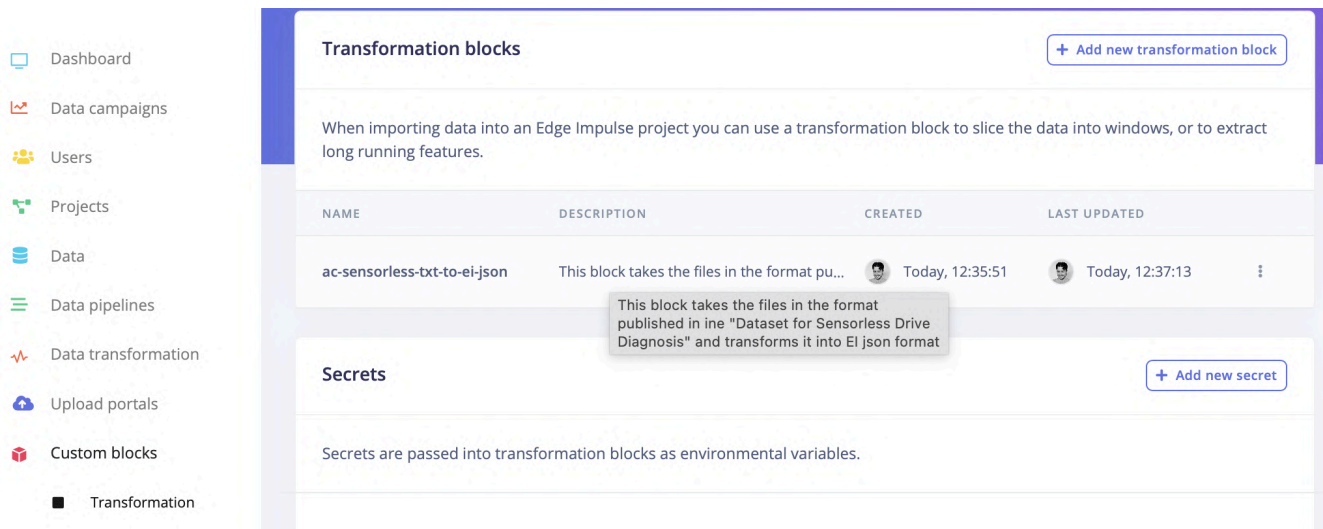


Figure 11: Custom blocks list in organization after adding a new block

DSP — Feature Engineering at the Edge

Digital signal processing (DSP) is the practice of using algorithms to manipulate streams of sensor data. When paired with embedded machine learning, it is common to use DSP to extract, modify, or generate signals before feeding them into machine learning models.

A few reasons to apply DSP are:

- Cleaning up a noisy signal
- Removing spikes or outlying values that might be caused by hardware issues
- Extracting the most important information from a signal
- Transforming the data from the time domain to the frequency domain

In Edge Impulse, DSP pre-processing is added to the impulse using the **Processing Block**. It can be configured to use one of many algorithms that Edge Impulse has already implemented, and each of them can fit a specific purpose better. For example:

- **Flatten:** This is the simplest block that extracts statistical features from a time-series sample, such as Max, Min, Standard Deviation, RMS, etc.
- **Spectral Features:** This block extracts frequency, power, and other characteristics of a signal. Low-pass and high-pass filters can also be applied to filter out unwanted frequencies. It is great for analyzing repetitive patterns in a signal, such as movements or vibrations from an accelerometer.

Multiple DSP blocks can be selected for one impulse — in which case raw data will be passed independently through all of them, and their outputs will be combined to serve as inputs to the machine learning block.

In case there exists a DSP algorithm that is specific to some sensor data and use case, and it is not present in Edge Impulse, it is possible to create a [custom processing block](#) with the code of a custom algorithm and use it as part of the impulse. The Edge Impulse team is available to assist in this process.



⚡ Add a processing block ×

Did you know? You can [bring your own DSP code](#).

DESCRIPTION	AUTHOR	RECOMMENDED
Flatten Flatten an axis into a single value, useful for slow-moving averages like temperature data, in combination with other blocks.	Edge Impulse	Add
Image Preprocess and normalize image data, and optionally reduce the color depth.	Edge Impulse	Add
Audio (MFCC) Extracts features from audio signals using Mel Frequency Cepstral Coefficients, great for human voice.	Edge Impulse	Add
Audio (MFE) Extracts a spectrogram from audio signals using Mel-filterbank energy features, great for non-voice audio.	Edge Impulse	Add
Spectral Analysis Great for analyzing repetitive motion, such as data from accelerometers. Extracts the frequency and power characteristics of a signal over time.	Edge Impulse	Add
Spectrogram Extracts a spectrogram from audio or sensor data, great for non-voice audio or data with continuous frequencies.	Edge Impulse	Add

Figure 12: Part of a list of pre-made processing blocks

Machine Learning Model

The machine learning model takes features generated by the processing block as an input and outputs a result. The type of result depends on the problem to solve and the application being built. Some examples of machine learning algorithm types include:

- **Classification:** Classification algorithms try to solve the problem of distinguishing between various *types*, or *classes*, of things. This could mean, based on microphone input, determining if the sound is coming from a fan, a chainsaw, or a pump. Classification models output a score from 0 to 1 that represents how confident the model is that a given sample belongs to each of the classes.
- **Regression:** Regression algorithms predict numerical values based on input features. Common use cases include estimating temperature based on historical data or estimating the motor speed based on the video feed of the motor spinning.

- **Object detection:** This set of algorithms locates the objects of interest on the provided images. One common output format is bounding boxes — the location and size of a “box” where the object is located on the image, with a confidence score for each bounding box as in classification models.

A machine learning model goes through two phases of lifecycle: **training and inference**. Training is a computationally intensive task — e.g., a convolutional neural network model is shown labeled data, and its weights are adjusted in a process called “backpropagation.” This is repeated many times until the model learns to get the right label accurate enough. Due to the intense computing power and infrastructure required, model training is performed in the cloud through the Edge Impulse platform.

Next, the model will be deployed. After that, it will be working in “forward pass” mode — in a process called “**inference**.” An optimized inference engine and a trained model are what gets deployed to the edge device.

Training

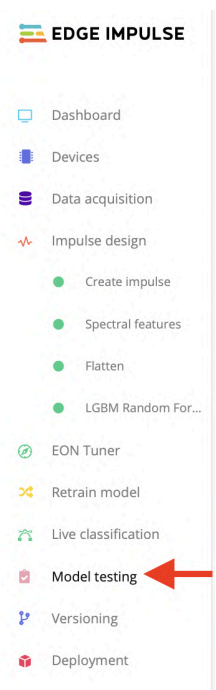
Model training is the process where the machine learning model learns to recognize patterns, correlations, and relationships in the input data.

Edge Impulse model training happens in the Learning Block included in the impulse. A [learning block](#) is the step of the pipeline that describes the model architecture and training parameters at train time (in the cloud) and performs model inference at inference time (on the target device or gateway). You can use a pre-existing model architecture (e.g. YOLOv5 or ResNet for object detection), or build your own [custom learning block](#).

Testing and Evaluation

After the model is trained, it can be tested in the Edge Impulse platform. Testing means running an inference over a set of samples that were not used during the training and evaluating the performance metrics.

Model testing can be accessed from any project through a dedicated tab in the left-side menu.



Automated Machine Learning with EON Tuner

The **EON Tuner** is Edge Impulse's AutoML (automated machine learning) tool designed to find and select the best embedded machine learning model for a given application within the constraints of your target device. It performs end-to-end optimization of the combination of a DSP algorithm and a machine learning model, finding the **ideal trade-off** between these two blocks to achieve optimal performance on the given target hardware.

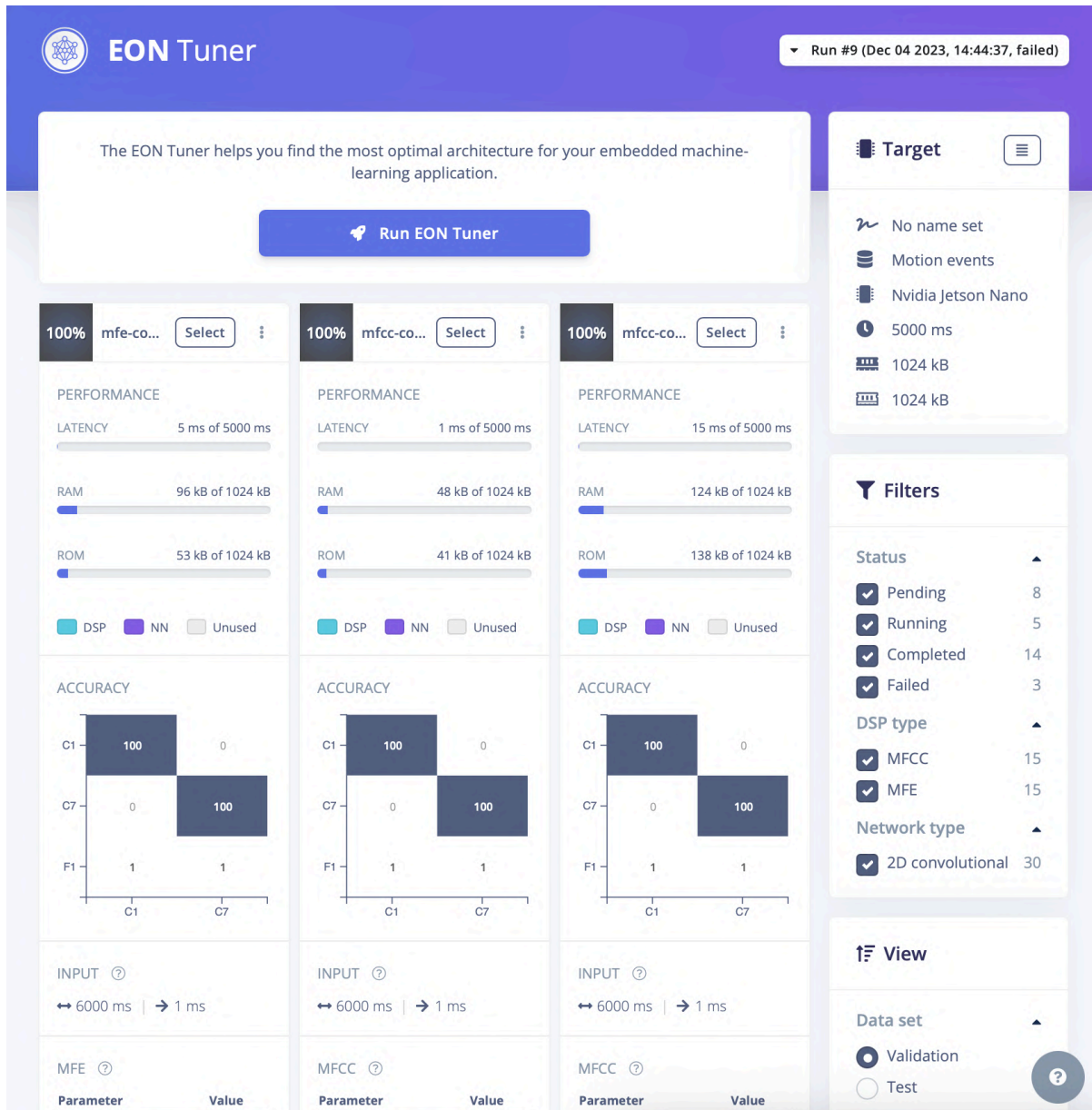
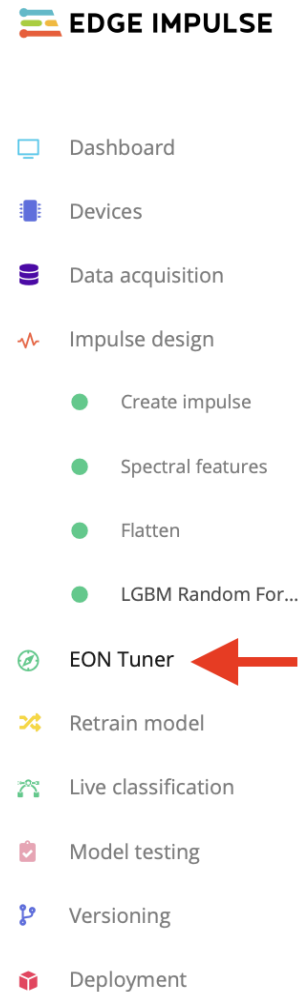


