Edge AI and Embedded ML

CSE/ECE 475

Jake Garrison, Shwetak Patel

Embedded ≠ Edge ≠ On-device

- Embedded ML isn't just "on-device ML"

Category	Examples	Embedded ML?
Microcontrollers (MCUs)	STM32,	✅
Low-end SoCs	ESP32, nRF52	✅
Raspberry Pi	Pi 3/4 with Linux OS	❌
Smartwatches	Apple Watch, Google Pixel	❌

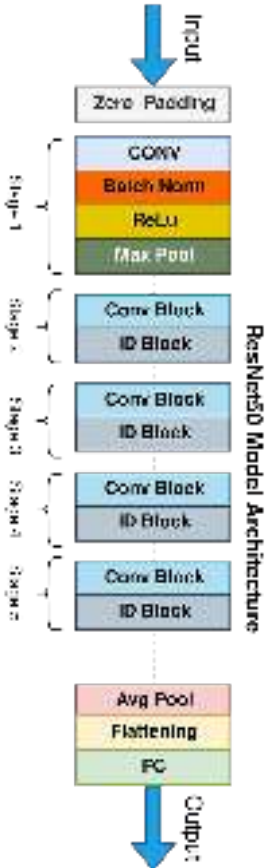
For a larger list of options grouped by vendor, see GitHub Notes: Popular Microcontrollers

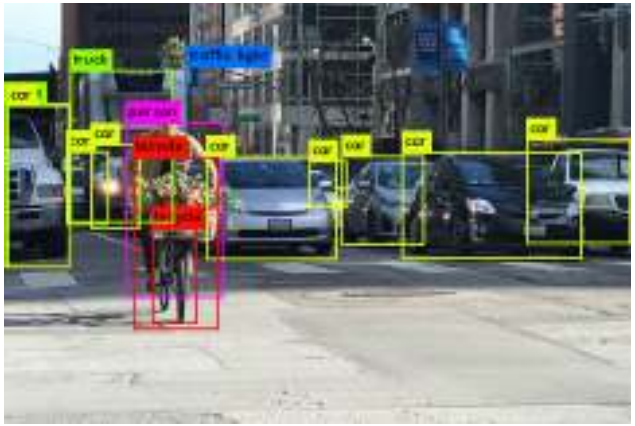
Why do we want embedded ML?

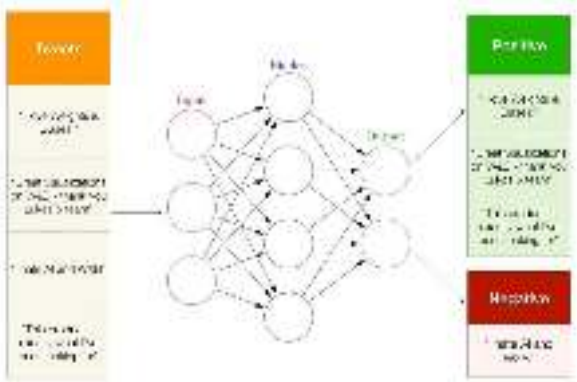
- Privacy
  - Keeps sensitive data (e.g., voice, video, health) on the device
- Low Latency
  - e.g., noise cancelling on earphones
- Offline Access
  - Works without internet (e.g., environmental sensing)
- Efficiency
  - Reduces data transfer, saving bandwidth and power
- Cost Saving
  - Minimizes cloud compute costs

Machine Learning Ecosystem

Classification Based Machine Learning







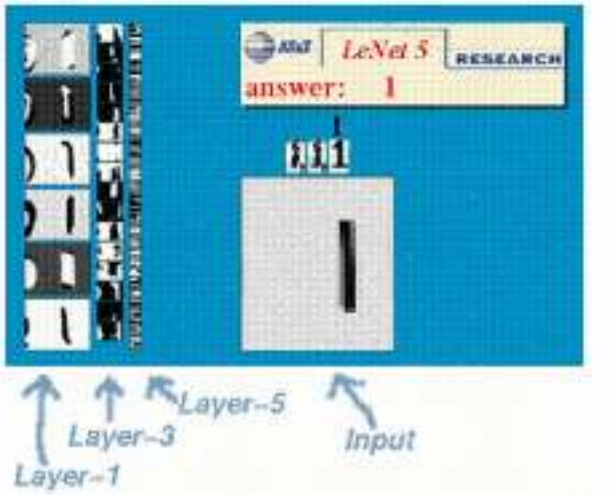
Characteristics:

- Relatively small
- Single execution
- Static shapes

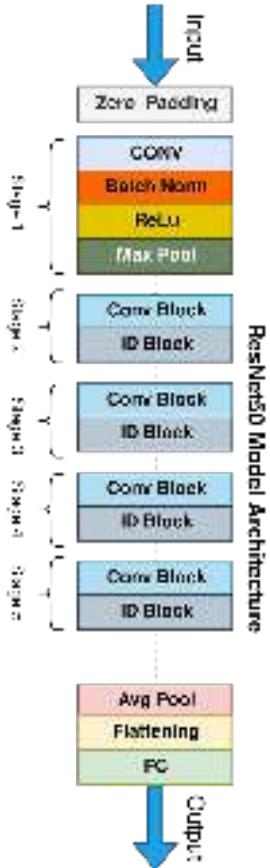
Benefits:

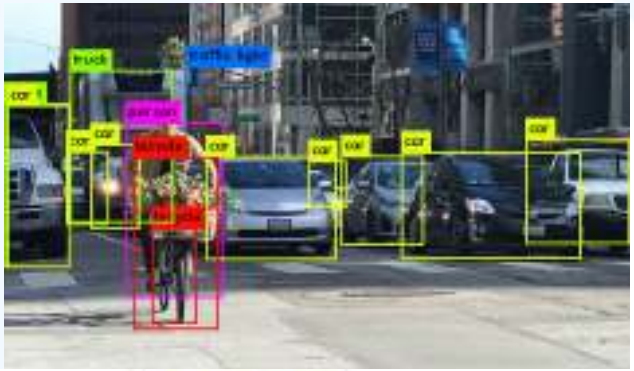
- Easy to deploy on any device
- Superb at targeted tasks.
- Cheap to create

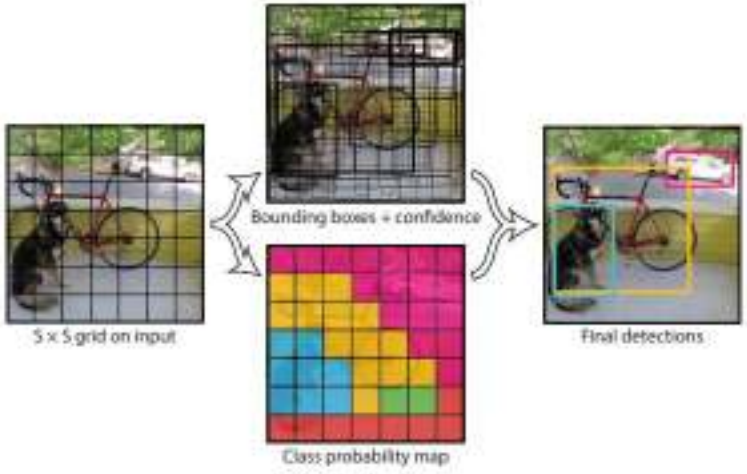
Image Classification Based Machine Learning



handwritten digits recognition in 1989







YOLO (You Only Look Once) [2015]

You Only Look Once:
Unified, Real-Time Object Detection

Joseph Redmon*, Santosh Divvala*†, Ross Girshick¶, Ali Farhadi*†
University of Washington*, Allen Institute for AI†, Facebook AI Research¶
http://pjreddie.com/yolo/

Abstract

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

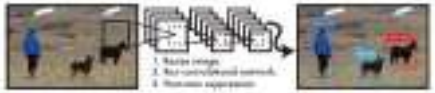


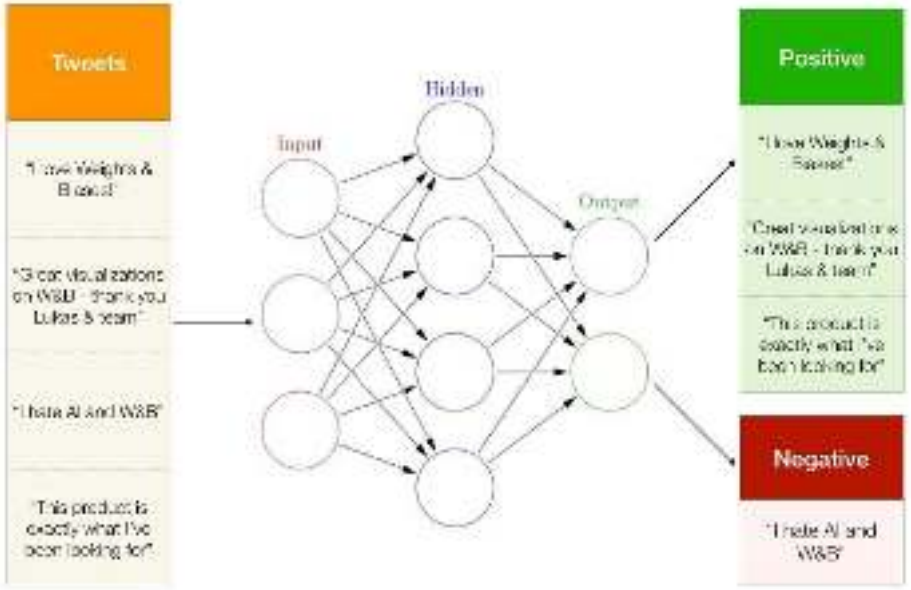
Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Table 1: Real-Time Systems on PASCAL VOC 2007. Comparing the performance and speed of fast detectors. Fast YOLO is the fastest detector on record for PASCAL VOC detection and is still twice as accurate as any other real-time detector. YOLO is 10 mAP more accurate than the fast version while still well above real-time in speed.

