# Edge AI Workshop

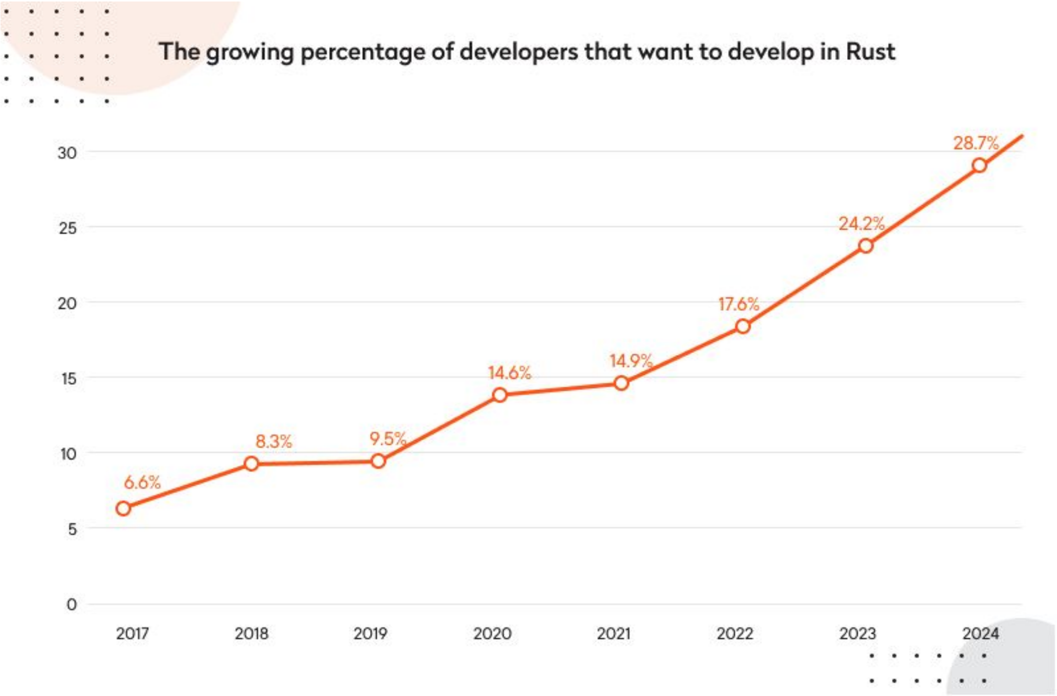
## Rust Workshop

# Materials

# Interest in Rust

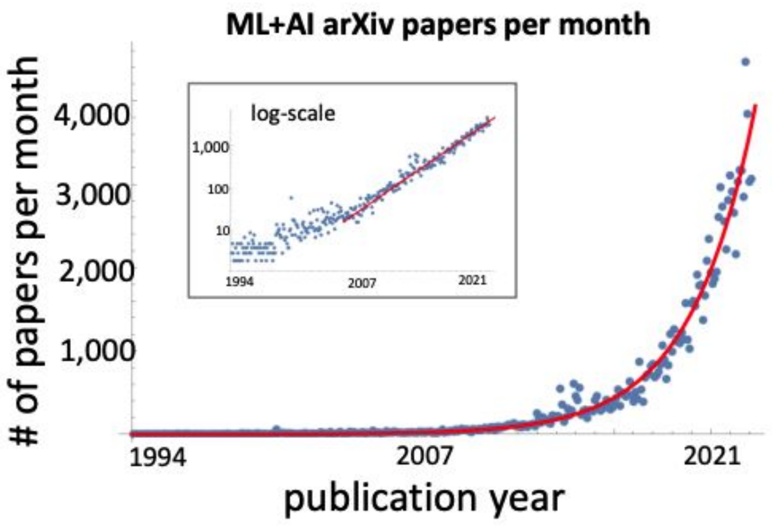
- Memory Safe
- Strongly Typed
- Extensive Ecosystem (crates.io)

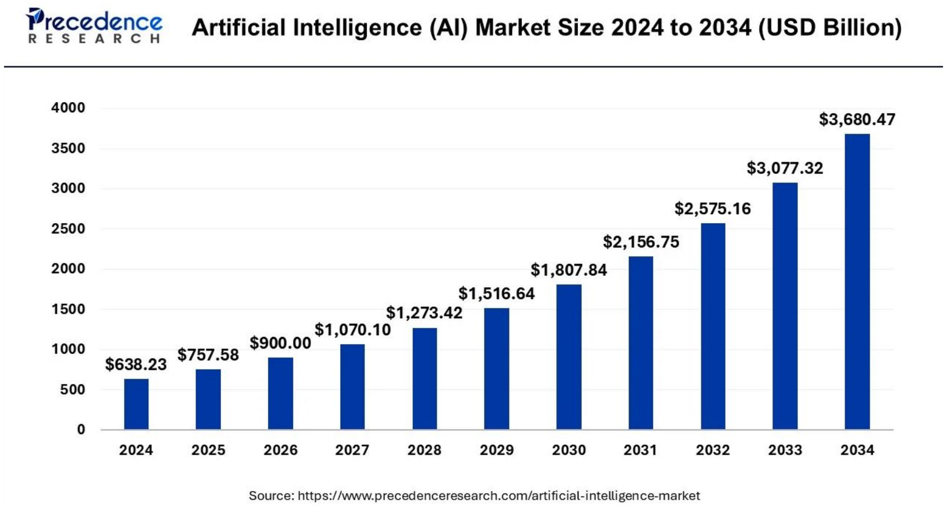
*Java Styles at C/C++ speed*



# Interest in AI - It keeps growing, but for how long?

- Exponential number of new papers each year!
  - NeurIPS (“best” AI conference) 2025: a record of **25,000** submissions
- More funding, investments for AI companies