Figure 13: EON Tuner's best suggested impulse configurations after a run completion

The advantages of using the EON Tuner include:

1. **Optimization for Target Devices:** It analyzes performance directly on any device fully supported by Edge Impulse, allowing for optimizations specific to the device's hardware capabilities.
2. **Support for Various Task Categories:** The tuner supports different types of sensor data, including motion, images, and audio, optimizing for common applications or task categories within these data types.
3. **Comprehensive Evaluation:** It evaluates different configurations for creating samples from your dataset, tests various parameters and configurations of processing blocks, and evaluates different model architectures, hyper-parameters, and data augmentation techniques.
4. **Flexibility in Configuration:** Users can define the EON Tuner Search Space to constrain the tuner to use steps defined by hardware, customer requirements, or internal knowledge, offering flexibility in meeting specific project needs.



EON Tuner can be accessed from any project from the left side menu.

Creating the Impulse — Step-by-Step Guide

With the knowledge about the data and all the building blocks of the impulse and the project, let's create an ML pipeline that can be deployed to an edge device.

Step 0: Import Data to Your Project

As mentioned in section “[Data](#),” there are several ways to import the data into the project. To import the “.txt” formatted AC motor dataset, the transformation block created in section “[Pre-Processing and Transformation](#)” will be used.

Navigate to the organization page and select the “**Data Transformation**” page.

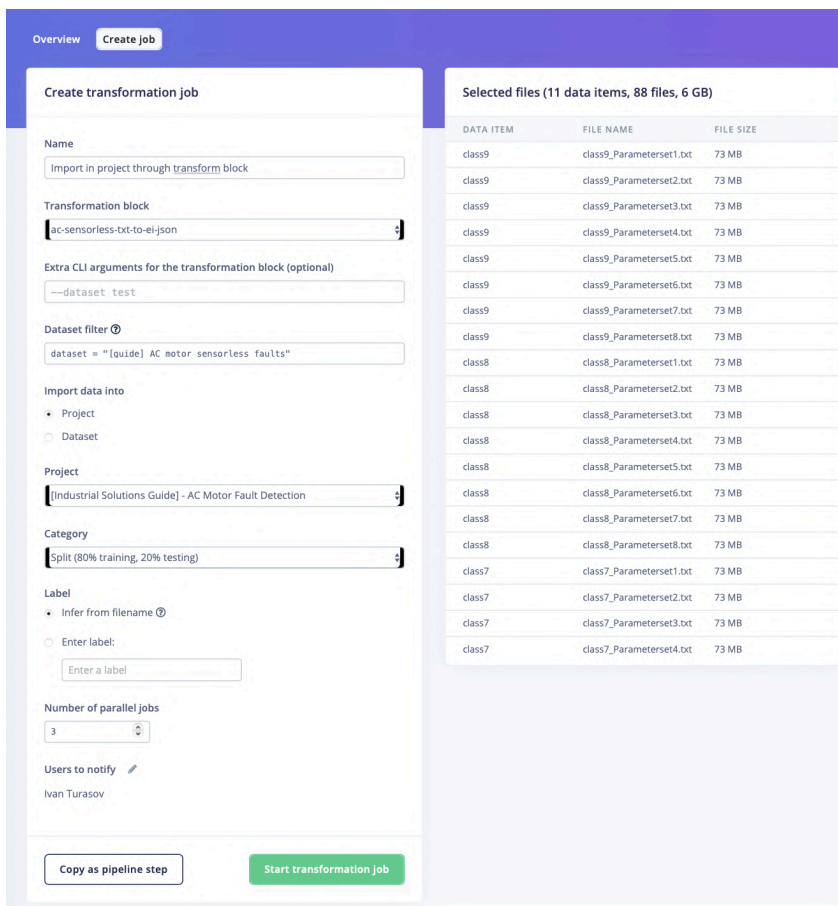


Figure 14: Configuring a transformation job to import the data from an organization dataset into the project

Figure 14 illustrates the configuration options for this step — select the transformation block created earlier, and the project to import the data to. Press **“Start transformation job.”** Once the job is complete, navigate to the project data page — the samples should be imported.

Step 1: Select All the Blocks That Will be Part of Your Impulse

Once the project has data, it’s time to select the blocks that the impulse will consist of. **Figure 15** illustrates what the impulse will look like.

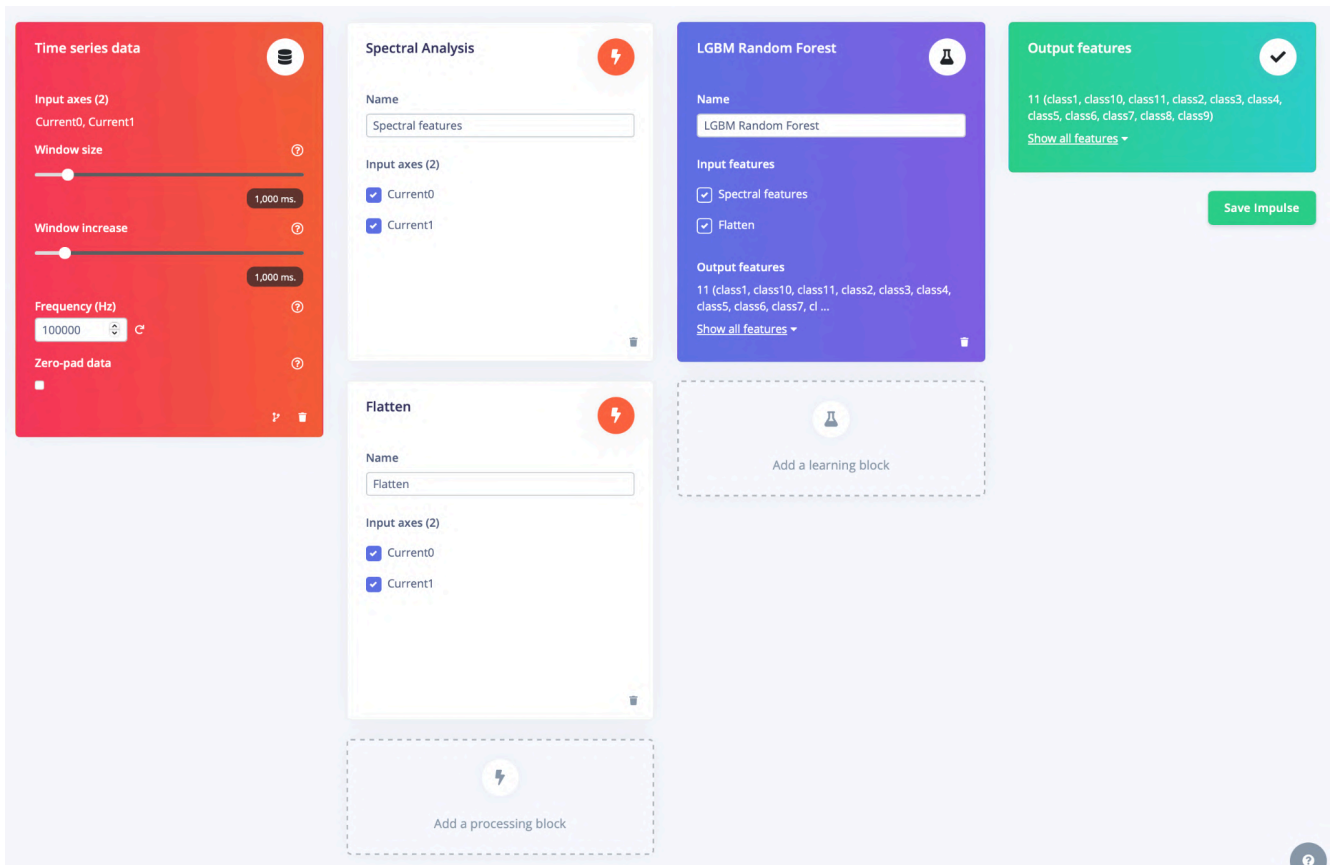


Figure 15: Configured Impulse view

Processing Block

Press **“Add a processing block”** to open a list of available processing blocks. For this project, we will select two blocks, so first add a **“Spectral Analysis”** block, then press **“Add a processing block”** once again and add a **“Flatten”** block. The list of input axes for each of the processing blocks corresponds to the format of the data that is in the project dataset — as mentioned before, each sample contains two axes of AC.