Also see MobileNet, EfficientNet

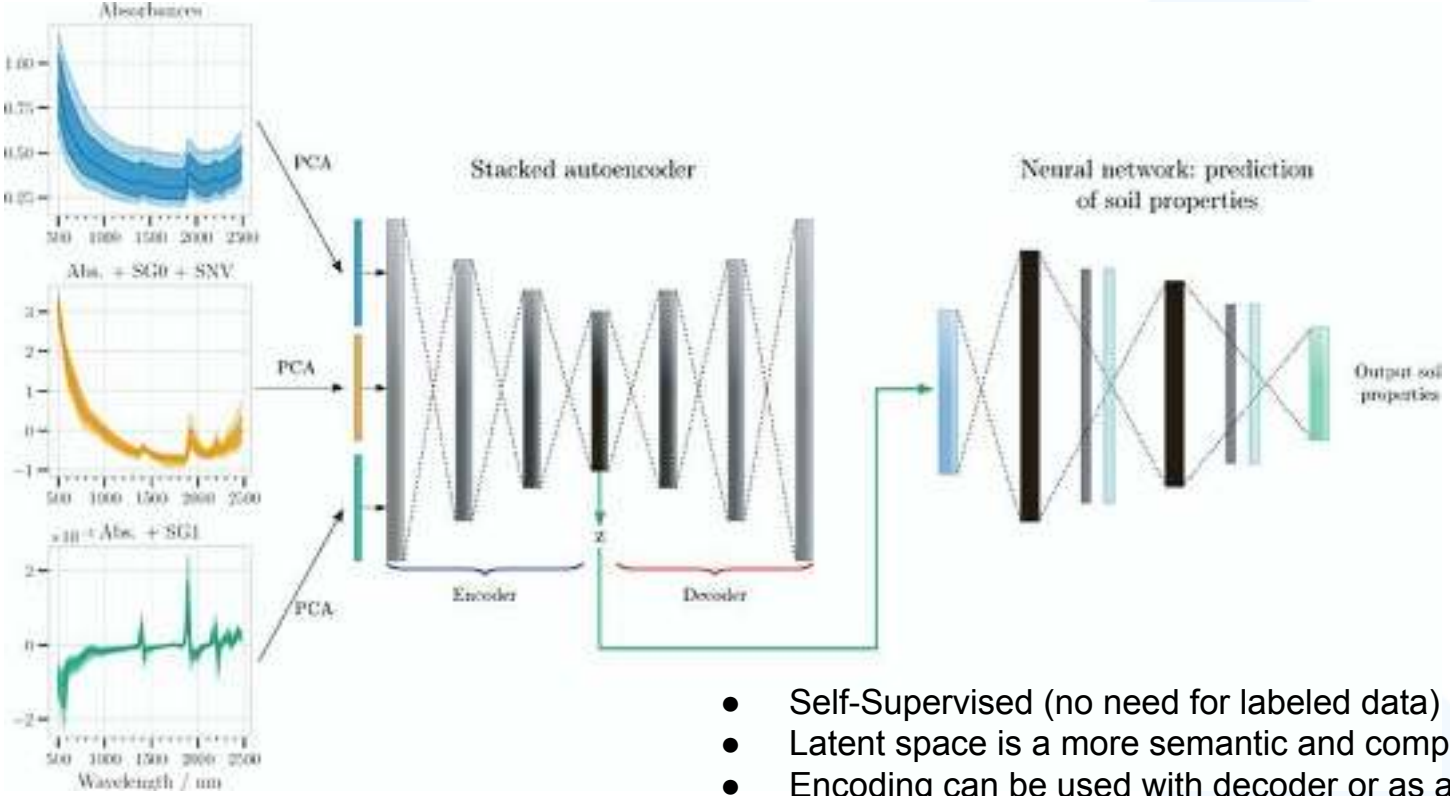
Sentiment Analysis [2015]



The food is flavorful, plentiful and reasonably priced.
The atmosphere is relaxed and casual. It's a great place
to order from or sit-in.

Ground Truth: RESTAURANT#GENERAL
Prediction:
RESTAURANT#GENERAL (93.16%)

Representation Learning



- Self-Supervised (no need for labeled data)
- Latent space is a more semantic and compressed representation
- Encoding can be used with decoder or as an input to other tasks

Image source: https://www.sciencedirect.com/science/article/abs/pii/S0016706121000410

Masked Autoencoders Are Scalable Vision Learners [2021]

Masked Autoencoders Are Scalable Vision Learners

Kaiming He* Xinlei Chen* Saining Xie Yanghao Li Piotr Dollár Ross Girshick

*equal technical contribution †project lead

Facebook AI Research (FAIR)

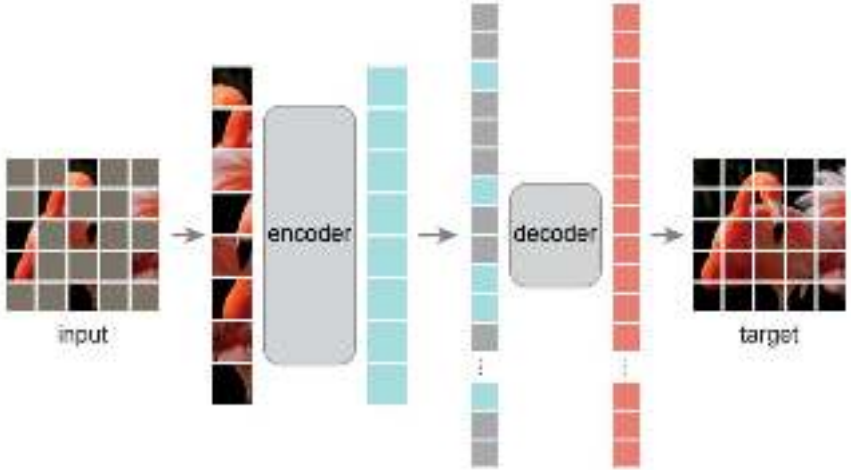


Figure 1. **Our MAE architecture.** During pre-training, a large random subset of image patches (*e.g.*, 75%) is masked out. The

Figure 10. **Uncurated random samples** on ImageNet

https://github.com/facebookresearch/mae

Masked Autoencoders that Listen 2023

Masked Autoencoders that Listen

Po-Yao Huang[1] Hu Xu[1] Juncheng Li[2] Alexei Baevski[1]
Michael Auli[1] Wojciech Galuba[1] Florian Metze[1] Christoph Feichtenhofer[1]

[1]Meta AI [2]Carnegie Mellon University

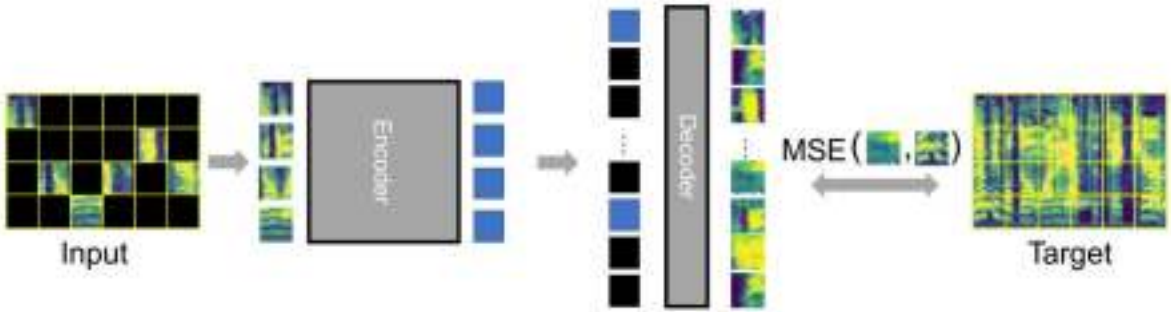
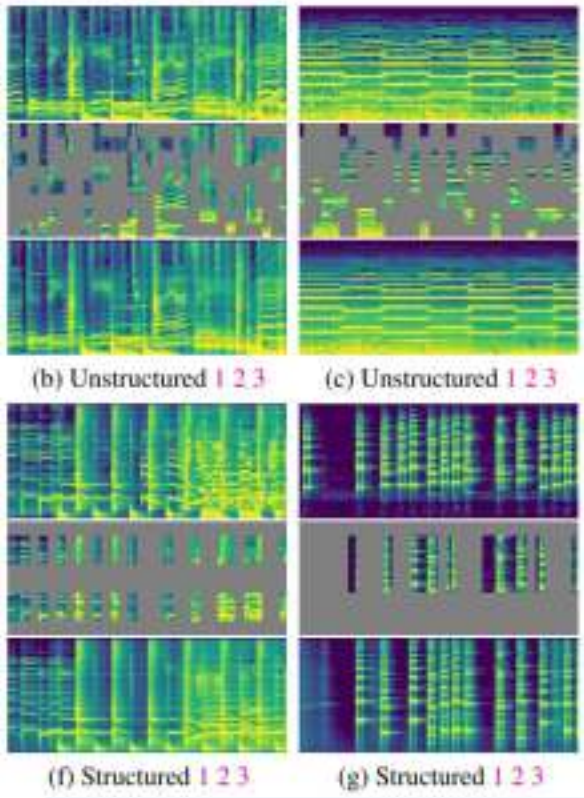
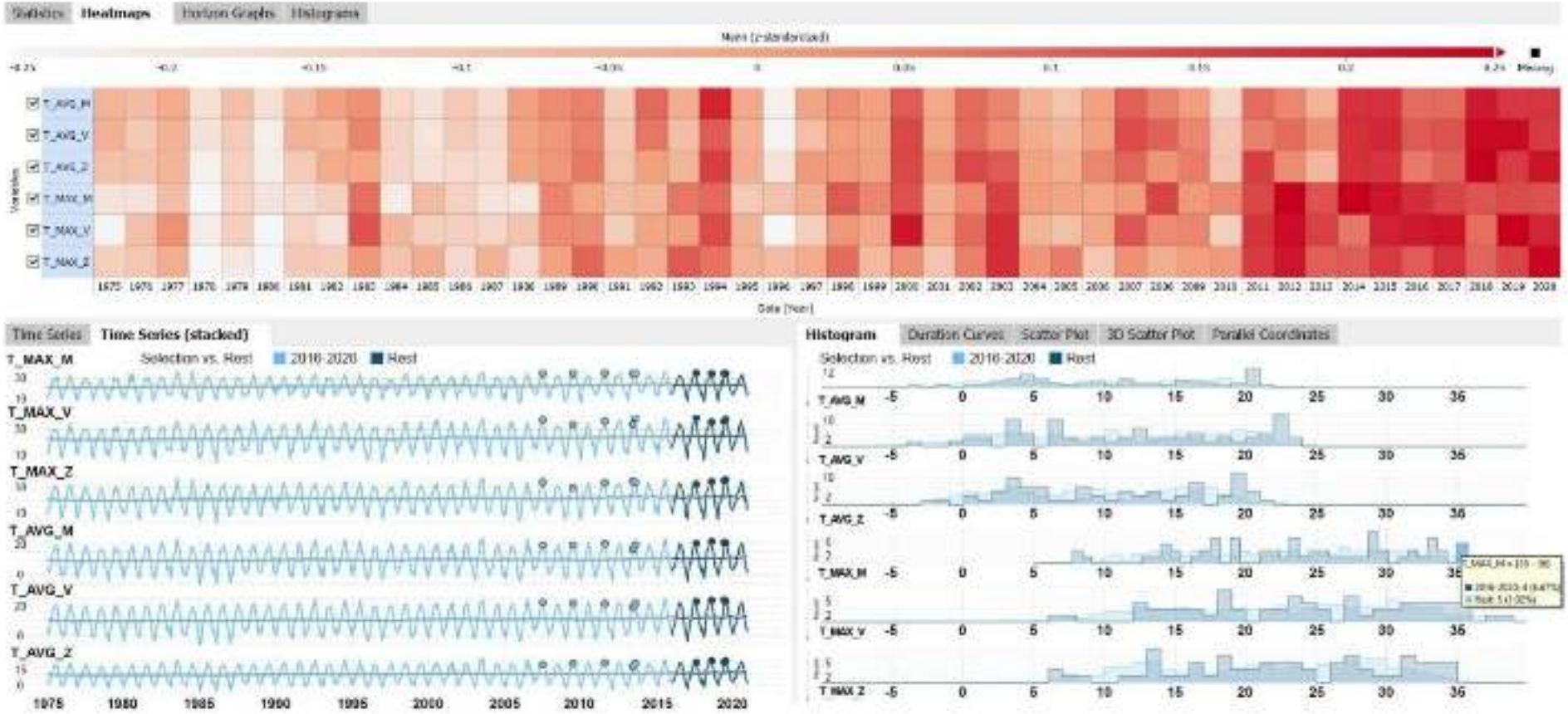


