Exactly right, Tsutomu — you've got the idea.

Yes, the **input to the compilation step is a trained AI model**, typically in a portable format like **ONNX**. You can absolutely:

1. **Download a pretrained ONNX model** (e.g., from ONNX Model Zoo)
2. **Compile it separately** for:
    - **TI AM68A** using the **TIDL compiler** in the Edge AI SDK
    - **Renesas RZ/V2N** using the **DRP-AI Translator + TVM toolchain**

🧰 Platform-Specific Compilation Pipelines

🔧 TI AM68A (Edge AI SDK)

- **Input**: ONNX, TFLite, or Caffe model
- **Tool**: tidl_model_import.out or Edge AI Studio
- **Steps**:
    1. Convert to intermediate representation (IR)
    2. Quantize (INT8 recommended)
    3. Generate .bin and .net files for runtime
- **Output**: TIDL artifacts + metadata for inference on C7x/MMA

🔧 Renesas RZ/V2N (DRP-AI TVM)

- **Input**: ONNX model
- **Tool**: DRP-AI Translator + TVM compiler
- **Steps**:
    1. Translate ONNX to DRP-AI intermediate format
    2. Quantize and optimize
    3. Generate DRP-AI binary and runtime metadata
- **Output**: DRP-AI binary + config for runtime execution

⚠️ Caveats to Watch For

- **Operator Support**: Both platforms support a subset of ONNX ops. You may need to simplify or retrain models.
- **Input Shape**: Fixed-size inputs are often required (e.g., 224×224 RGB)
- **Quantization**: Post-training quantization may degrade accuracy — some models benefit from quantization-aware training
- **Runtime APIs**: TI uses TIDL runtime; Renesas uses DRP-AI runtime — both require specific initialization and buffer handling

---

If you'd like, I can walk you through compiling a specific ONNX model (like MobileNet or

YOLOv5) for both platforms, or help you set up a reproducible test harness to compare inference latency and buffer flow. Want to try that?

Great question, Tsutomu — and no, **"compile" does not mean training** in this context. Here's how it breaks down for both TI and Renesas:

🧠 **What "Compile" Means in Embedded AI**

In embedded AI toolchains like TI's Edge AI SDK or Renesas' DRP-AI TVM, **"compiling a model" refers to converting a trained model into a format optimized for inference on the target hardware** — not training it.

🔧 **Steps Typically Involved in "Compiling":**

1. **Model Conversion**: From frameworks like TensorFlow, PyTorch, or ONNX to an intermediate format
2. **Quantization**: Reducing precision (e.g., FP32 → INT8) for speed and efficiency
3. **Operator Mapping**: Matching neural network layers to hardware-accelerated kernels
4. **Graph Optimization**: Fusing layers, pruning unused ops, etc.
5. **Code Generation**: Producing binaries or runtime blobs for deployment

---

📦 **TI AM68A: Edge AI Compilation Flow**

- Uses **TI Deep Learning (TIDL)** compiler
- Converts models to run on **C7x DSP + MMA (Matrix Multiply Accelerator)**
- Supports quantization-aware training or post-training quantization
- Output: .bin and .net files for runtime execution

🧪 **Renesas RZ/V: DRP-AI Compilation Flow**

- Uses **DRP-AI Translator + TVM** toolchain
- Converts ONNX models into DRP-AI executable format
- Handles quantization, operator mapping, and memory layout
- Output: DRP-AI binary + metadata for runtime

🧠 Training vs. Compilation

Step	Training	Compilation
Purpose	Learn model weights from data	Optimize model for inference
Tools	PyTorch, TensorFlow, etc.	TIDL (TI), DRP-AI TVM (Renesas)
Output	.pt, .pb, .onnx	.bin, .net, DRP-AI binary
Runs on	GPU/CPU during dev	Embedded AI accelerator

If you're prepping models for display overlays or latency profiling, the compilation step is where you'd align tensor shapes, memory layout, and scheduling to match your pipeline. Want help walking through a sample compile flow for either platform?