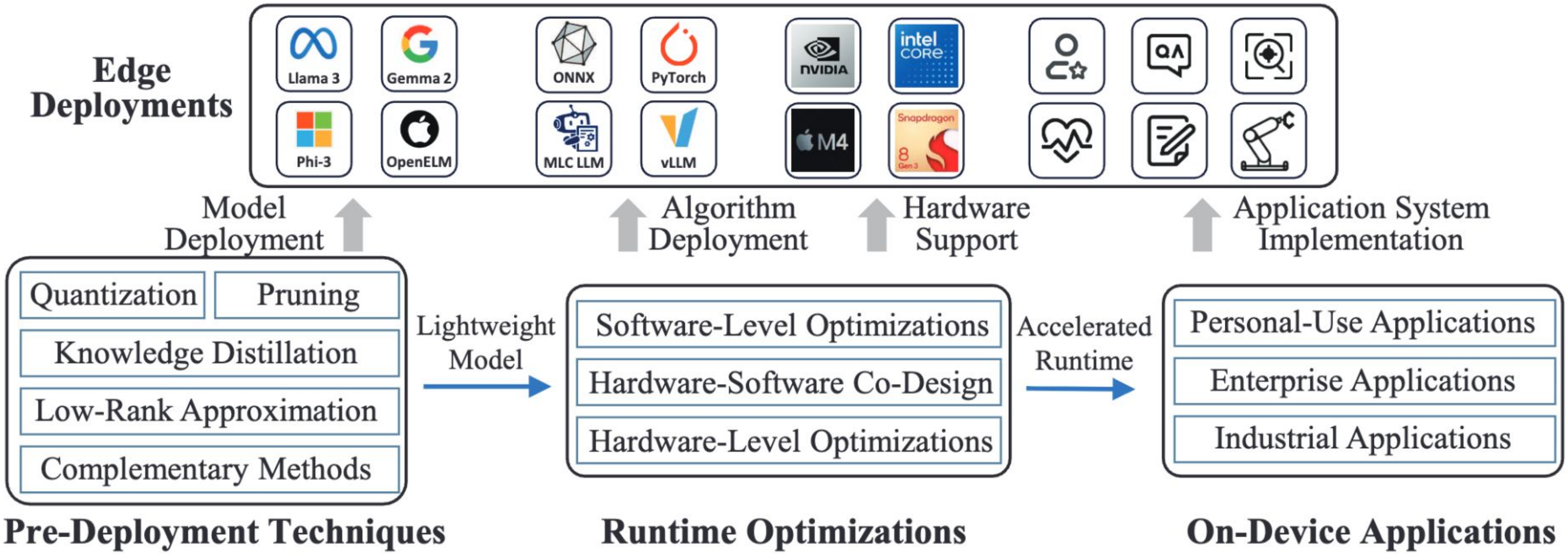


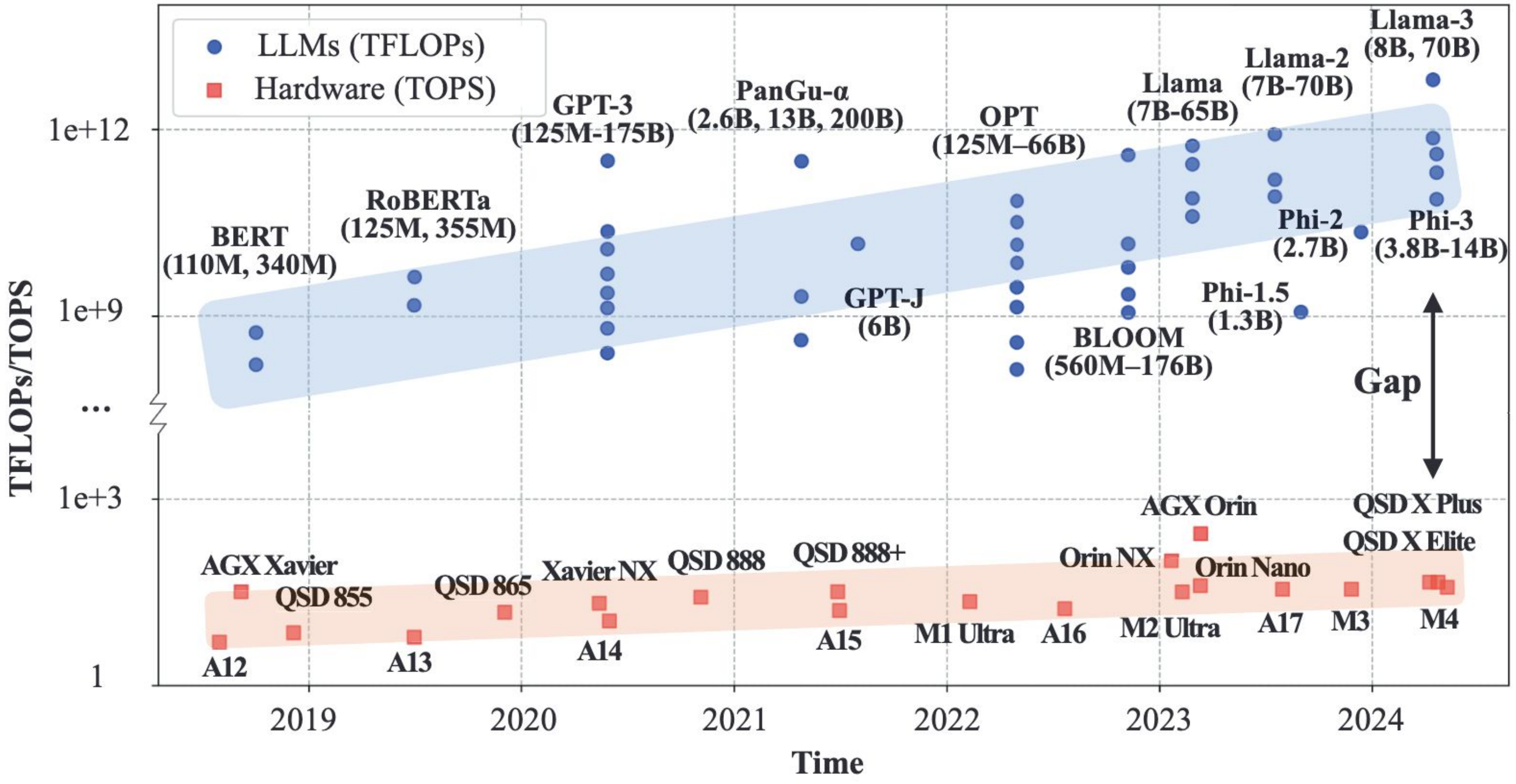
# Edge AI

- Deployment of AI models directly on local devices or “edge devices”
    - like sensors, cameras, IoT gadgets, smartphones, industrial machinery
    - Raspberry Pi

- Self-driving cars
- Drones
- Smartphones
- etc.