Figure 1: **Audio-MAE for audio self-supervised learning**. An audio recording is first transformed into a spectrogram and split into patches. We embed patches and mask out a large subset (80%). An encoder then operates on the visible (20%) patch embeddings. Finally, a decoder processes the order-restored embeddings and mask tokens to reconstruct the input. Audio-MAE is minimizing the mean square error (MSE) on the masked portion of the reconstruction and the input spectrogram.



https://github.com/facebookresearch/AudioMAE

Masked Autoencoders *for any multimodal sequences*



Embedded Use Case?

Auto Encoders are typically too large to run on the edge

But…encoder and decoder can live on different machines

- Encoder can be designed to be small and **compress** lots of raw sensor data into a representation embedding
- Large models can live **in the cloud** and decode the embedding to reconstruct or do downstream tasks
- Privacy and bandwidth benefits

Generative AI

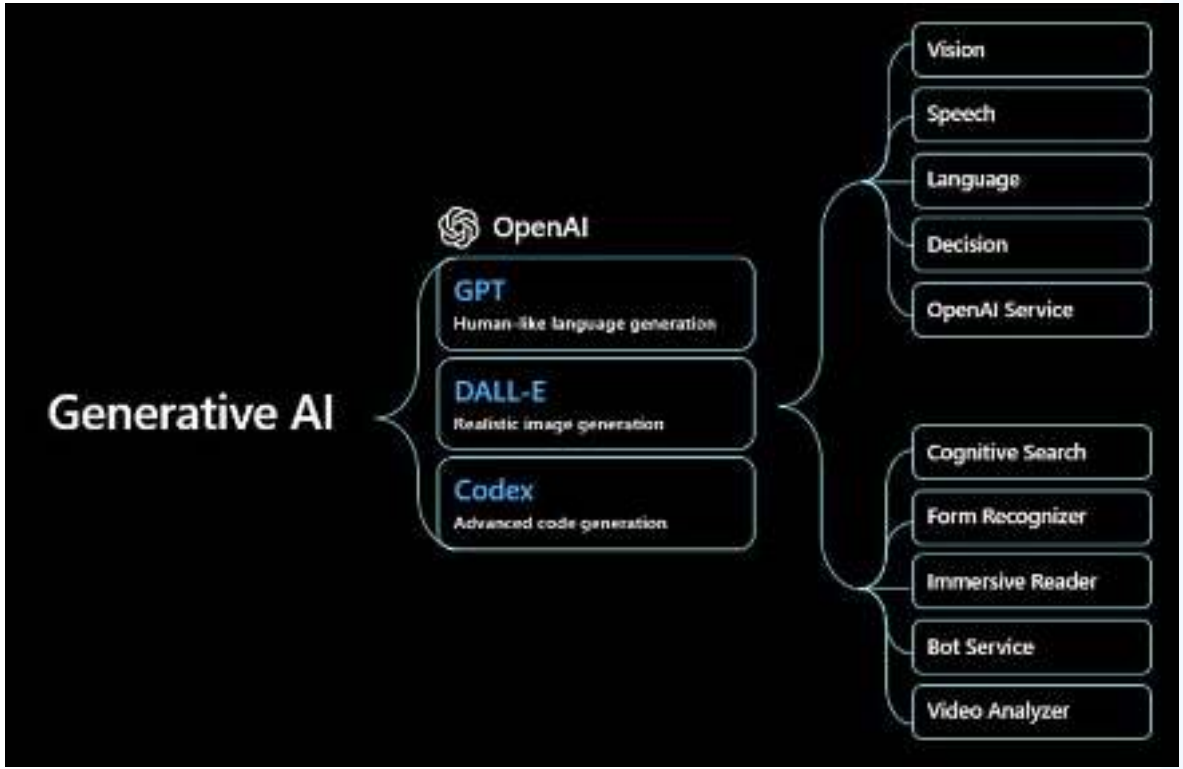
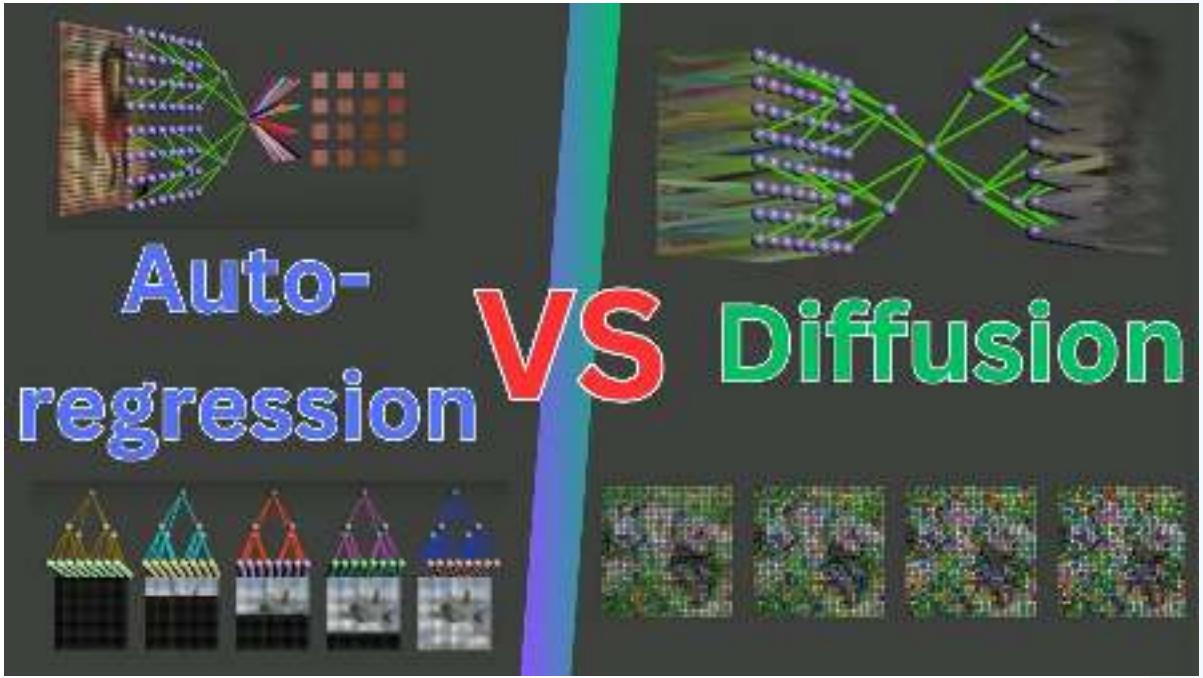


Image Generation



Or combine aspects of **both**:

- Stable diffusion use AR Encoder for text, diffusion for image gen
- Post training LLMs with diffusion and RL policies (DiFFPO)
- Time series Forecasting use AR for modeling trend and diffusion for modeling noise

Image source: [CPU Bandwidth Vs Memory Bandwidth]