Learning Block

Press “**Add a learning block**” to open a list of available processing blocks. For this project, we will use a **LightGBM** technique (short for **Light Gradient-Boosting Machine**) — a highly efficient gradient boosting algorithm that typically works well with low dimensional inputs and is more efficient than deep learning when working with DSP blocks like Flatten and Spectral Features.

Step 2: Generate Features

Both of the selected processing blocks can be configured using relevant parameters, after which feature generation will happen. This means that all the raw data samples in the training and testing sets will be put through the selected processing algorithms, and the generated features will represent these samples as inputs to the ML algorithm. In this case, LightGBM.

Spectral Features

Navigate to the “**Spectral features**” tab in the Impulse design section of the project menu.

- Impulse design
 - Create impulse
 - Spectral features
 - Flatten
 - LGBM Random For...

There are a lot of parameters that this processing block takes. The [DSP Autotuner](#) makes it easy to automatically fine tune the processing block parameters. With one click of a button, the autotuner looks at the entire dataset and recommends a set of parameters tuned to make the most out of the dataset.

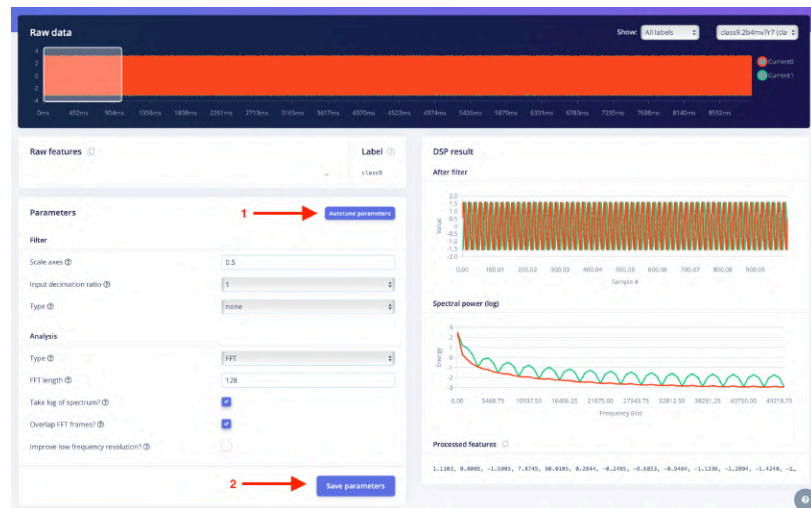


Figure 16: "Spectral features" processing block page

Click “**Autotune parameters**” to start the autotuner job. After it’s completed, press “**Save parameters**” to advance to a feature generation screen.

Press “**Generate features**” to initiate the feature generation job.

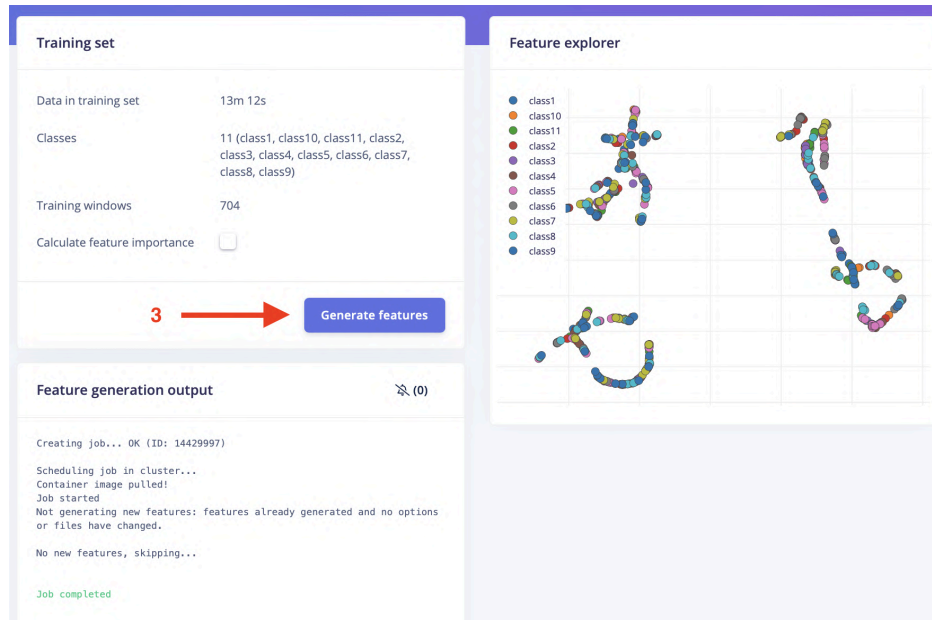







Figure 17: View of features generated from the “Spectral features” block

Once the job is complete, notice the message “**Job completed**” in the job log alongside the feature explorer. This visualization illustrates how the samples are represented in the feature space of the generated parameters.

Flatten

-  Impulse design
 -  Create impulse
 -  Spectral features
 -  Flatten
 -  LGBM Random For...

Navigate to the “**Flatten**” tab in the Impulse design section of the project menu.

Looking at the raw data window, this block generates seven features that are statistical measures of the signal, namely: Average, Minimum, Maximum, RMS, Standard Deviation, and Kurtosis.

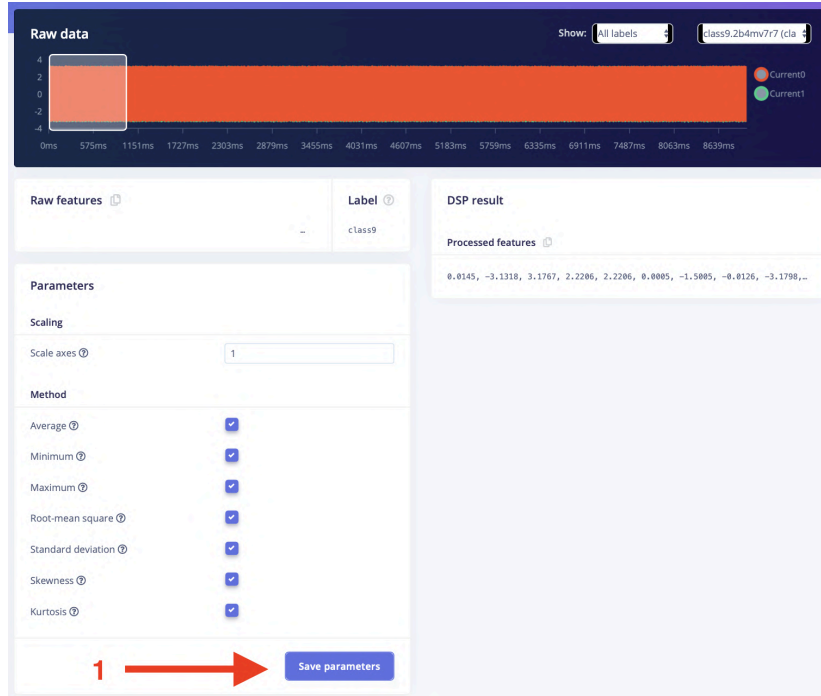


Figure 18: "Flatten" processing block page

There are no parameters for these features except for the set to be selected hence there is no **Autotuner** option. Select all of the features, then press "**Save parameters**" to advance to a feature generation screen.

Press "**Generate features**" to kick off the feature generation job.

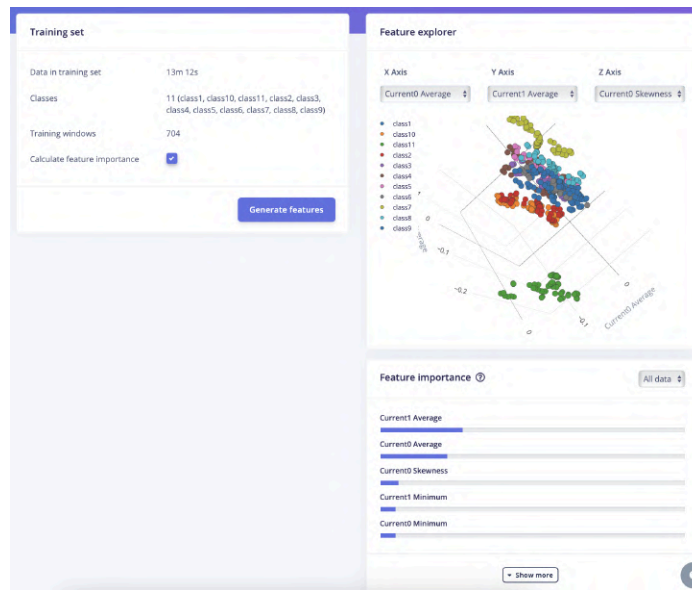







Figure 19: View of features generated from the "Flatten" block

When the job is complete, you will see the message “**Job completed**” in the job log, alongside a feature explorer — a visualization that illustrates how the samples are represented in the feature space of the generated parameters

Step 3: Train the Machine Learning Model

After the features are generated, they are ready to be used for model training. Navigate to the “**LGBM Random Forest**” tab. You’ll be presented with a screen with model hyperparameters that will be used for training.

It is a good rule of thumb to start with default parameters and train the model to get a baseline performance. After that, parameters like the number of training iterations can be adjusted. EON Tuner can be used for that, as described in a section above ([Tuning with EON tuner](#)).

-  Impulse design
-  Create impulse
-  Spectral features
-  Flatten
-  LGBM Random For...

Press “**Start training.**” The training log will appear on the left, and once it’s complete, the model’s accuracy against a validation set and a confusion matrix will be displayed. In this case, the model achieved **95.6% accuracy** across the whole validation set. Some classes are recognized better than others, which is reflected in different accuracies per class in the confusion matrix.

This is the first measure of the model performance that can be considered. Already now it’s possible to reason about how to improve the model performance — for example, if one class performs much worse than all the others, it might mean that it’s **underrepresented** in the training set. In this case, it might be a good idea to collect more samples of this class to **balance the dataset** better.

The next step is model testing.