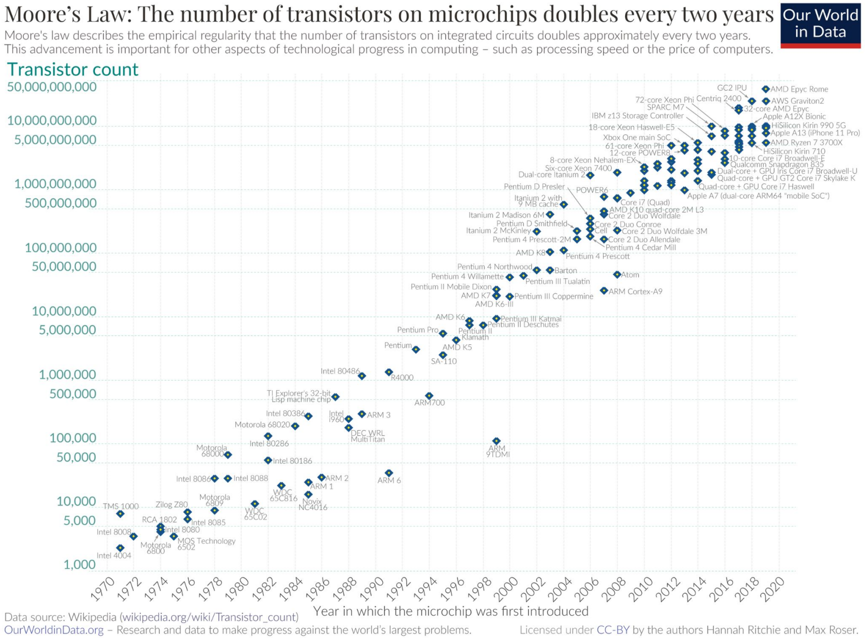

Zheng, Yue, et al. "A review on edge large language models: Design, execution, and applications." ACM Computing Surveys 57.8 (2025): 1-35.




Zheng, Yue, et al. "A review on edge large language models: Design, execution, and applications." ACM Computing Surveys 57.8 (2025): 1-35.