Text Generation

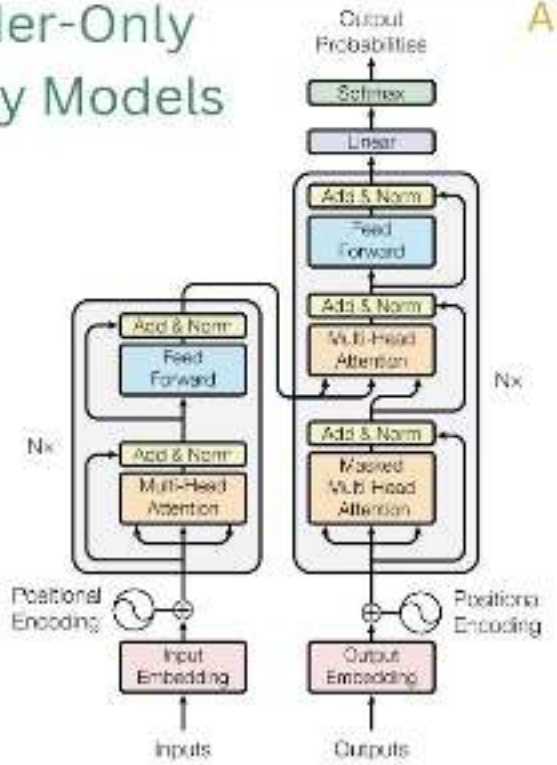
Attention Is All You Need

Decoding Encoder-Only and Decoder-Only Models

And Question About Transformers

BERT

Encoder



GPT

Decoder

Google

OpenAI ChatGPT

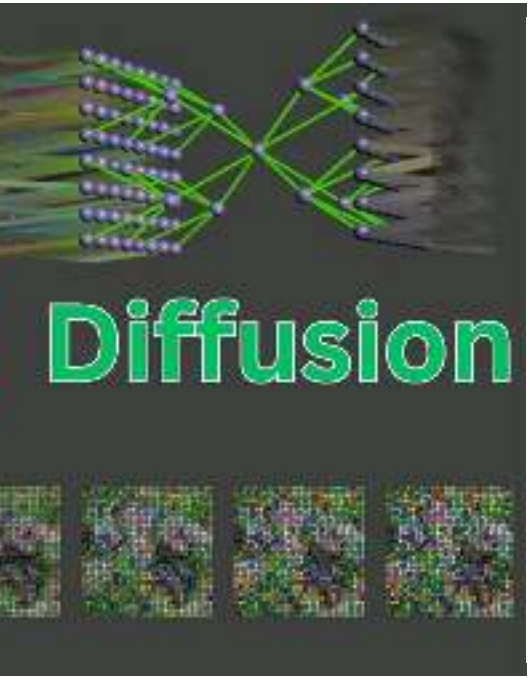
Auto Regressive Text Generation - Llama 3.1

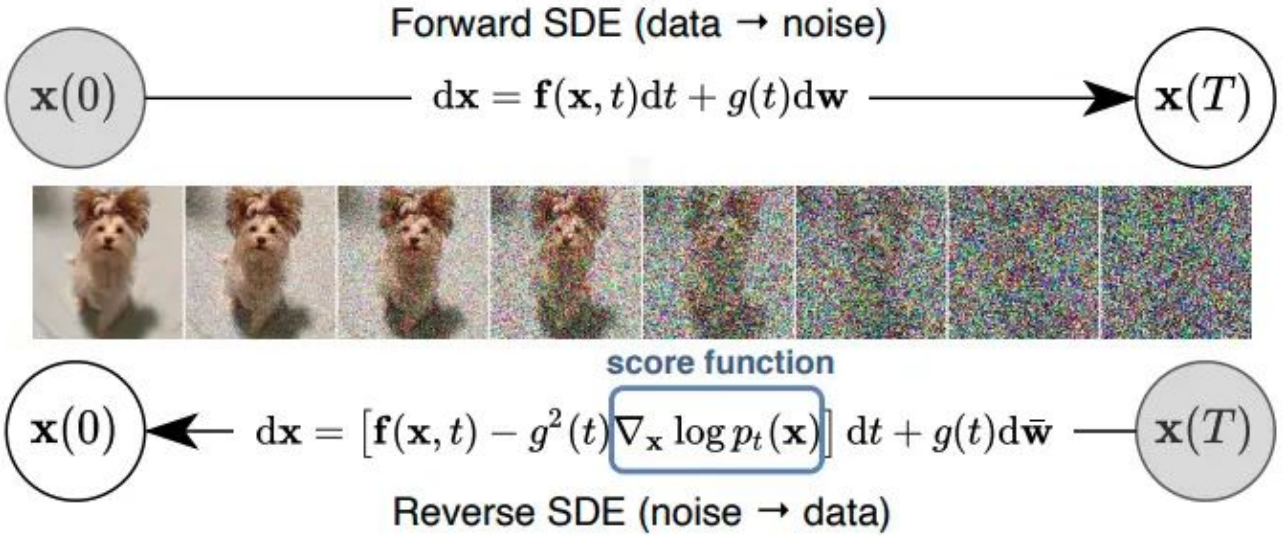
We believe there are three key levers in the development of high-quality foundation models: data, scale, and managing complexity. We seek to optimize for these three levers in our development process:

- **Data.** Compared to prior versions of Llama (Touvron et al., 2023a,b), we improved both the quantity and quality of the data we use for pre-training and post-training. These improvements include the development of more careful pre-processing and curation pipelines for pre-training data and the development of more rigorous quality assurance and filtering approaches for post-training data. We pre-train Llama 3 on a corpus of about 15T multilingual tokens, compared to 1.8T tokens for Llama 2.

- **Managing complexity.** We make design choices that seek to maximize our ability to scale the model development process. For example, we opt for a standard dense Transformer model architecture (Vaswani et al., 2017) with minor adaptations, rather than for a mixture-of-experts model (Shazeer et al., 2017) to maximize training stability. Similarly, we adopt a relatively simple post-training procedure based on supervised finetuning (SFT), rejection sampling (RS), and direct preference optimization (DPO; Rafailov et al. (2023)) as opposed to more complex reinforcement learning algorithms (Ouyang et al., 2022; Schulman et al., 2017) that tend to be less stable and harder to scale.

Image Generation

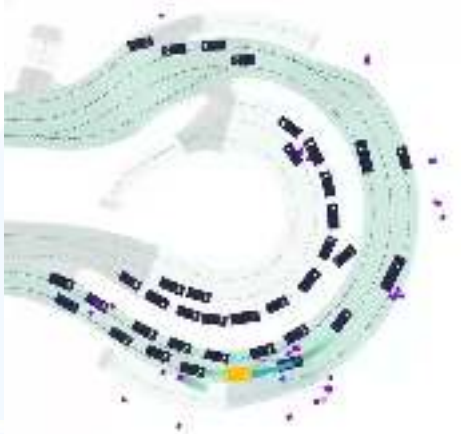


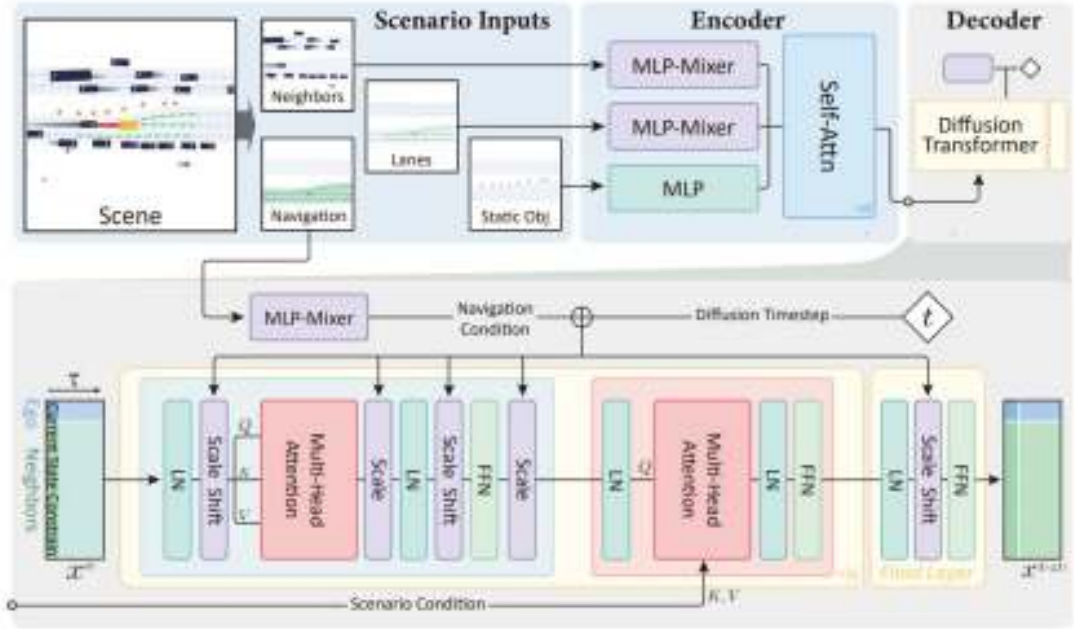


Diffusion Based Control / Planning

DIFFUSION-BASED PLANNING FOR AUTONOMOUS DRIVING WITH FLEXIBLE GUIDANCE

Yinan Zheng[1*], Ruiming Liang[2,3†], Kexin Zheng[1*], Jinliang Zheng[1], Liyuan Mao[2],
Jianxiong Li[1], Weihao Gu[4], Rui Ai[4], Shengbo Eben Li[1], Xianyuan Zhan[1,5†], Jingjing Liu[1]
[1] Tsinghua University [2] Institute of Automation, Chinese Academy of Sciences
[3] The Chinese University of Hong Kong [4] Shanghai Jiao Tong University





https://zhengyinan-air.github.io/Diffusion-Planner/

When do we not want embedded ML?

e.g., Generative AI:

- Absolutely massive and complex
- Extremely expensive to create and run

Slide modified from Josh Fromm, 2023

Training versus Inference

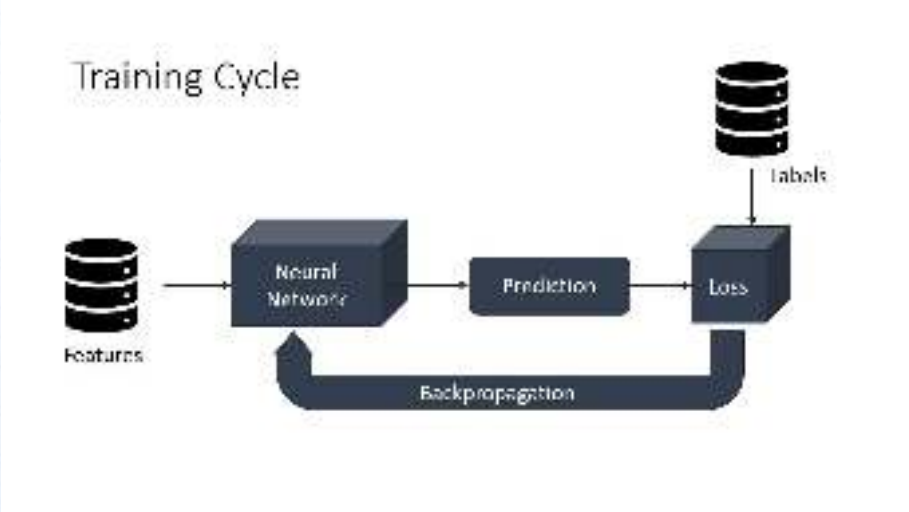
Training:

- Focus is on creating good datasets for a task.
- Quality is more important
- Only working on GPUs is ok.

