Optimisation Techniques

1. **PyTorch 2.0 - compile**

https://pytorch.org/get-started/pytorch-2.0/

https://pytorch.org/blog/accelerating-generative-ai-2/

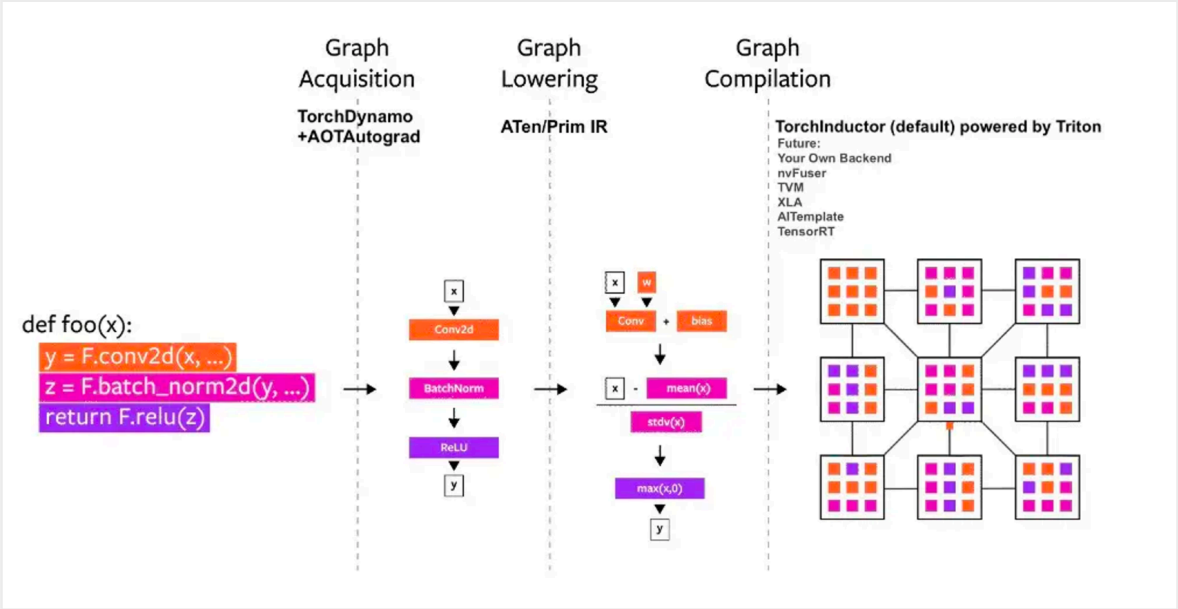
- Torch.compile: A compiler for PyTorch models
- GPU quantization: Accelerate models with reduced precision operations
- Speculative Decoding: Accelerate LLMs using a small “draft” model to predict large “target” model’s output
- Tensor Parallelism: Accelerate models by running them across multiple devices.

torch.compile

```
opt_module = torch.compile(module)
```

Underpinning torch.compile are new technologies – TorchDynamo, AOTAutograd, PrimTorch and TorchInductor.

- **TorchDynamo** captures PyTorch programs safely using Python Frame Evaluation Hooks and is a significant innovation that was a result of 5 years of our R&D into safe graph capture
- **AOTAutograd** overloads PyTorch’s autograd engine as a tracing autodiff for generating ahead-of-time backward traces.
- **PrimTorch** canonicalizes ~2000+ PyTorch operators down to a closed set of ~250 primitive operators that developers can target to build a complete PyTorch backend. This substantially lowers the barrier of writing a PyTorch feature or backend.
- **TorchInductor** is a deep learning compiler that generates fast code for multiple accelerators and backends. For NVIDIA and AMD GPUs, it uses OpenAI Triton as a key building block.