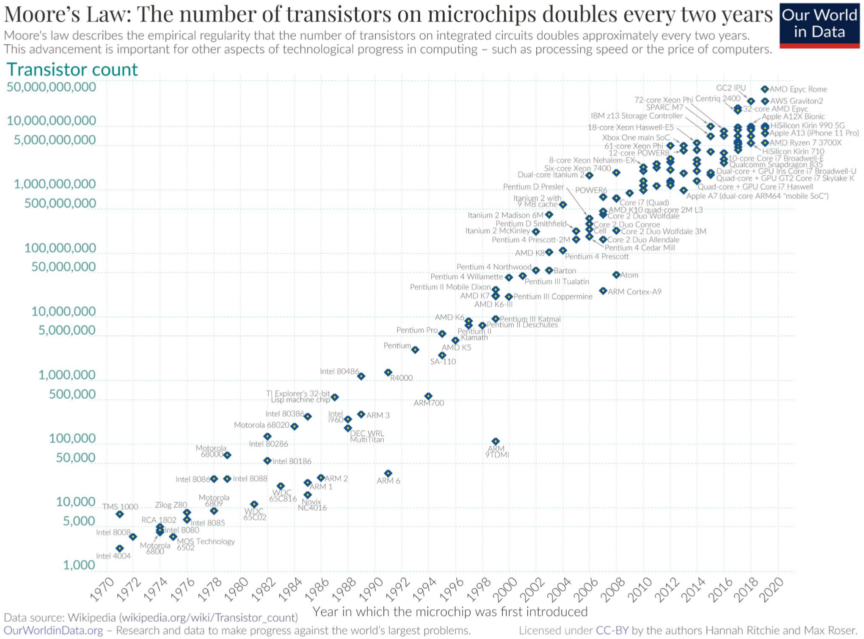
# Edge AI - Why is it possible?

- Moore’s Law
  - Just 6-7 years ago, running a real-time ~50M parameter model on edge devices was a struggle
  - Now, we can run 1B parameter models
    - ~10x more computing power



# Edge AI - Why is it possible?

- Moore's Law
  - Just 6-7 years ago, running a real-time ~50M parameter model on edge devices was a struggle
  - Now, we can run 1B parameter models
    - ~10x more computing power

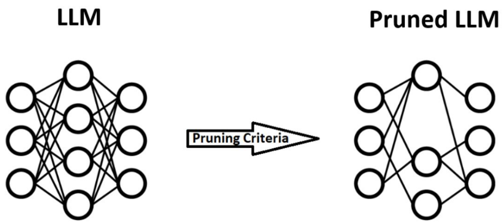



- Not just due to hardware advancements!
- We have better inference techniques:
  - Quantization, pruning
  - **Distillation!**

# Ingredients for Edge AI

- Good, cheap hardware
- If you don't have it, just wait a couple of years.

# Ingredients for Edge AI

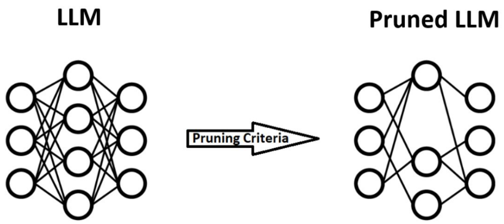



- Good, cheap hardware
- If you don't have it, just wait a couple of years.

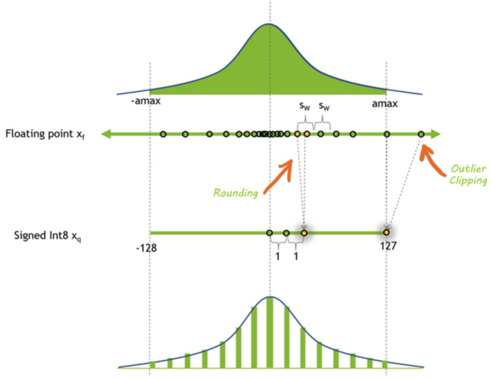
- Model pruning
- Get a big model and make it smaller by deleting neurons / layers

# Ingredients for Edge AI

- Good, cheap hardware
- If you don't have it, just wait a couple of years.



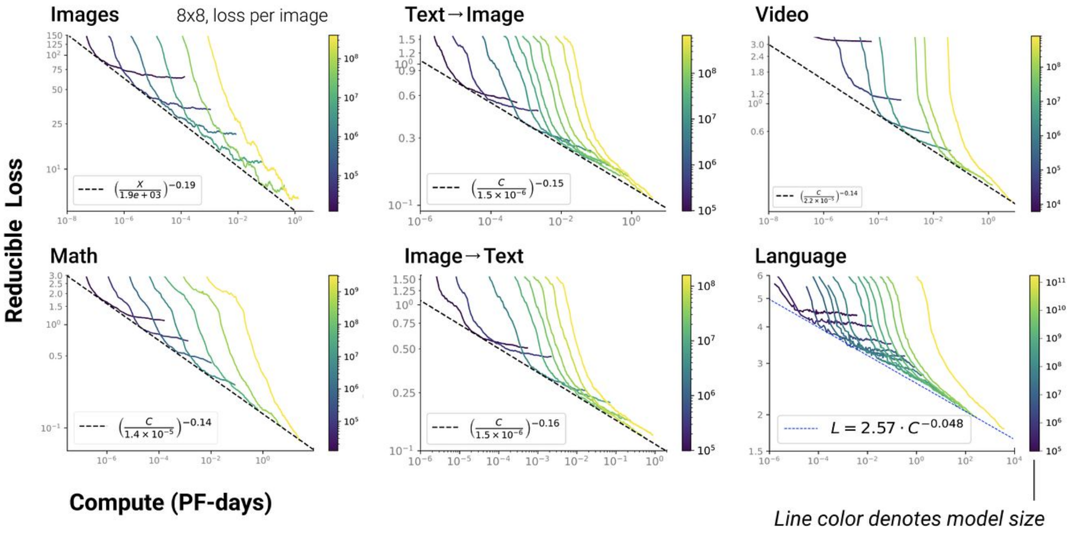

- Model pruning
- Get a big model and make it smaller by deleting neurons / layers