Inference:

- Optimization leads to huge savings.
- Matters for embedded or edge

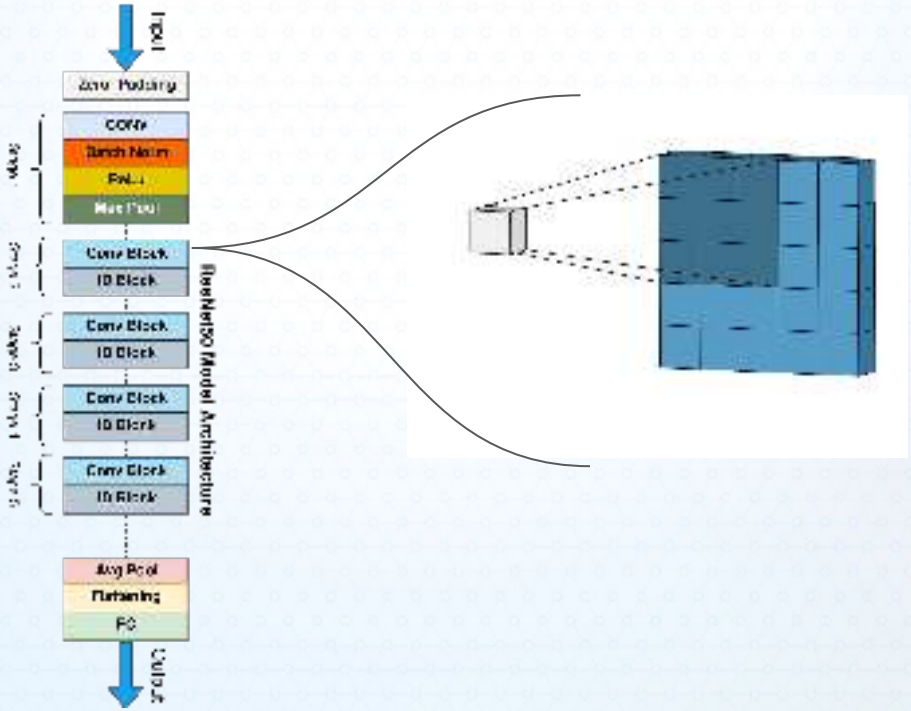
At scale, chips are designed differently for training versus serving



Slide modified from Josh Fromm, 2023

Anatomy of Model

Image Model Anatomy

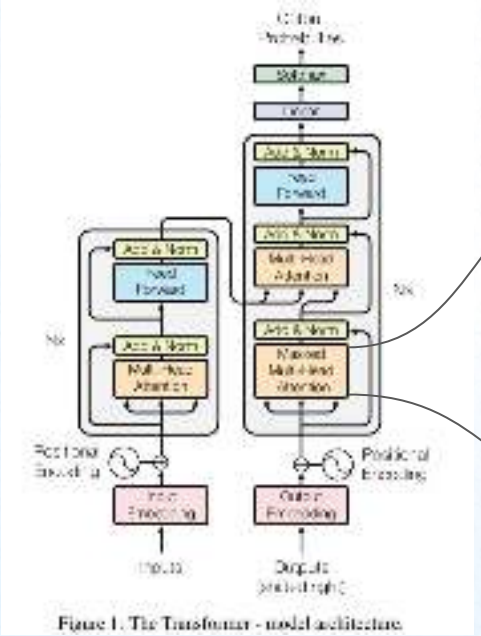


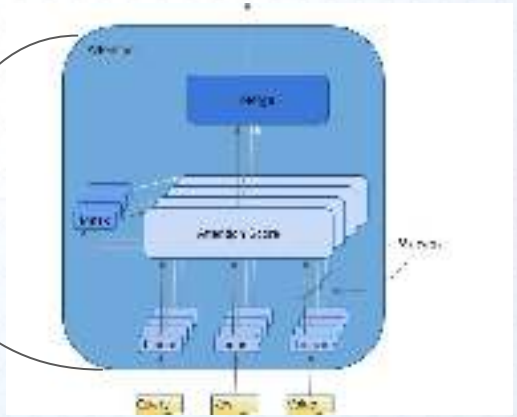
Important to understand how data flows through the model

For a fully connected (dense) layer:
output = Wx + b

Slide modified from Josh Fromm, 2023

NLP Model Anatomy





Attention(Q,K,V)=softmax(QK^T)V

Even the latest models are essentially piles of matrix multiplication.

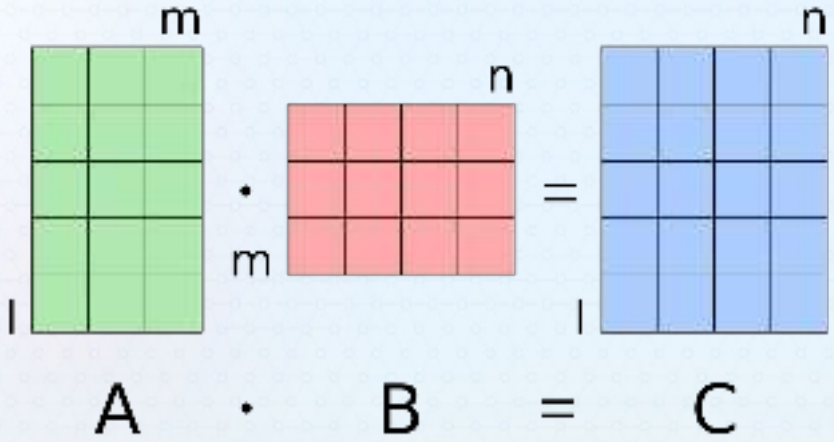
Slide modified from Josh Fromm, 2023

The All Powerful Matrix Multiply

Optimizing models often reduces to optimizing MatMul.

What makes this difficult?

- Compute Requirements
- Massive Movement



Slide modified from Josh Fromm, 2023

Optimizing Matrix Multiplication

Colab: Matrix Multiplication Notebook [broken link?]

- Loop Reordering
- Vectorization
- Parallelization / Multi-threading

Note: Embedded SDKs have their own DSP / Linear Algebra libraries and data types that should be used

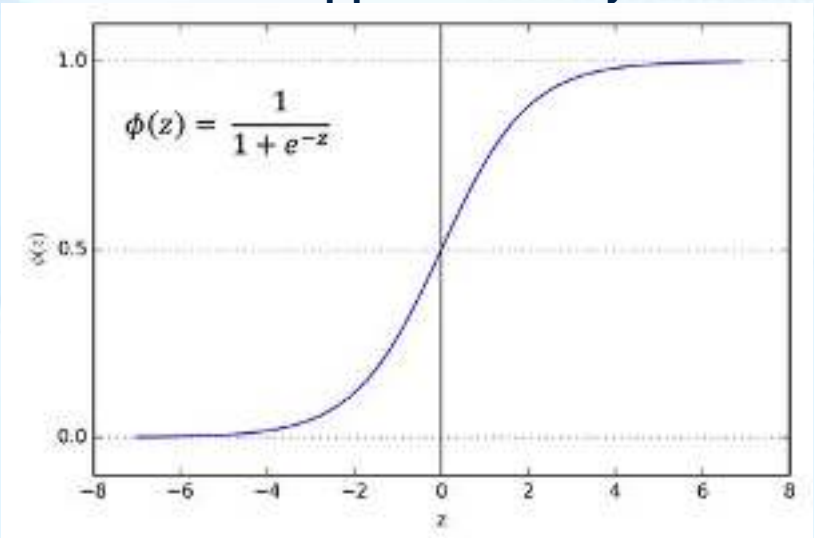
Slide modified from Josh Fromm, 2023

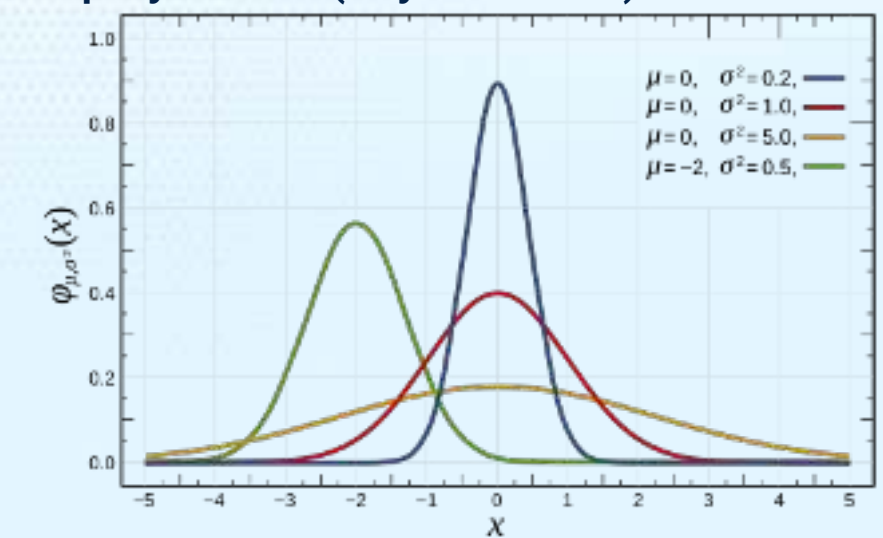
Non-Linear Operation

Non-linear like vacuum tubes, transistors, neurons which have activation energy, saturation