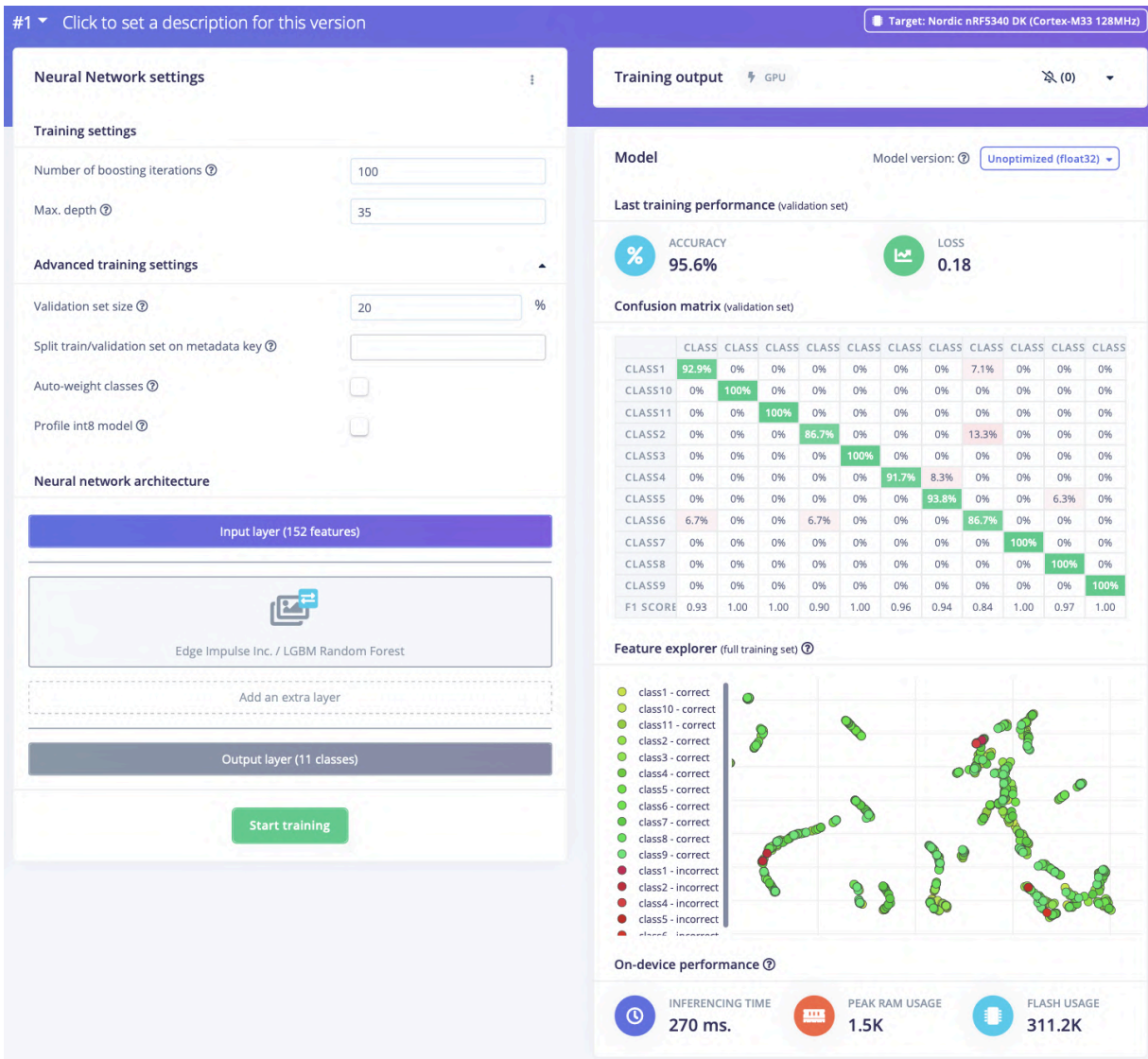


Figure 20: View of "LGBM" learn block page after the model training is completed

Step 4: Test the Machine Learning Model

To test the model, navigate to the **"Model testing"** tab. It will present a set of samples from the original dataset that was set aside and not involved with the training — this can offer a sense of how the model will perform once deployed. After training the model, new samples can be continuously added to the training set. For instance, if additional experimental runs are conducted on a test device and the data is uploaded directly to the training set, those samples will promptly appear on this page. Subsequently, the trained model can be tested against these newly added samples.

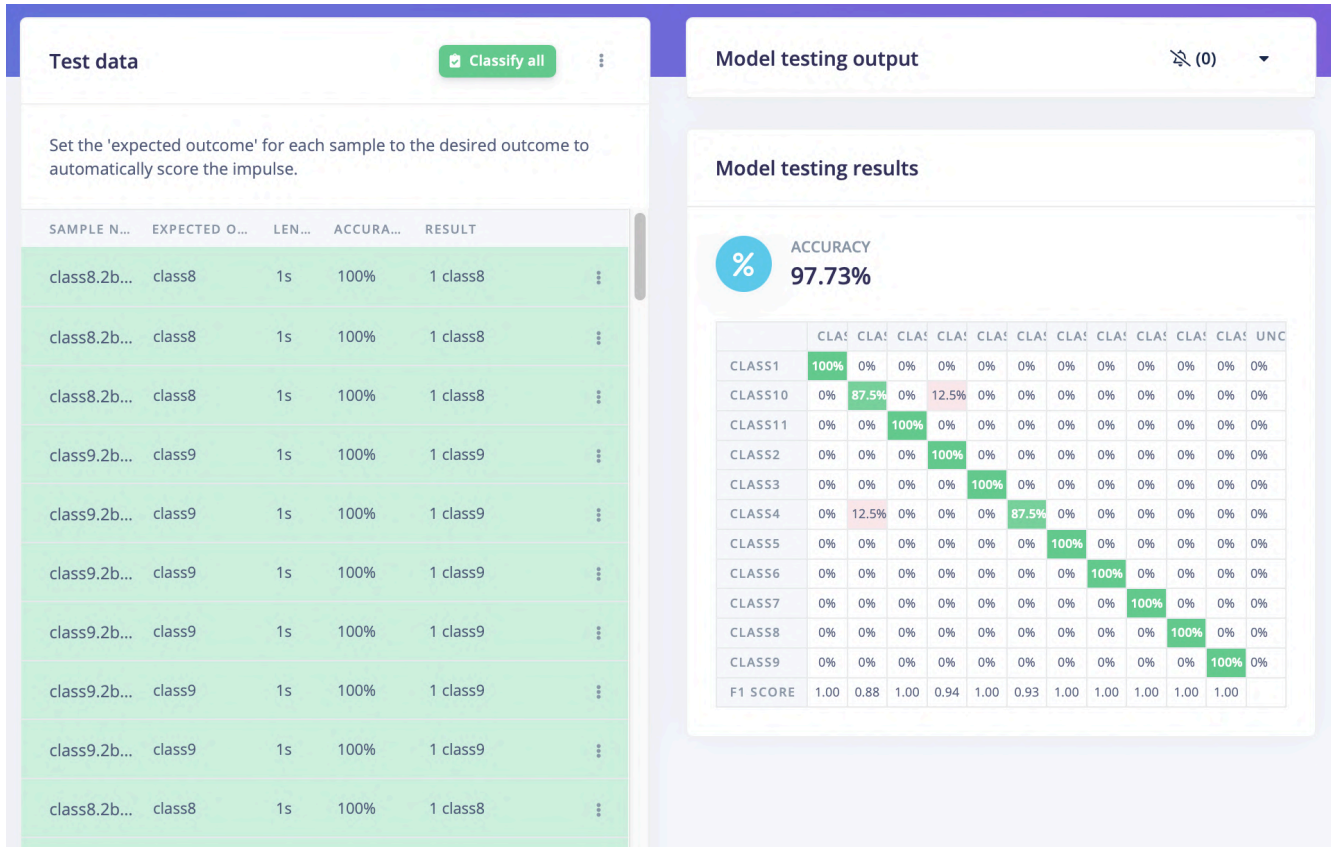


Figure 21: View of "Model testing" page after the testing is completed

Deploying the Impulse — Putting it All to the Edge

After the impulse is created and the model performance is satisfactory — it is time to deploy it to the device.

There are numerous deployment options, depending on the goal and target. The most flexible one is a **C++ library**.

This option is available in the dropdown menu in the "Deployment" section of the project.

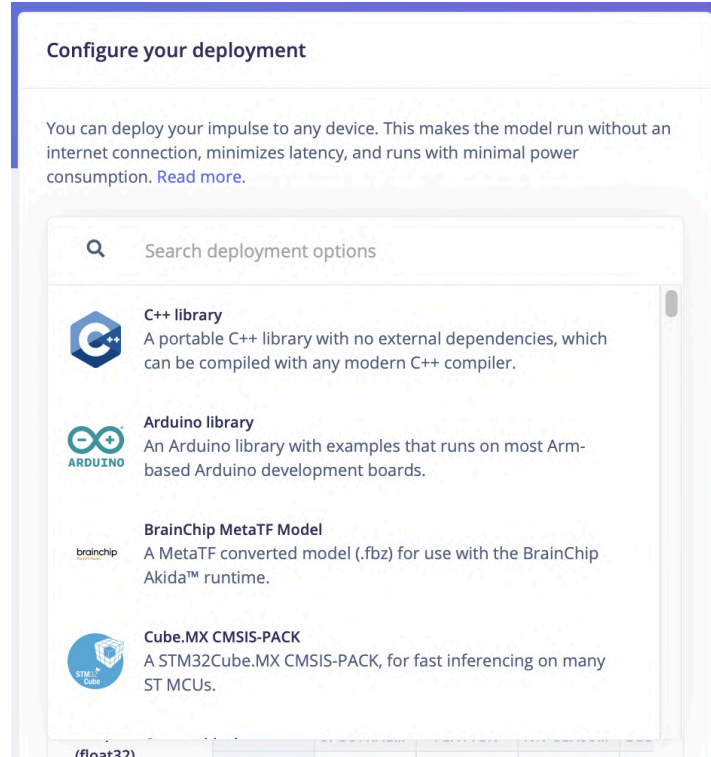


Figure 22: Deployment options dropdown at "Deployment" page of the project

After selecting it, clicking “**Build**” will start the library generation process. The Edge Impulse platform will generate an archive that contains our C++ SDK and configuration of the impulse, including a highly optimized model.