- Reduce precision of weights to save memory / inference time

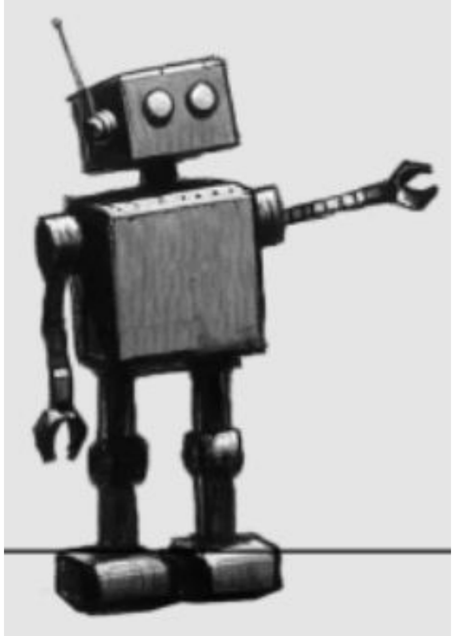
# The real reason: We scaled-up

- Consistent in AI: **scale** leads to predictable improvements in capability
  - Model size, dataset size, compute
- Moore's Law enables us to scale faster
- Paradoxically, scaling up also facilitates scaling down!



# The real reason: We scaled-up and distilled the knowledge

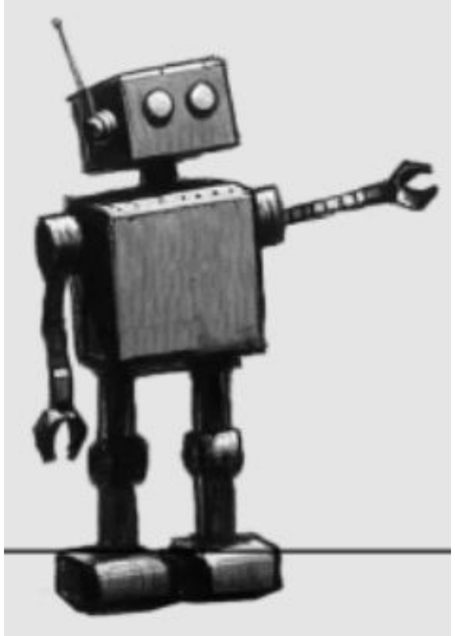
- We trained larger and larger and more capable models
  - GPT-2, GPT-3, GPT-4, GPT-5, ...
  - Deepseek-R1 (~400B params)
  - Qwen-480B

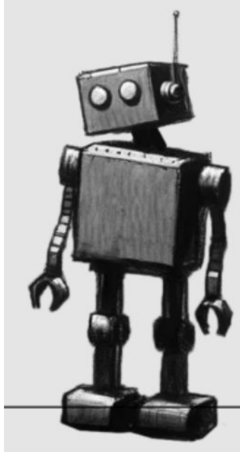


# The real reason: We scaled-up and distilled the knowledge

- We trained larger and larger and more capable models
  - GPT-2, GPT-3, GPT-4, GPT-5, ...
  - Deepseek-R1 (~400B params)
  - Qwen-480B

- **Large models** can then teach **smaller faster** versions of themselves
  - gpt-mini, gpt-nano
  - deepseek -7B
  - qwen-1B, qwen-3B



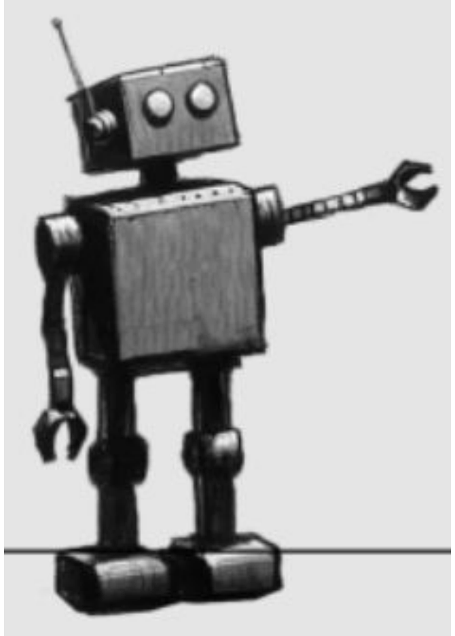


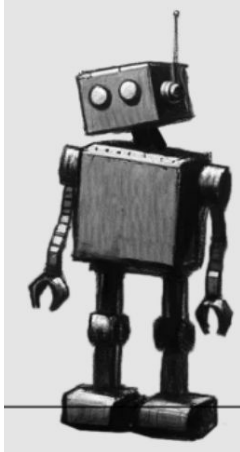
# The real reason: We scaled-up and distilled the knowledge