Current hardware uses approximations because:

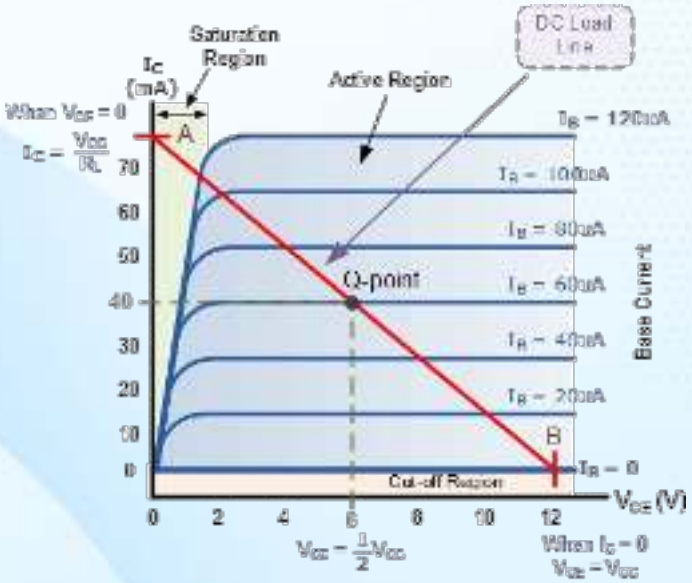
- Computing e^x or 1/(1+e^−x) exactly is very slow
- You can approximate by a low-order polynomial (Taylor Series)



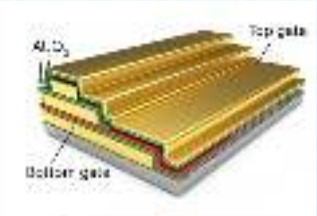


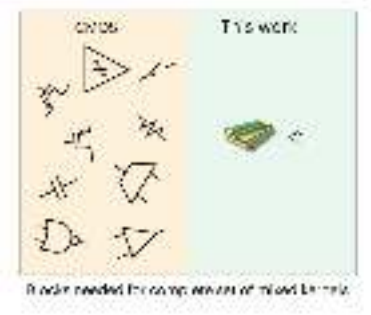
Optimizing Non-Linear Operation

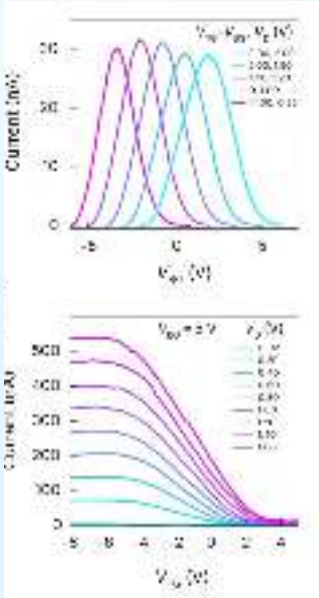
Current transistor



Reconfigurable Analog Transistor / Neuromorphic Computing





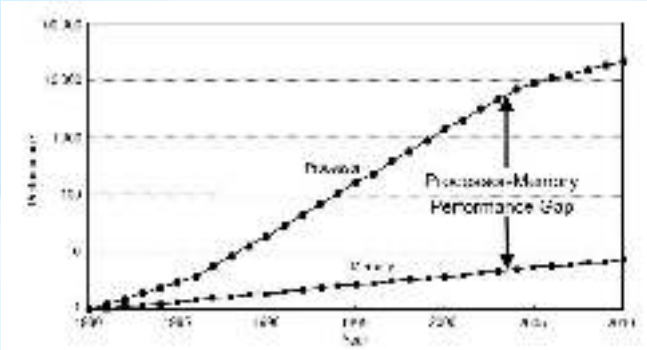


Reconfigurable mixed-kernel heterojunction transistors for personalized support vector machine classification | Nature Electronics

Optimizing Memory Access and Bandwidth

For CPUs or other accelerators (GPU)
Moving data between memory and compute units is a bottleneck
Solution:

- L1, L2… Cache (Intel CPU usually <1 MB)
- Unified Memory Architecture (Apple M-series, AMD)
- In-Memory Computing
    - Resistive RAM arrays can do multiplication



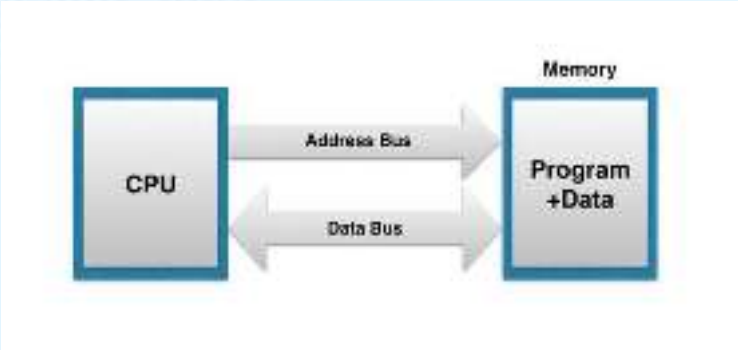


Image source: [CPU Bandwidth Vs Memory Bandwidth]

Embedded ML

- Efficiency was already a goal for server/cloud ML
- Embedded ML pushes efficiency to the extreme

Embedded ML

We want our models to be:

- Low Latency / Fast Inference
  - Real-time applications (e.g., wake word detection, object tracking)

- Low Power Consumption
  - Important for battery-operated devices like phones and wearables

- Efficient Use of Compute & Memory
  - Devices like MCUs and IoT sensors have limited RAM and CPU

Embedded ML

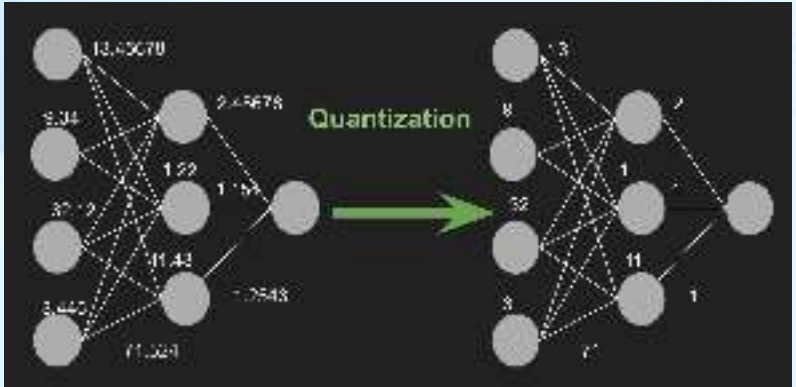
Core Strategies:

- Small models!
- Robust performance on small models!
- Small input size!
- Use hardware accelerated operations

Quantization

Reduce model size by converting weights, for example, from 32-bit floats to 8-bit integers (int8 is 1/4 the size of float32):

- Compression & Memory savings: ~1/4 in model size
- Speedup: Up to 2–4x faster inference
- Accuracy loss: can be minimal with quantization-aware training (QAT)
- Can be done post-training (less effectively)



Pruning

Removing unimportant parameters:

- Compression & Memory savings: -1/2 or less typically
- Speedup: can result in 2–3x
- Accuracy loss: typically <2% if done carefully from my experience

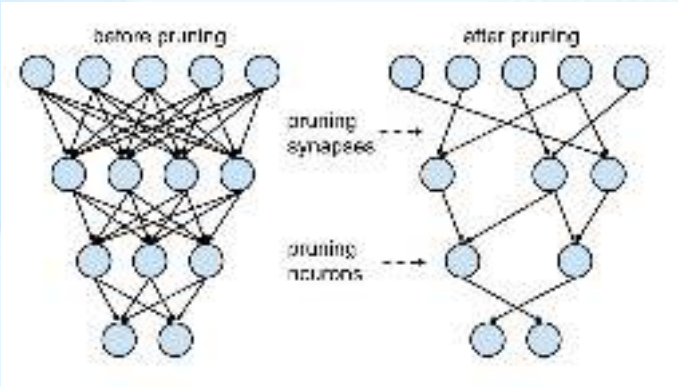


Image source: [Pruning in Deep Learning Model. Pruning in deep learning basically used… | by Souvik Paul | Medium]

Knowledge Distillation

Training a smaller "student" model from a larger "teacher" model

- Compression & Memory savings: ?? varies
- Accuracy loss: minimal
- Note: this is a (re-)training job, means it requires a lot of computing

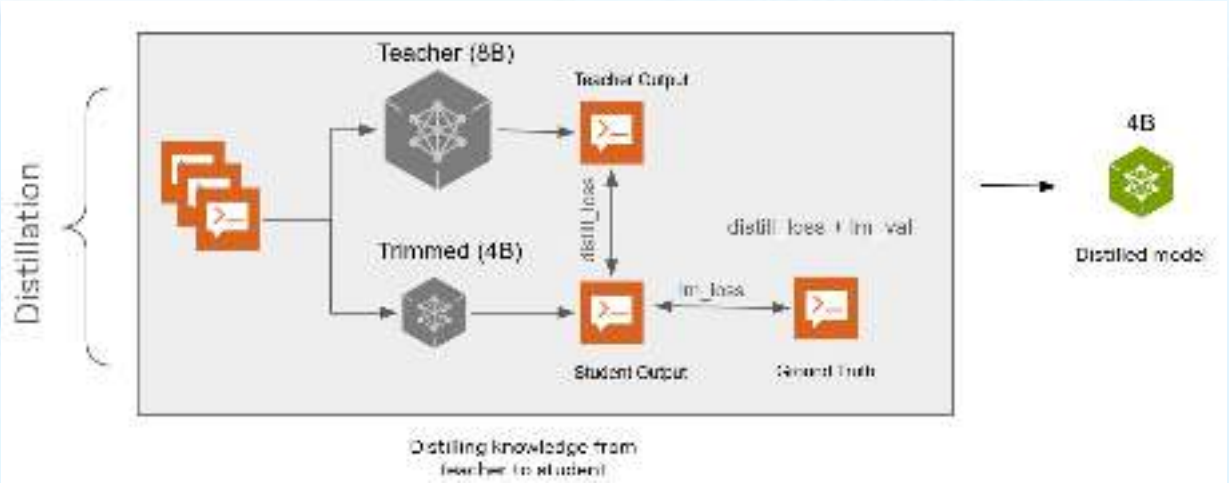


Image source: [LLM Model Pruning and Knowledge Distillation with NVIDIA NeMo Framework | NVIDIA Technical Blog]