This library can be included directly in a firmware project (e.g., Zephyr, plain Linux, FreeRTOS, etc.). The SDK comes in the form of **non-compiled source code** and is available to reuse and adjust **all parts** in the firmware project as needed.

To make things easy, Edge Impulse offers a set of open-source model **firmware projects** for several target systems, including but not limited to [Linux](#) and [Zephyr](#).

Prerequisites

The following sections describe how to deploy an impulse to a Linux machine using the [example-standalone-inferencing-linux](#) project. This project can be used to compile for any Linux-based target, including a simple computer. The easiest way to follow the steps is from a Linux machine.

However, as described above, the impulse can be deployed to **numerous other target devices, including MCUs that Zephyr RTOS supports.**

To test the impulse on such a device (e.g., **Nordic nRF5340 DK**), use the [example-standalone-inferencing-zephyr](#) repository and follow the “[Running your impulse locally \(Zephyr\)](#)” guide in the Edge Impulse documentation portal.

The same model files that are acquired from the platform using the “C++ library” deployment option are used to build any target project (i.e., **model-parameters**, **tflite-model**, and **edge-impulse-sdk** folders — described further in this section).

EON Compiler

The **Edge Optimized Neural (EON) compiler** is a powerful tool developed by Edge Impulse designed to optimize and effectively run neural networks with reduced RAM and flash usage, all while maintaining accuracy comparable to TensorFlow Lite for Microcontrollers. The [EON Compiler](#) incorporates a proprietary compiler that compiles and optimizes neural networks to C++, eliminating complex code, significantly reducing device resource utilization, and saving inference time.

Key Benefits of Enabling EON Compiler:

- 25–55% less RAM
- 35% less flash
- Same accuracy as TFLite
- Faster inference

Download the C++ library and prepare the application

First, **clone the [standalone Linux repository](#)**. The application repository contains all the necessary infrastructure to build and run an application on a Linux machine, namely five applications: *audio.cpp*, *camera.cpp*, *collect.cpp*, *eim.cpp*, and *custom.cpp*. A more detailed description of each of them can be found in [README.md](#) — for now, we will focus on the **custom.cpp**.



This code contains the logic to read an input file — for example containing the raw features as they come from a sensor. This example code is useful to quickly test what kind of results the model will generate on the device for any given input.

Next, unzip the archive generated after the build is complete — it should contain the following three folders: **model-parameters**, **tflite-model**, and **edge-impulse-sdk**.

Place these folders in the root of the repository. The file structure should look the following way:

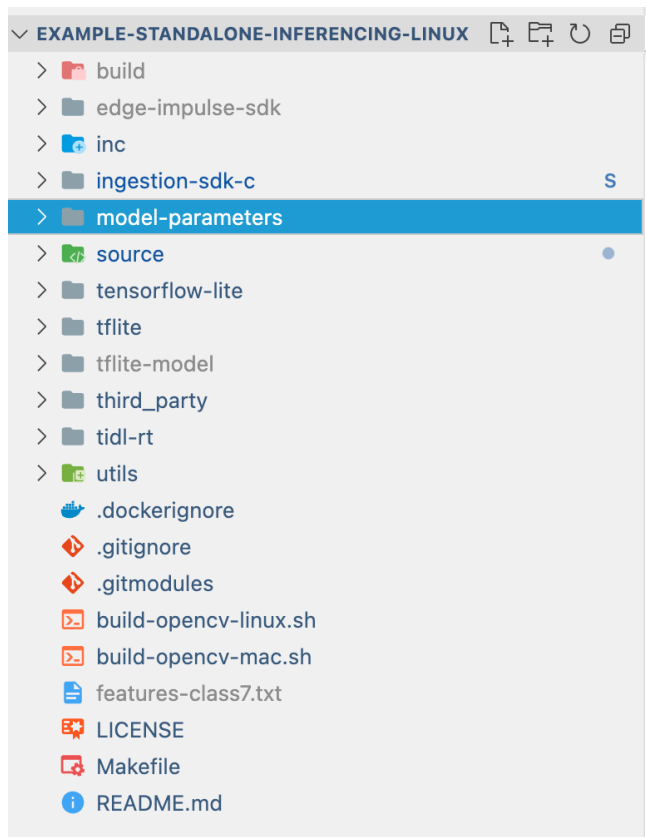


Figure 23: Directory structure of the example-standalone repository after adding C++ library acquired from the platform deployment

Every time it is necessary to test another model or a new version of the same model downloaded from the platform, delete these three folders and replace them with the new ones.

Build and Test the Application

After everything is in place, it is time to build the project. There are numerous architectures available to build the application. Detailed build instructions with all the parameters are described in the repository [README.md](#). For standard desktop Linux, a simple command can be used, namely:

```
example-standalone-inferencing-linux on ? master [!?]
→ APP_CUSTOM=1 make -j `nproc` █
```

This will create a binary executable “./build/custom” that contains the necessary parts from our SDK as well as the code for the DSP blocks and the model created in the platform. Additional build options and flags are described in the repository README.

To **test the inference**, we need to have a sample of raw data. Navigate to the project page in the platform, open the page of either of the DSP blocks, and select some sample on the top right, for example, a sample from Class 7.

Click the “**Copy**” icon next to the “**Raw features**” pane. Next, in the root of the app repository, create a file called “features-class7.txt” and paste the copied 199800 floating point numbers in that file.

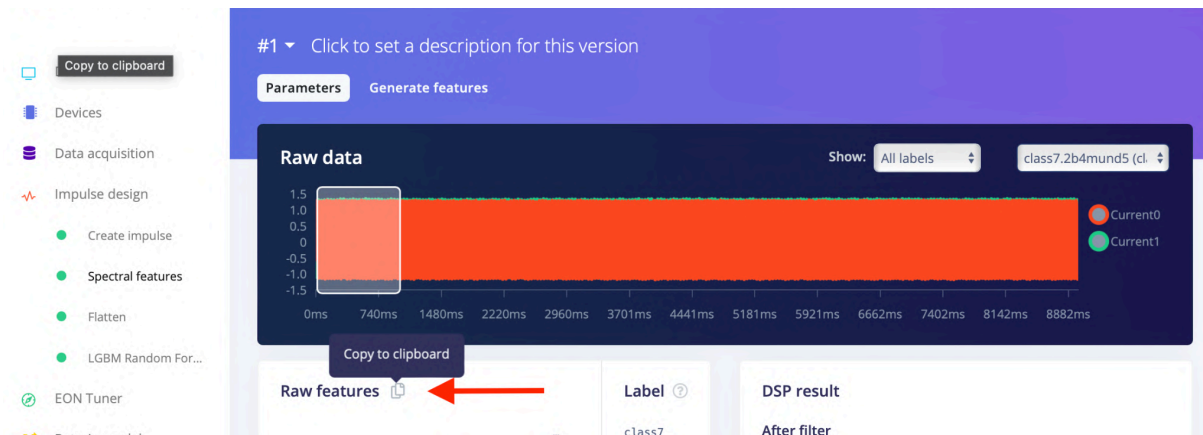


Figure 24: Acquiring raw features of a sample from the “Spectral features” processing block

From the root of the repository, run the compiled binary, passing the .txt file as input, as shown below. Now the resulting **pre-processed features array** (outputs of the DSP block) and the **model's confidence for each class**, alongside **time measurements**, are visible for each stage of the impulse:

```
example-standalone-inferencing-linux on | master [!?] via | Cbase
→ build/custom features-class7.txt
Features (8 ms.): 0.998265 0.007953 -1.499700 7.771976 58.886593 1.383018 -0.405614 0.
391694 -0.636127 -0.121417 -0.669746 -0.526612 -0.709357 -0.733772 -0.762411 -0.791245
-0.824493 -0.861216 -0.901668 -0.947285 -0.996059 -1.050112 -1.109890 -1.175086 -1.24
9730 -1.321470 -1.412610 -1.507216 -1.620430 -1.580738 -1.890473 -1.523187 -2.266463 -
1.503542 -2.686627 -1.516952 -2.731098 -1.559850 -2.770667 -1.631899 -2.436920 -1.7381
95 -2.148041 -1.884124 -1.969891 -1.911861 -1.856735 -1.818225 -1.789497 -1.770461 -1.
759593 -1.756564 -1.761426 -1.774501 -1.795380 -1.823284 -1.860869 -1.906858 -1.963088
-2.033115 -2.112726 -2.046961 -2.330124 -1.938513 -2.657937 -1.872221 -3.029419 -1.84
0387 -3.031240 0.997511 0.004379 -1.500359 7.873979 60.005020 0.362070 -0.419784 -0.65
4157 -0.957128 -1.099327 -1.205390 -1.292397 -1.359538 -1.419788 -1.475357 -1.525527 -
1.573317 -1.620576 -1.669361 -1.719059 -1.769460 -1.824118 -1.880286 -1.940517 -2.0234
40 -2.048874 -2.122470 -2.203468 -2.280187 -2.357839 -2.438747 -2.520064 -2.597731 -2.
539857 -2.695756 -2.558097 -2.739831 -2.600260 -2.780784 -2.669243 -2.768857 -2.733423
-2.695602 -2.660168 -2.631030 -2.598196 -2.572203 -2.555846 -2.538806 -2.531569 -2.52
5798 -2.529477 -2.535406 -2.546891 -2.563246 -2.586028 -2.611861 -2.646406 -2.681561 -
2.726405 -2.770532 -2.822382 -2.877910 -2.933686 -2.987903 -2.908446 -3.037740 -2.8824
06 -3.040673 0.010954 -2.783200 2.889200 1.994232 1.994202 0.032621 -1.497599 0.086460
-2.779700 2.893300 1.993682 1.991808 -0.024252 -1.493808
Running impulse...
Predictions (time: 0 ms.):
class1: 0.017982
class10: 0.017982
class11: 0.017663
class2: 0.017684
class3: 0.066523
class4: 0.103487
class5: 0.017382
class6: 0.017680
class7: 0.684447
class8: 0.019721
class9: 0.019447
run_classifier returned: 0 (DSP 8 ms., Classification 0 ms., Anomaly 0 ms.)
Begin output
[0.01798, 0.01798, 0.01766, 0.01768, 0.06652, 0.10349, 0.01738, 0.01768, 0.68445, 0.01
972, 0.01945]
```

Figure 25: Output of a compiled executable performing inference over the previously extracted raw features

Figure 24 on the previous page illustrates the output of the compiled executable that performs inference over the raw features extracted in the previous step. First, it prints the **processed features array** — the combined output of the “Spectral Features” and “Flatten” blocks. It then shows which **confidence** the model assigns to each of the 11 classes for this given sample. The **highest confidence (0.684)** was assigned to class 7, meaning that the model classified the sample it was provided as **class 7**. This is in line with the real label of the sample we provided it with.

