# Procedure to Run an AI (CNN dogs vs cats) on KV260 with Burn

**1️⃣ Model Creation and Training**

1. Choose a model:
   1. Example: a simple CNN (MobileNet, lightweight ResNet, or a custom model).
2. Prepare the data:
   1. Organize images into two folders: cat/ and dog/.
   2. Create training, validation, and test datasets.
3. Train the model (on PC):
   1. With TensorFlow/Keras (.h5) or PyTorch.
   2. Adjust hyperparameters (batch size, epochs, learning rate).
4. Evaluate performance:
   1. Check accuracy and loss on the test set.
   2. At this stage, you have a working model… but only usable on your PC.

2️⃣ **Model Conversion**

1. **Export to ONNX**
   * a. If you have a .h5, convert it to .onnx
   * b. Install ONNX:
     1. pip install onnx
2. Check ONNX compatibility:

python3 - <<'PY'

import onnx

m = onnx.load("models/dog\_vs\_cat.onnx")

print("IR version:", m.ir\_version)

print("Opset imports:", [(d.domain, d.version) for d in m.opset\_import])

print("First nodes:", [n.op\_type for n in m.graph.node][:30])

PY

* If you see exotic operations, check whether they are supported by Burn.
* Otherwise, re-export your model with another opset or adjust its structure.

**3️⃣ Preparing the KV260**

1. Clone Burn:
   * git clone https://github.com/tracel-ai/burn.git
   * cd burn
2. Install dependencies:
   * Rust + Cargo (build tool).
   * Update the KV260 (internet required).
3. Organize project files:
   * Create a folder for your model (model/dog\_vs\_cat.onnx).
   * Create a folder for test images (images/).

**4️⃣ Code Generation with onnx2burn**

1. Install the Burn ONNX import tool:
   * cargo install burn-import
2. Convert your model:
   * burn-import onnx models/dog\_vs\_cat.onnx --out generated\_model
3. This generates Rust code (generated\_model.rs) that can be used directly.

**5️⃣ Rust Project Creation**

1. Create a new project:

cargo new dog\_vs\_cat\_burn --bin

cd dog\_vs\_cat\_burn

1. Copy the generated code (generated\_model/\*) into src/.
2. Edit your Cargo.toml:

[dependencies]

burn = "0.18"

burn-ndarray = "0.18"

image = "0.25" # for image loading and preprocessing

**6️⃣ Inference Implementation (main.rs)**

Minimal example structure:

mod generated\_model; // code produced by onnx2burn

use generated\_model::MyModel; // name depends on the generator

fn main() -> Result<(), Box<dyn std::error::Error>> {

// 1) load and preprocess the image (use image crate)

// 2) convert into Burn tensor

// 3) load weights (if necessary: generated\_model::load\_state("..."))

// 4) run inference

// let pred = model.forward(input\_tensor);

// 5) display label

Ok(())

}

**7️⃣Compilation and Execution**

On the KV260:

cargo build --release

./target/release/dog\_vs\_cat\_burn images/test1.jpg

* ⚠️ Compilation on ARM can be slow.
* If needed, cross-compile from your PC to ARM (more complex to set up).

**8️⃣ Debugging and Common Errors**

* ONNX import error: unsupported operation → simplify your model or use onnxruntime instead.
* Preprocessing error: wrong shape or channel order → ensure Rust preprocessing matches your Python pipeline.
* Very slow build on KV260: possible → use cargo build --release or cross-compilation.