More

- Streaming operations (for CNNs or token based models VLM/LLM)
  - Separable convolutions
- Vector quantized embeddings (VQ-VAE)
- Low-Rank Factorization
  - Decomposing weight matrices to reduce computational requirements

- Architecture Ideas
  - MobileNet (Google)
  - TinyML (Song Han, MIT)
    - e.g., MCUNet
  - SqueezeNet
  - YOLO-Pico
  - Coral On-Device Examples
  - Search ESP32 on HackerNews or Github for ideas

Tools and Frameworks:

AI MCU github notes

- LiteRT (formerly known as TensorFlow Lite)
  - Convert a model into the FlatBuffers format (.tflite) and run them in LiteRT. Support: TensorFlow, PyTorch, JAX
  - Integrate the model into your app. Support: Android (JAVA), iOS (Swift), Micro (embedded devices using C)

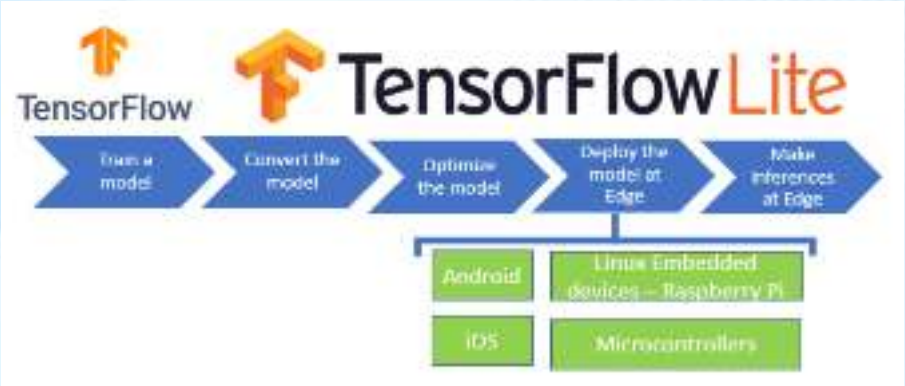


Image source: Everything about TensorFlow Lite and start deploying your machine learning model - Latest News from Seeed Studio

Tools and Frameworks:

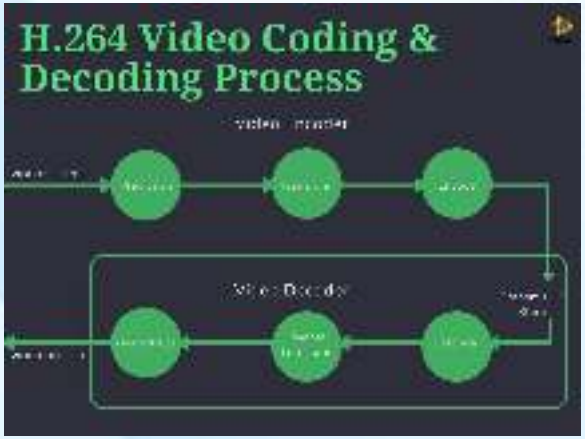
LiteRT Micro only supports a subset of operations due to memory and code size constraints. Unsupported Ops:

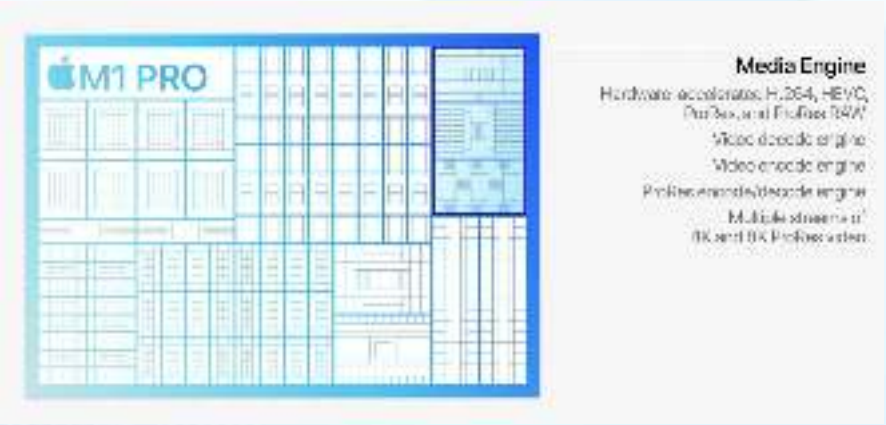
- Conv 3D
- LSTM
- Attention

  - **Supported MCUs**
  - **Examples**

Edge AI

Video encoding is a demanding algorithm, however its been made trivial by silicon.



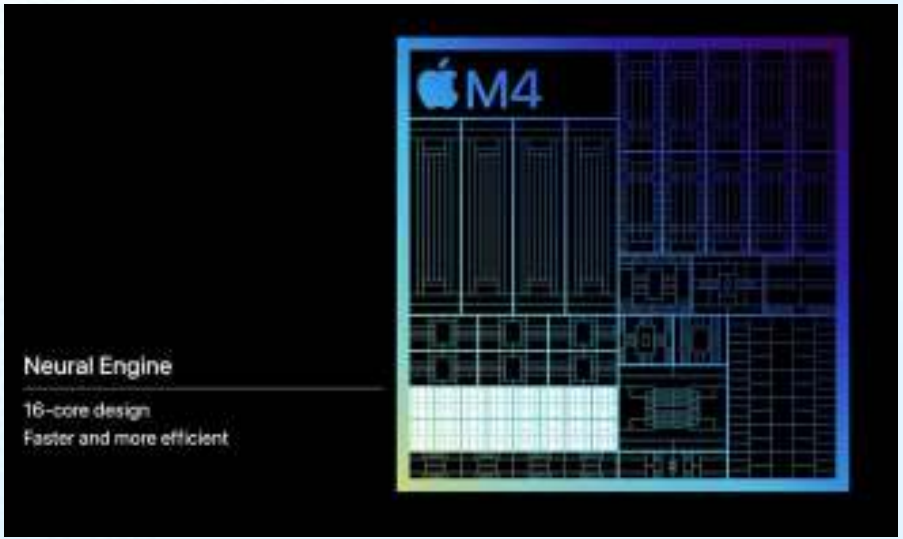


Slide modified from Josh Fromm, 2023

Edge AI

AI chips will become similarly ubiquitous and commoditized (e.g., Apple H1, Qualcomm)

Most embedded systems will likely include one (e.g., AD MAX78000, TI).



Slide modified from Josh Fromm, 2023

Conclusion

Though ML applications evolve rapidly, their core components have only gotten simpler.

Now is a fantastic time to work at the intersection of machine learning and hardware.

Slide modified from Josh Fromm, 2023

Demos

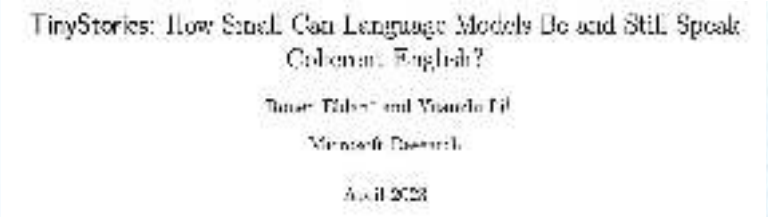
Jake Garrison, Alexander Metzger, Jiuyang Lyu

Arduino IMU Playground

- [On-Device ML Playground]

ESP LLM

This project ports the Llama 2 transformer architecture to the ESP32-S3 using a **tiny** llama.c based LLM **trained on children's stories**


TinyStories: How Small Can Language Models Be and Still Speak Coherent English?

Ronen Eldan and Yuanzhi Li

Microsoft Research

April 2023

- Code: [ESP-32-LLM github]
  a. Very small vocab
  b. Short term memory
  c. Use ESP matrix algebra
- Based on llama2.c by Andrej Karpathy and on esp32-llm by Dave Bennet

Model Configuration

The current model uses these parameters:

```
--vocab_source=custom
--vocab_size=512
--dim=64
--n_layers=4
--n_heads=4
--n_kv_heads=4
--multiple_of=4
--max_seq_len=128
--batch_size=128
```

This is the **SMALLEST** from the TinyStories Paper

Project Structure

- `src/llm.c` - Main LLM implementation
- `src/llm.h` - Header file with data structures and function declarations
- `src/main.c` - ESP32 application entry point
- `components/` - External components and dependencies

Performance

Current performance metrics:

- Inference speed: ~17 tokens/second
- Memory efficiency: Uses optimized data structures and FreeRTOS tasks

Memory Usage

- **Flash (SPIFFS):** ~1.05 MB (Model + Tokenizer files)
- **PSRAM (Heap):**
  - ~1.0 MB for Model Weights (static).
  - ~1.0 MB for RunState/KV-Cache (dynamic, allocated at startup).
  - Total PSRAM required: ~2.5 MB (leaving headroom for ESP-IDF overhead).

ESP32 Port: micro-llm VS Original llama2.c

More details in README: [ESP-32-LLM github]

Matrix Algebra

The matmul function, the most performance-critical part, is heavily optimized.