The build processes, the toolchains utilized, and the core application logic may vary depending on the hardware target. However, the fundamental concept remains consistent: The dependency-free C++ code exported from the Edge Impulse platform's project can be easily integrated into any firmware project and called through several API calls.

Automation

Developing ML applications is an iterative procedure. As the experiment matures and more data is collected, keeping track of all the steps necessary to transform the data to the format for use in the current use case can become cumbersome.

Edge Impulse provides a capability to build automated **Data Pipelines** — a predefined set of steps that can be triggered based on events (for example, when new data is added to the S3 storage), time period (e.g., once every week), or manually.

One can import datasets from existing cloud storage buckets, automate and schedule the imports, label the new data, retrain the model, automatically schedule a deployment task, and many more automation scenarios.

Figure 26 illustrates an example of applying automated pipelines in an end-to-end architecture that can be applied in the development of an ML-enabled industrial appliance.

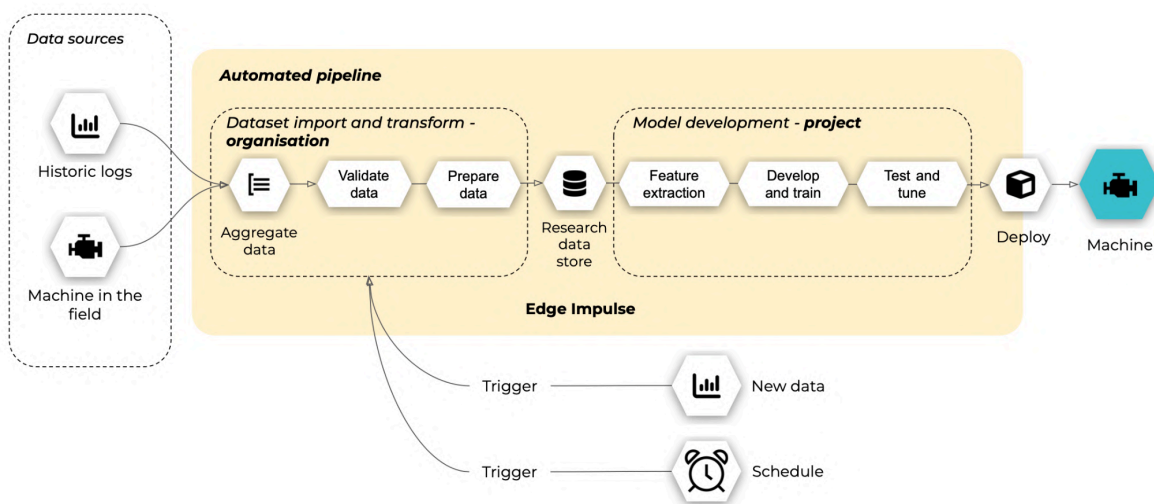


Figure 26: An example end-to-end automated ML flow set up in Edge Impulse

Below are the instructions to create two pipelines:

- One for **fetching, transforming, and importing** the new data from an S3 bucket to a project dataset
- And the other to **retrain the model** with the updated dataset and then create a new C++ deployment package

Automating Data Import and Transformation — Organization Data Pipeline

In the “**Data**” section a transformation block was introduced that takes the data in the format of the open sensorless AC motor failures dataset and converts it into JSON format of Edge Impulse. It was used to transform the AC data in the S3 bucket and store it either in an Organization Dataset or directly in the project.

Now, it is time to set up a pipeline that will perform the same set of actions but automatically as soon as new data appears in your S3 bucket.

Figure 27 illustrates an example of such a pipeline from an architectural point of view.

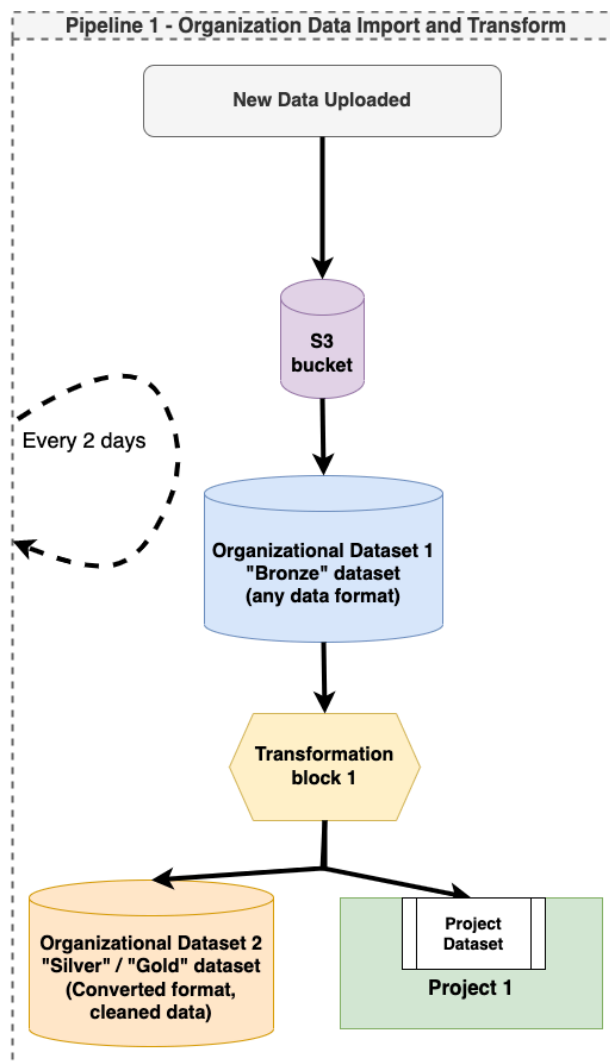


Figure 27: Example architecture of organization data import and transform pipeline

Step 1 — Copy the Transformation Job as a Pipeline Step

Navigate to the “Data Transformation” tab in your organization, select the “**Create Job**” pane on top, and fill in the parameters of the job.

Select the transformation block that was uploaded to the organization earlier, and give the job a name. Now, instead of running this job directly, click “**Copy as pipeline step.**” This will copy a JSON-encoded job descriptor in the buffer.

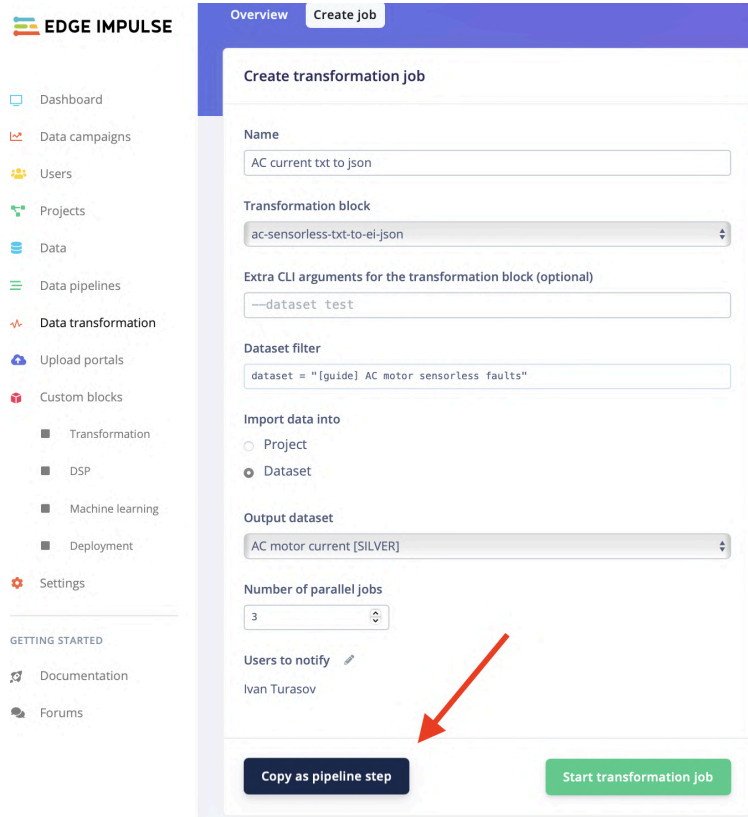


Figure 28: Configuring a transformation job and copying it as a pipeline step

Step 2 — Create the Automatic Pipeline

Navigate to the “**Data Pipelines**” tab on the organization page. Press “**+ Add new pipeline**” and paste the pipeline step from the previous point between the square brackets. Fill in the name and the description of the transform job, select a project that this pipeline will course the data in, and optionally a second dataset in the Edge Impulse organization where the transformed data is stored (called “SILVER dataset in this example”).

Set the **interval** at which this pipeline will be automatically re-run (in this example, it is set to 2 days).

Press “**Add pipeline**” — after this, the pipeline will launch for the first time, and the “Active pipelines” progression will be in view. It should be clear when the next time this pipeline will run based on the parameters now specified.