- We trained larger and larger and more capable models
  - GPT-2, GPT-3, GPT-4, GPT-5, ...
  - Deepseek-R1 (~400B params)
  - Qwen-480B

- **Large models** can then teach **smaller faster** versions of themselves
  - gpt-mini, gpt-nano
  - deepseek -7B
  - qwen-1B, qwen-3B

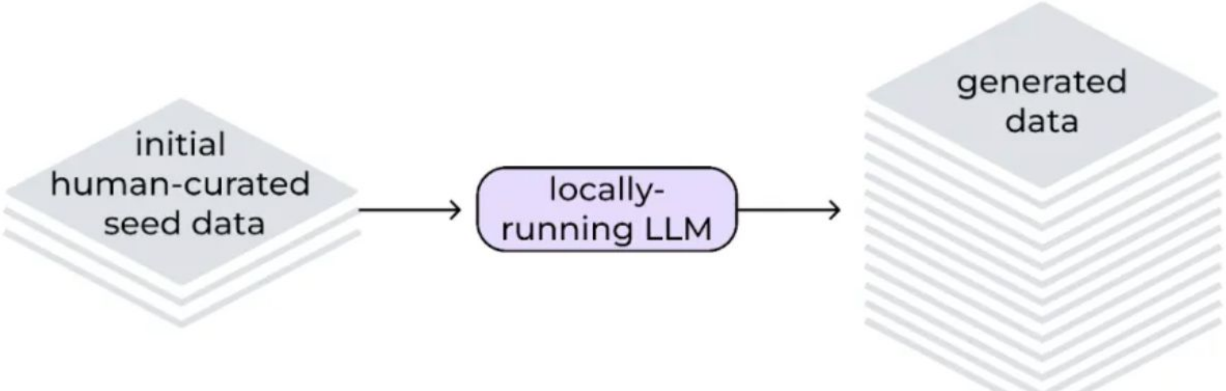
- Small models reach a performance level that cannot be otherwise obtained





# Nowadays, “distillation” has be rebranded

- **“Synthetic Data”**
- Use an (very) large, highly capable model to generate more (high-quality) data / clean existing data
    - Paraphrasing
    - Generate instructions
    - **Enrich dataset** by automatic annotations



# Sounds good but …

[Bitter lesson]: progress of AI in the past 70 years boils down to

- Develop progressively more general methods with weaker modeling assumptions
- Add more data and computation (i.e. scale up)

## The Bitter Lesson

### Rich Sutton

**March 13, 2019**

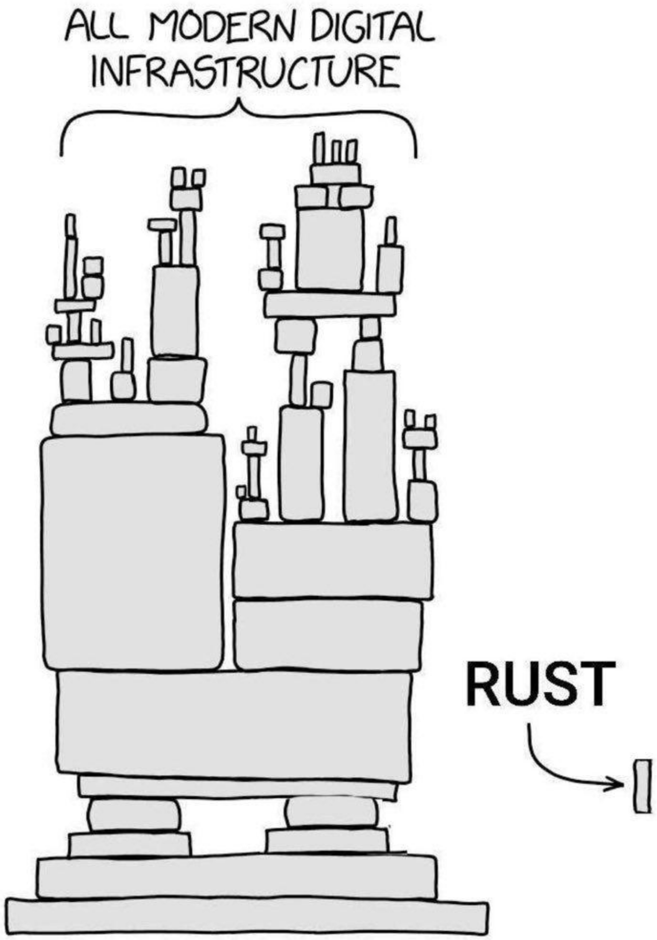
The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin. The ultimate reason for this is Moore's law, or rather its generalization of continued exponentially falling cost per unit of computation. Most AI research has been conducted as if the computation available to the agent were constant (in which case leveraging human knowledge would be one of the only ways to improve performance) but, over a slightly longer time than a typical research project, massively more computation inevitably becomes available. Seeking an improvement that makes a difference in the shorter term, researchers seek to leverage their human knowledge of the domain, but the only thing that matters in the long run is the leveraging of computation. These two need not run counter to each other, but in practice they tend to. Time spent on one is time not spent on the other. There are