1. **Hardware-Accelerated Dot Product:** The O(n) for loop for dot products is replaced by a single call to the esp-dsp library, which uses the ESP32-S3's vector instructions.
   - **Original `run.c`:**
     ```
     for (int j = 0; j < n; j++) {
         val += w[i * n + j] * x[j];
     }
     ```
   - **ESP32 Port (`llm.c`):**
     ```
     dsps_dotprod_f32_ae32(w, x, &val, n);
     ```
2. **RTOS-based Parallelization:** Instead of OpenMP's `parallel for`, the port uses a persistent FreeRTOS task (`matmul_task`) pinned to Core 1. The `matmul` function on Core 0 computes the first half of the work, signals the task via a semaphore to compute the second half, and then uses an event group to sync both cores before returning.

Multi-Head Attention (Forward)

The same manual parallelization pattern is applied to the multi-head attention loop. The `for` loop over heads is split, with Core 0 and a dedicated `forward_task` on Core 1 each processing half the workload concurrently.

- **Pinned Tasks:** Specific compute-heavy tasks (`matmul_task`, `forward_task`) are pinned to Core 1, while the main application logic runs on Core 0.

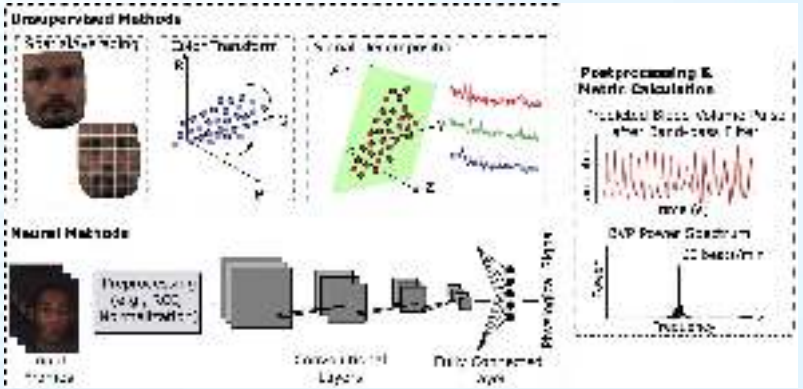
Memory

- **Vocabulary Size: 512 tokens.** Standard LLMs use 32k to 100k. A standard vocab (32000) with a dimension of 64 would require a 7GME embedding table alone. By reducing vocab to 512, the embedding table and final classifier weights are negligible in size.
- **Context Window: 512 tokens.** This is the model's short-term memory. It can recall and attend to the previous 512 tokens generated or provided in the prompt.
  - **Model Weights:** The 1.8MB model file is read from SPIFFS storage and loaded entirely into external PSRAM at boot.
  - **RunState (KV Cache):** The dynamic memory required during inference (activations and the key-value cache) is allocated in PSRAM.
- **From Memory-Mapping to Direct Loading:** The original `mmap` approach is unavailable. The port reads the model file into memory. The ESP32 port replaces this by reading the model from a SPIFFS filesystem directly into a buffer in external PSRAM.
  ```
  // Original: mmap() approach
  data = mmap(NULL, file_size, PROT_READ, MAP_PRIVATE, fd, 0);

  // ESP32 Port: Direct load from SPIFFS to PSRAM
  data = (float*)ps_malloc(file_size);
  fread(data, 1, file_size, file);
  ```
- **Compatibility Shims:** To minimize code changes, POSIX functions like `munmap` are redefined as macros that call `free()` or do nothing, preserving the original structure.
  ```
  #define munmap(ptr, length) custom_munmap(ptr) // custom_munmap calls free()
  #define close(fd) custom_close(fd)             // custom_close does nothing
  ```
- **Memory Debugging:** The port uses `esp_get_free_heap_size()` during loading to log available PSRAM, a critical step for debugging on a resource-constrained device.

Heart Rate from Face (rPPG)

- [rPPG on web]



[2006.03790] Multi-Task Temporal Shift Attention Networks for On-Device Contactless Vitals Measurement

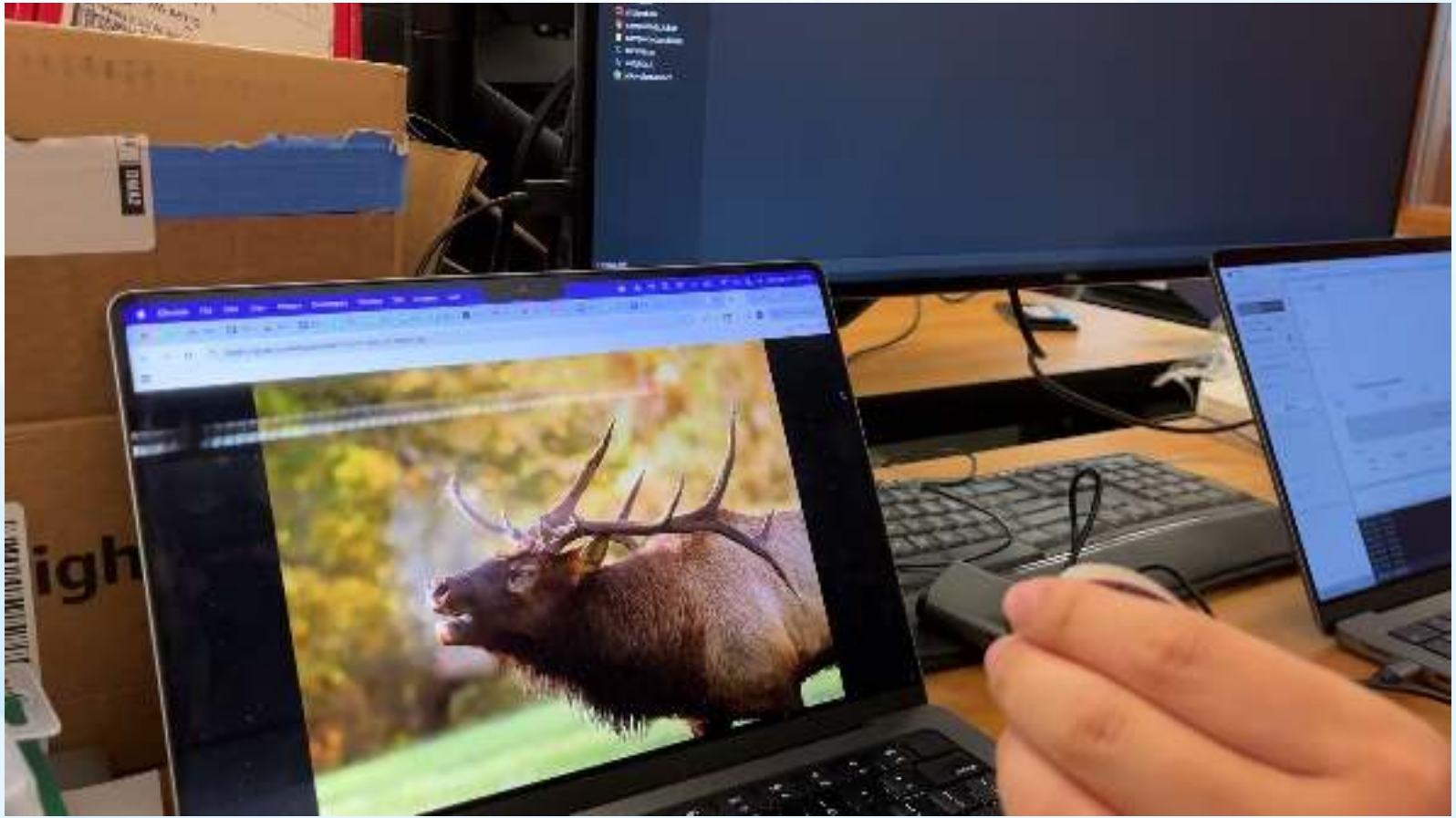
MAX78000 Limitations

- 442 KB model weight limitation
- 90x90 image resolution (without CNN streaming mode)
- 224x224 image resolution (CNN streaming mode)
- 512KB flash memory
  - Firmware
  - Weight and Bias
  - Peripheral (camera, mic, etc) Buffer
- Can only do pooling before convolution, only supports fixed kernel sizes
- SPI (Buggy)
  - First byte read/write always unreliable from our experience
- Manual processor and memory allocation for CNN

MAX78000 Cam02 Module

Demo (Face Detection)

Demo (Elk Detection)



AI85FaceIdNet vs YOLOv8pico (ours)

```
The target architecture is set to "armv7e-m".
Open On-Chip Debugger (Analog Devices 0.12.0-1.0.0-7)  OpenOC
Licensed under GNU GPL v2
Report bugs to <processor.tools.support@analog.com>
0x00082124 in ?? ()
Loading section .text, size 0x52e1c lma 0x10000000
Loading section .ARM.exidx, size 0x8 lma 0x10052e28
Loading section .data, size 0x9d4 lma 0x10052e28
Loading section .shared, size 0x4 lma 0x100537fc
Start address 0x10003154, load size 342812
Transfer rate: 32 KB/sec, 14250 bytes/write.
Section .text, range 0x10000000 -- 0x10052e1c: matched.
Section .ARM.exidx, range 0x10052e28 -- 0x10052e28: matched.
Section .data, range 0x10052e28 -- 0x100537fc: matched.
Section .shared, range 0x100537fc -- 0x10053800: matched.
[Inferior 1 (Remote target) detached]
```

```
The target architecture is set to "armv7e-m".
Open On-Chip Debugger (Analog Devices 0.12.0-1.0.0-7)  OpenOC
Licensed under GNU GPL v2
Report bugs to <processor.tools.support@analog.com>
0x00082124 in ?? ()
Loading section .text, size 0x5a2d0 lma 0x10000000
Loading section .ARM.exidx, size 0x8 lma 0x1005a2d0
Loading section .data, size 0x9d4 lma 0x1005a2d8
Loading section .shared, size 0x4 lma 0x1005acac
Start address 0x1000345c, load size 371888
Transfer rate: 31 KB/sec, 14303 bytes/write.
Section .text, range 0x10000000 -- 0x1005a2d0: matched.
Section .ARM.exidx, range 0x1005a2d0 -- 0x1005a2d8: matched.
Section .data, range 0x1005a2d8 -- 0x1005acac: matched.
Section .shared, range 0x1005acac -- 0x1005acb0: matched.
[Inferior 1 (Remote target) detached]
```