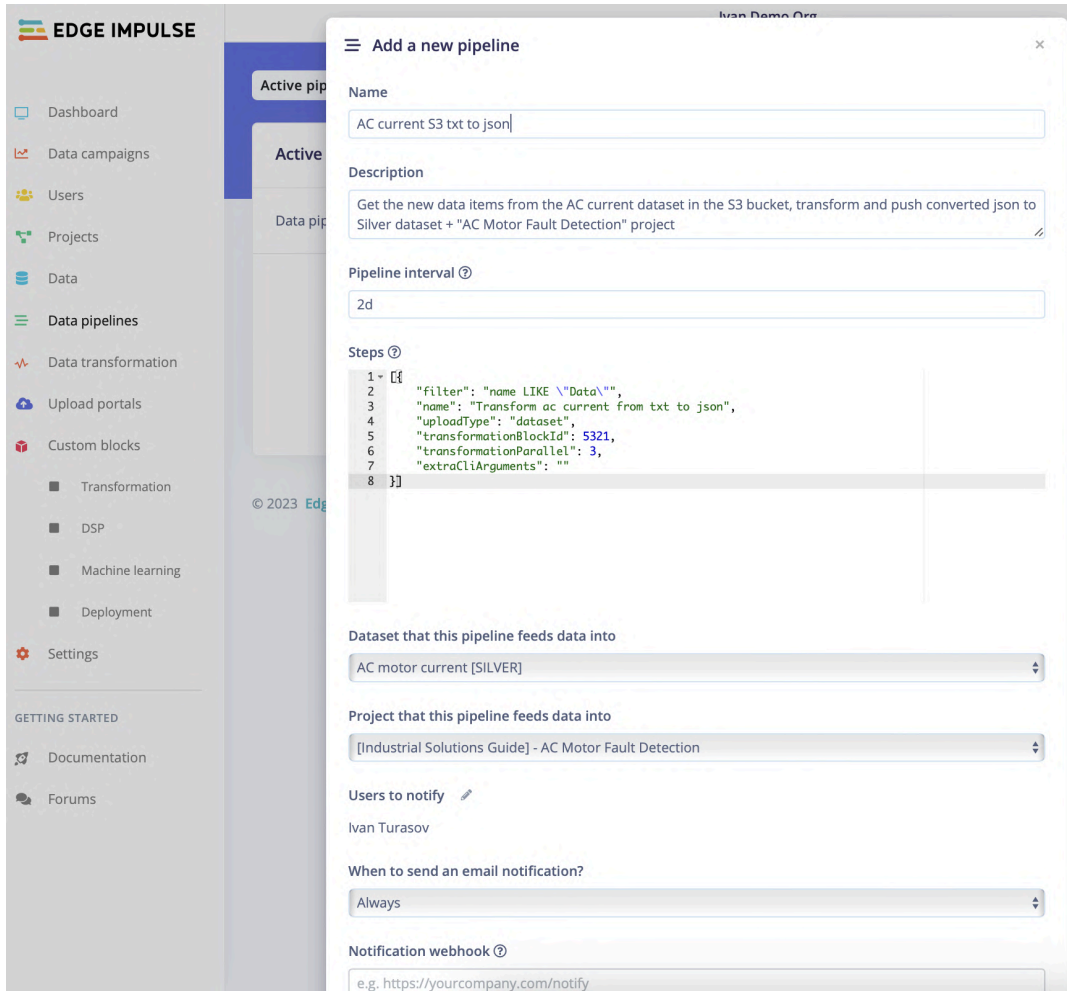


Figure 29: Copying the transformation job JSON to a pipeline

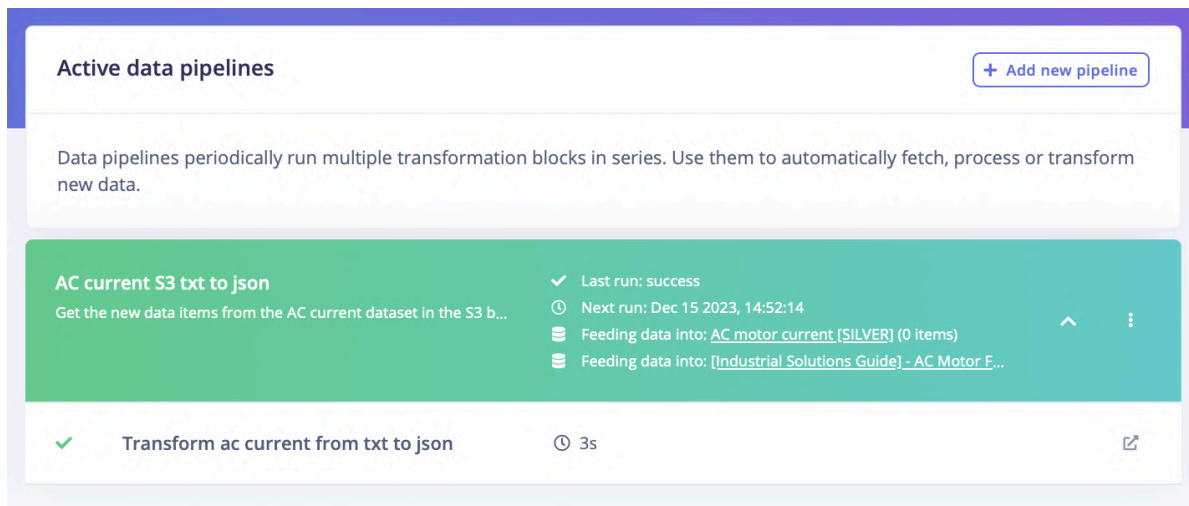


Figure 30: Configured pipeline view after a successful run

Automating Model Retraining and Deployment

The pipeline created will ensure that the project dataset is **always up to date with the organizational data**.

The next step is to amend it with another pipeline on the **project level**. As new data gets added, the pipeline smoothly manages the entire impulse lifecycle from start to finish:

- It **recreates DSP features** for an updated dataset and retrains the model
- It **retrains and versions the retrained model**, keeping track of the model's evolution over time
- It also **constructs a new C++ library deployment**, ensuring that the updated model is efficiently integrated into the target device of choice

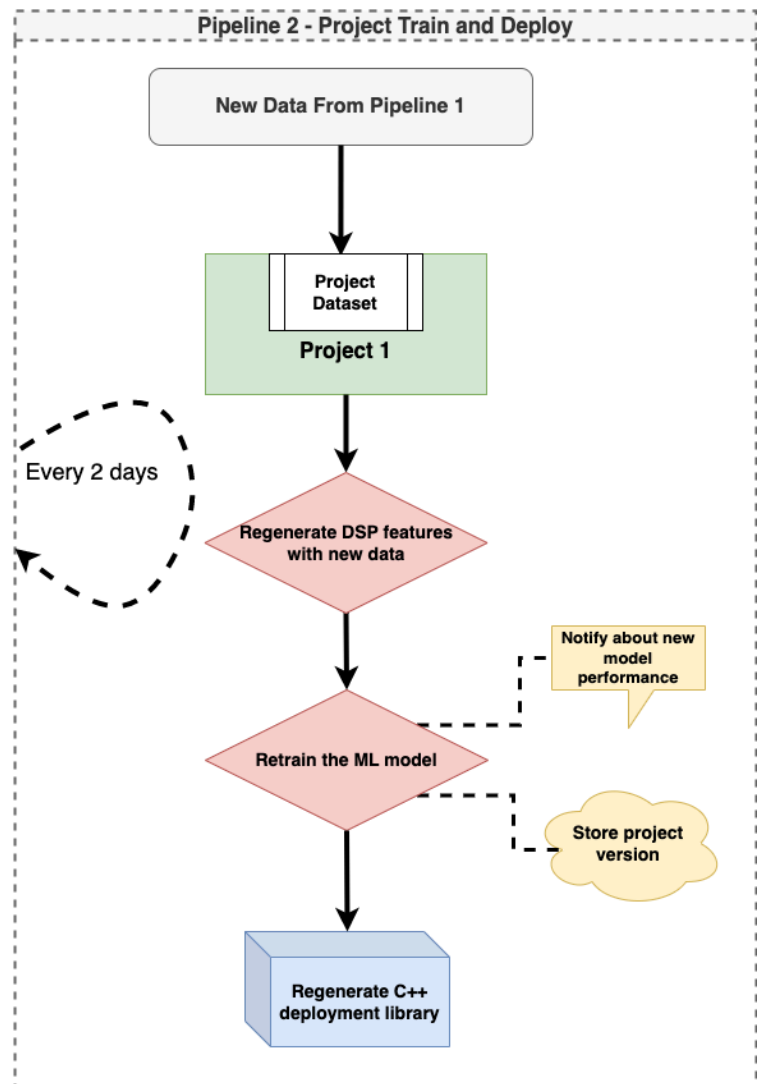


Figure 31: Example architecture of automated Project train and deploy pipeline

This approach allows to close the loop and automate the last mile of the end-to-end Edge ML flow — from data ingestion all the way to optimized edge library creation.

Follow the steps below to configure this automated data pipeline:

Step 1 — Navigate to Data Source Configuration in the Project

Click “+ Add new data source” and select “Don't import data.” The reason for this selection is that the data in the project is already updated by the previous pipeline.

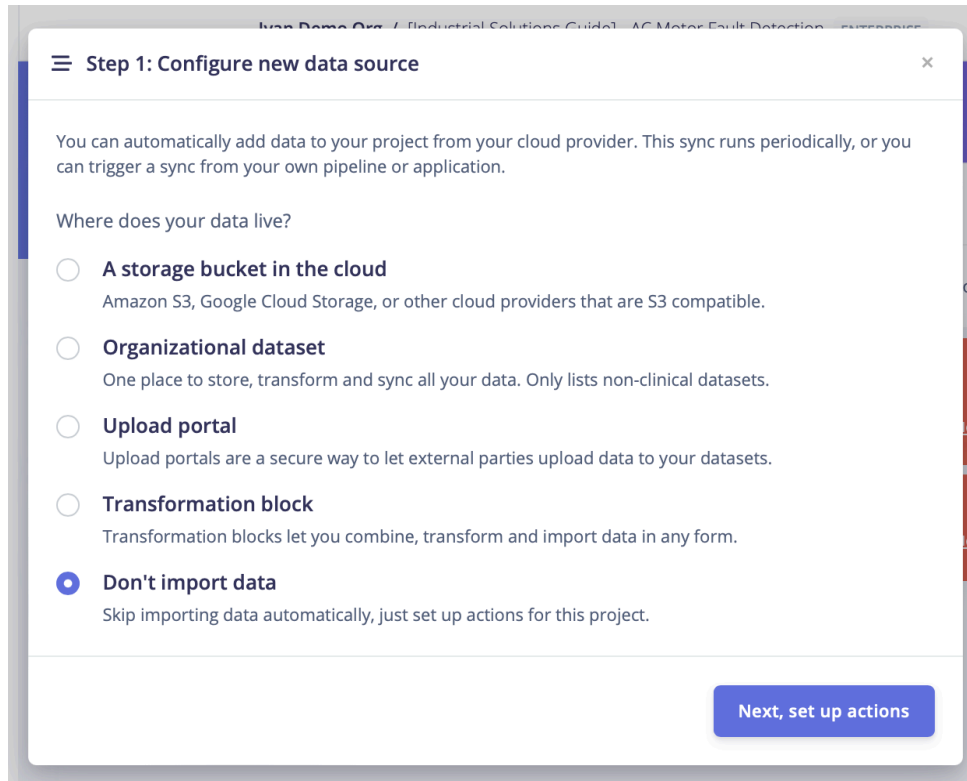


Figure 32: Configuring a data source for a project

Step 2 — Configure Pipeline Steps and Interval

Select the project actions that will be automatically performed every time the pipeline is invoked. Configure the pipeline interval. All the parameters can be changed at a later point. Press “Create pipeline.”