# AI + Rust

- Edge AI
- Tokenizers
- WASM
- Raspberry PIs

- In practice, Rust is useful for building high-performance data pipelines

- For example: 🤗 **HuggingFace** tokenizers library is built on top of Rust



# Common ML Frameworks (non Rust)

Training

- Pytorch (Meta)
- Tensorflow (Google)
- JAX (Google)

Inference

- GGML
- ONNX Runtime (Microsoft)
- LiteRT (Google)
- Executorch (Meta)
- TVM (Apache)
- TF Micro (Google)

# Why different Frameworks for Inference and Training?

Training

- Focus on flexibility
- Feature-rich
- GPU first

(most in Python)

Inference

- Focus on speed and size
- Compiled into apps / devices
- Cross platform
- GPU, CPU, NPU, MCU

(most in C++)

# Specialized Inference Frameworks

- Cadence HiFi NN Lib
- ARM-NN
- Apple Core ML
- Rockchip NPU
- Huawei CANN
- Android NNAPI
- Intel OpenVino
- Xilinx Vitis-AI
- Qualcomm QNN

Everyone builds their own inference engine…

# How Rust can help

Everything that makes Rust great in other use-cases also applies to ML:

- Easy cross-compilation
- Great optimization of the same code on different architectures
- Memory Safety
- Awesome Tooling
- Great abstractions of complex patterns

# Rust ML Frameworks

Burn
(Tracel AI)

ort
ML inference for Rust
(Pyke.io)

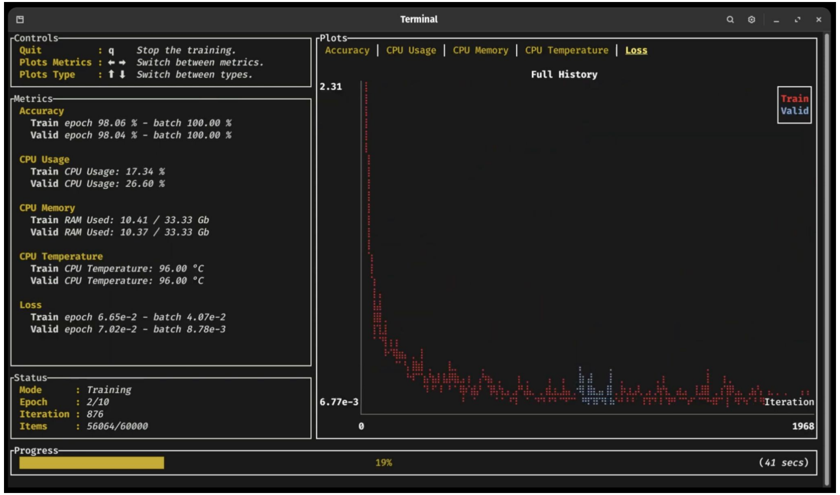
tract
(Sonos)

Candle
Hugging Face

# Burn (Tracel AI)

- Training & Inference
- GPU, CPU & MCU (no_std)
- Own GPU compute language (CubeCL)
- GPU works on NVIDIA, AMD, Intel & Web

- Only basic ONNX import
- GPU and larger model focus
- Especially slow for embedded platforms
- No Accelerator support (NPU)



# Burn ONNX Import

| ONNX OP | Import Support | Burn Support |
|---|---|---|
| Abs | ✅ | ✅ |
| Acos | ❌ | ❌ |
| Acosh | ❌ | ❌ |
| Add | ✅ | ✅ |
| And | ✅ | ✅ |
| ArgMax | ✅ | ✅ |
| ArgMin | ✅ | ✅ |
| Asin | ❌ | ❌ |
| Asinh | ❌ | ❌ |
| Atan | ❌ | ❌ |
| Atanh | ❌ | ❌ |
| Attention | ✅ | ✅ |
| AveragePool1d | ✅ | ✅ |
| AveragePool2d | ✅ | ✅ |
| BatchNormalization | ✅ | ✅ |
| Bernoulli | ✅ | ✅ |
| BitShift | ✅ | ✅ |

| | | |
|---|---|---|
| RNN | ❌ | ✅ |
| RoiAlign | ❌ | ❌ |
| Round | ✅ | ✅ |
| Scan | ❌ | ❌ |
| Scatter | ❌ | ✅ |
| ScatterElements | ❌ | ❌ |
| ScatterND | ❌ | ❌ |
| Selu | ❌ | ❌ |
| SequenceAt | ❌ | ❌ |
| SequenceConstruct | ❌ | ❌ |
| SequenceEmpty | ❌ | ❌ |
| SequenceErase | ❌ | ❌ |
| SequenceInsert | ❌ | ❌ |
| SequenceLength | ❌ | ❌ |
| SequenceMap | ❌ | ❌ |
| Shape | ✅ | ✅ |
| Shrink | ❌ | ❌ |
| Sigmoid | ✅ | ✅ |
| Sign | ✅ | ✅ |
| Sin | ✅ | ✅ |
| Sinh | ✅ | ✅ |

[...]

# Candle (Hugging Face)

Hugging Face

- Good selection of already implemented models
- Many examples
- Support for CUDA, Metal, Intel MKL, Apple Accelerate
- Programmatically first

- Only very basic ONNX import
- No universal GPU support (AMD, Intel, Web)
- No embedded accelerators

```rust
use candle_core::{Device, Tensor};

fn main() -> Result<(), Box<dyn std::error::Error>> {
    let device = Device::Cpu;

    let a = Tensor::randn(0f32, 1., (2, 3), &device)?;
    let b = Tensor::randn(0f32, 1., (3, 4), &device)?;

    let c = a.matmul(&b)?;
    println!("{c}");
    Ok(())
}
```

# Tract (Sonos)

- Pure Rust
- Support of most ONNX ops
- Super easy import
- Optimized for smaller models on embedded devices (Raspberry Pi)
- Great CLI Test and Benchmarking Tool

- CPU only
- No no_std support

# ORT (pyke.io)

- Wrapper around ONNX Runtime (C++)
- Support 100% of ONNX operators
- Super easy to use with prebuilt static libraries
- Supports all GPUs and many accelerators
- Super fast on CPUs and embedded
- Battle tested and optimized

- C++ wrapping can make things harder
- Builds for accelerators can be tricky

# ORT - Execution Providers

| EP | Cargo feature | Supported | Binaries |
|---|---|---|---|
| NVIDIA CUDA ↗ | `cuda` | 🔷 | ✅ |
| NVIDIA TensorRT ↗ | `tensorrt` | 🔷 | ✅ |
| Microsoft DirectML ↗ | `directml` | 🔷 | ✅ |
| Apple CoreML ↗ | `coreml` | 🔷 | ✅ |
| AMD ROCm ↗ | `rocm` | 🔷 | ❌ |
| Intel OpenVINO ↗ | `openvino` | 🔷 | ❌ |
| Intel oneDNN ↗ | `onednn` | 🔷 | ❌ |
| XNNPACK ↗ | `xnnpack` | 🔷 | ✅ |
| Qualcomm QNN ↗ | `qnn` | 🔷 | ❌ |
| Huawei CANN ↗ | `cann` | 🔷 | ❌ |
| Android NNAPI ↗ | `nnapi` | 🔷 | ❌ |
| Apache TVM ↗ | `tvm` | 🔷 | ❌ |
| Arm ACL ↗ | `acl` | 🔷 | ❌ |
| ArmNN ↗ | `armnn` | 🔷 | ❌ |
| AMD MIGraphX ↗ | `migraphx` | 🔷 | ❌ |
| AMD Vitis AI ↗ | `vitis` | 🔷 | ❌ |
| Rockchip RKNPU ↗ | `rknpu` | 🔷 | ❌ |
| WebGPU | `webgpu` | 🔷 | ✅ |
| Microsoft Azure ↗ | `azure` | 🔷 | ❌ |

# ORT - Upcoming features (v2.0.0-rc.10)

- Alternative Backends (Tract / Candle)
- Model Editor API (build models programmatically)
- no_std support
- WebGPU support with prebuilt-libraries
- Static linking of CUDA and TensorRT libraries

# Workshop Overview

1. **Part 0: Introduction to the Team & Rust for Edge A**
2. **Part I: Lecture on Computer Vision**
   a. Main problems in Computer Vision
   b. What exactly is a neural network? (CNNs / Transformers)
   c. What exactly is an image embedding?
   d. Computer vision on the Edge
3. **Hands-On I: Air-gapped Face recognition on the P**
4. **Part II: Lecture on Natural Language Processing**
   a. A bit of history & development of modern LLMs
   b. How does an LLM work? Tokenizers, pretraining, post-training
   c. Context Engineering: Tool calling, RAG
   d. Libraries: tokenizers-rs, llama.cpp
5. **Hands-On II: Chat with a LLM on Pi**

# Workshop Overview

1. **Part 0: Introduction to the Team & Rust for Edge A**
2. **Part I: Lecture on Computer Vision**
    a. Main problems in Computer Vision
    b. What exactly is a neural network? (CNNs / Transformers)
    c. What exactly is an image embedding?
    d. Computer vision on the Edge
3. **Hands-On I: Air-gapped Face recognition on the P**
4. **Part II: Lecture on Natural Language Processing**
    a. A bit of history & development of modern LLMs
    b. How does an LLM work? Tokenizers, pretraining, post-training
    c. Context Engineering: Tool calling, RAG
    d. Libraries: tokenizers-rs, llama.cpp
5. **Hands-On II: Chat with a LLM on Pi**

# Workshop Overview

1. Part 0: Introduction to the Team & Rust for Edge A
2. **Part I: Lecture on Computer Vision**
    a. Main problems in Computer Vision
    b. What exactly is a neural network? (CNNs / Transformers)
    c. What exactly is an image embedding?
    d. Computer vision on the Edge
3. Hands-On I: Air-gapped Face recognition on the P
4. Part II: Lecture on Natural Language Processing
    a. A bit of history & development of modern LLMs
    b. How does an LLM work? Tokenizers, pretraining, post-training
    c. Context Engineering: Tool calling, RAG
    d. Libraries: tokenizers-rs, llama.cpp
5. Hands-On II: Chat with a LLM on Pi