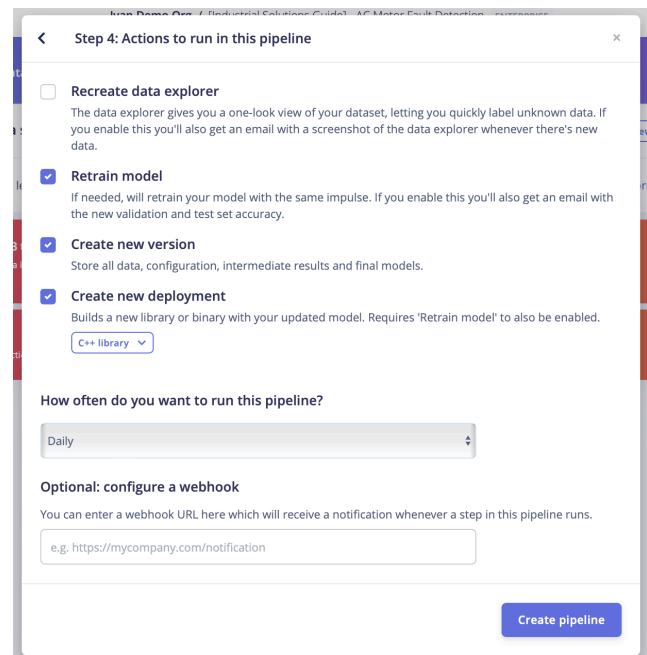


Figure 33: Configuring a pipeline for the project

Collaboration

Collaboration and reproducibility are essential pillars of impactful product development and maintenance. **Edge Impulse** offers a unified environment where both embedded and ML teams can work together. It supports the deployment of trained models directly onto resource-constrained embedded devices.

To add collaborators, press the icon in the **“Collaborators”** pane on the project dashboard and enter a username or an email address of a person to add. This feature ensures that the ML models are integrated smoothly into the embedded system, eliminating potential integration challenges and fostering collaboration between the two teams.

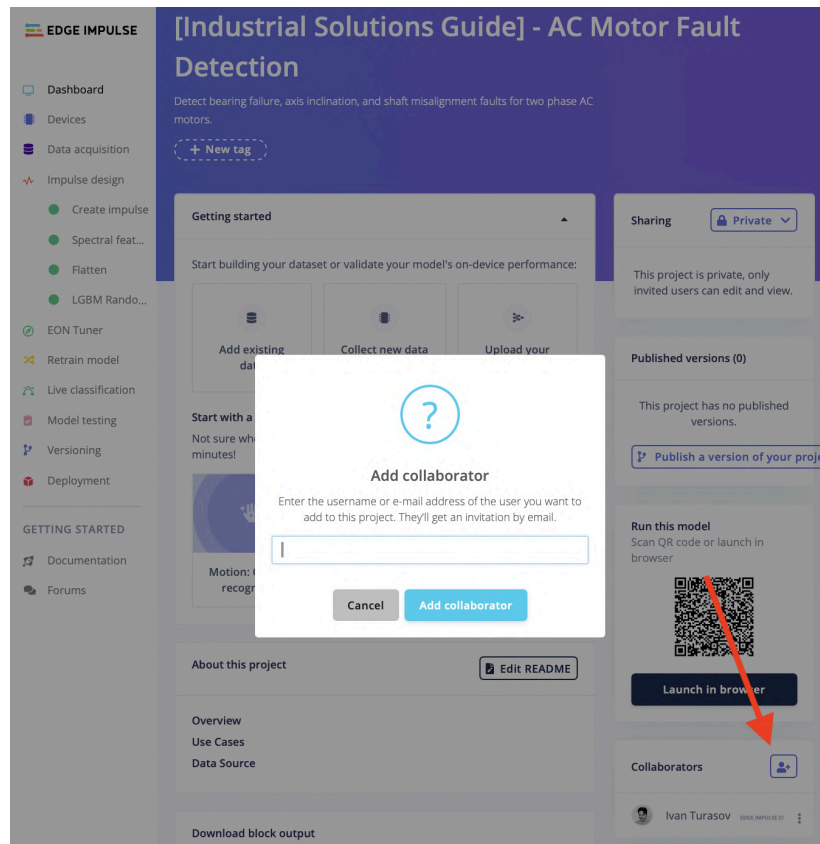


Figure 34: Adding a collaborator to the project

Additionally, by documenting the origin of the dataset, data processing steps, and transformations within your project's README, Edge Impulse fosters reproducible engineering practices, allowing your peers to **replicate and validate** your findings.

Press "Edit README" in the "About this project" pane on the project dashboard to create a description.

Summary

Edge Impulse's components are designed to address the different stages of a typical **ML pipeline**. The arrangement of these components reflects the common process of data collection, data cleaning/ transformation, feature extraction, model training, testing, and deployment.

By mirroring the standard stages of the ML pipeline, Edge Impulse ensures users have all the tools they need at each stage. This integrated approach enables teams or anyone, whether an expert or a beginner, to build and deploy ML models seamlessly.

The "glue" that connects all these components together in Edge Impulse is its unified software platform. This allows users to manage each stage of the machine learning workflow right from data ingestion to model deployment, making it faster and easier than ever to get to market with Edge AI.

All the components are organized in a cohesive manner and tightly integrated into the platform's user interface. For example, data gathered and processed in one stage of the workflow is seamlessly available for the next, and so on. Additionally, Edge Impulse provides [APIs and SDKs](#) for integrating its platform with other tools and systems. This enables users to customize their workflows and use external tools when necessary.

The real power of Edge Impulse lies in its **modular and flexible architecture**. Users can modify and iterate over an impulse design, data, and model as often as needed until achieving a satisfactory result.



F.A.Q.

What are the minimum hardware requirements to run the Edge Impulse inferencing library on my embedded device?

The minimum hardware requirements for the embedded device depend on the use case, anything from a Cortex-M0+ for vibration analysis to Cortex-M4F for audio, Cortex-M7 for image classification to Cortex-A for object detection in video.

What frameworks does Edge Impulse use to train the machine learning models?

We use a wide variety of tools, depending on the machine learning model. For neural networks, we typically use TensorFlow and Keras. For object detection models we use TensorFlow with Google's Object Detection API, and for 'classic' non-neural network machine learning algorithms, we mainly use sklearn. For neural networks, you can see (and modify) the Keras code by selecting “**Switch to expert mode**” in the block context menu.

Another big part of Edge Impulse is the processing blocks, which can be used for data cleansing or data processing to extract important features before passing it to a machine learning model. The source code for these processing blocks can be found on GitHub: [edgeimpulse/processing-blocks](https://github.com/edgeimpulse/processing-blocks) (and one can build your own processing blocks as well).

What engine does Edge Impulse use to compile the Impulse?

It depends on the hardware.

For general-purpose MCUs, we typically use **EON Compiler** with TFLite Micro kernels (including hardware optimization, e.g. via CMSIS-NN, ESP-NN).

On Linux, if you run the Impulse on the CPU, we use TensorFlow Lite.

For accelerators, we use a wide variety of other runtimes, e.g., hardcoded network in silicon for Syntiant, custom SNN-based inference engine for Brainchip Akida, DRP-AI for Renesas RZV2L, etc.

Is there a downside to enabling the EON Compiler?

The [EON Compiler](#) compiles your neural networks to optimized C++ source code, which is then compiled into your application. This is great if you need the lowest RAM and ROM possible (EON typically uses 30-50% less memory than TensorFlow Lite), but you also lose some flexibility to update your neural networks in the field — as it is now part of your firmware.

By disabling EON, we place the full neural network (architecture and weights) into ROM and load it on demand.

Can I use a model that has been trained elsewhere in Edge Impulse?

Yes. [Bringing your own model \(BYOM\)](#) feature was designed for this.

How does the Feature Explorer visualize data that has more than three dimensions?

Edge Impulse uses [UMAP](#) (a dimensionality reduction algorithm) to project high dimensionality input data into a 3-dimensional space. This even works for extremely high dimensionality data such as images.

What is the typical power consumption of the Impulse running on my device?

Simple answer: To get an indication of time per inference, we show performance metrics in every DSP and ML block in the Studio. Multiply this by the active power consumption of your MCU to get an indication of power cost per inference.

A more complicated answer: It depends. Normal techniques to conserve power still apply to ML, so try to do as little as possible (do you need to classify every second, or can you do it once a minute?), be smart about when to run inference (can there be an external trigger like a motion sensor before you run inference on a camera?), and collect data in a lower power mode (don't run at full speed when sampling low-resolution data, and see if your sensor can use an interrupt to wake your MCU — rather than polling).

Also see [Analyze Power Consumption in Embedded ML Solutions](#).

What is the .eim model format for Edge Impulse for Linux?

See the [extensive documentation page](#) on our documentation portal.

How is the labeling of the data performed?

Using the Edge Impulse Studio data acquisition tools (like the serial daemon or data forwarder), you can collect data samples manually with a predefined label. If you have a dataset that was collected outside of Edge Impulse, you can upload your dataset using the Edge Impulse CLI, [data ingestion API](#), web uploader, [enterprise data storage bucket tools](#) or enterprise upload portals. You can then utilize the Edge Impulse Studio to split up your data into labeled chunks, crop your data samples, and more to create high quality machine learning datasets.

About the Author

[Ivan Turasov](#) is a seasoned Edge ML Solutions Engineer at Edge Impulse, where he works with Edge Impulse customers, helping them build ML-enabled products that go to market, ensuring best practices of Edge ML and Edge Impulse are applied. He graduated with a MSc in Embedded Systems from Eindhoven University of Technology in the Netherlands.