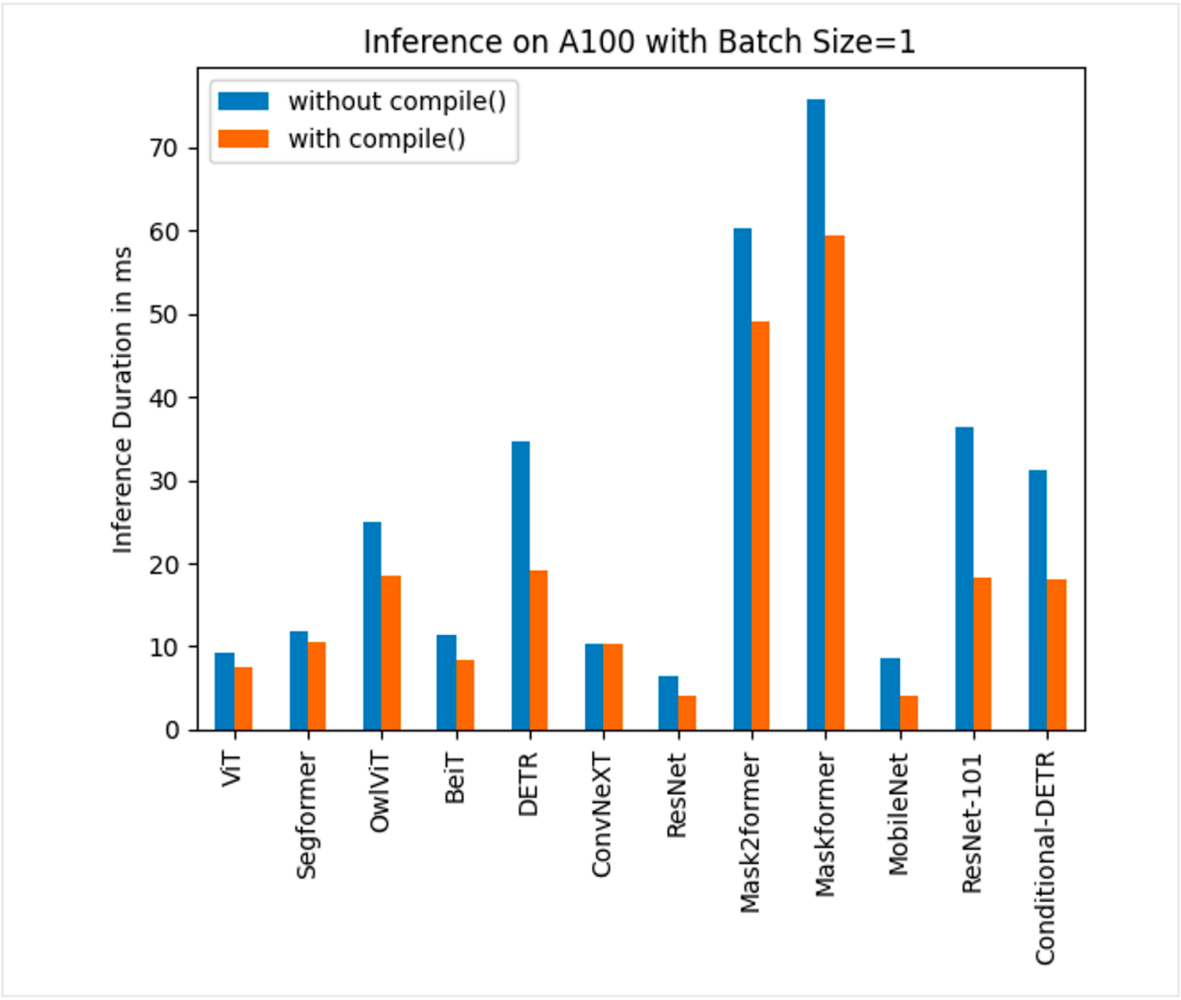


reduce-overhead: optimizes to reduce the framework overhead and uses some extra memory. Helps speed up small models

torch.compile(model, mode="reduce-overhead")

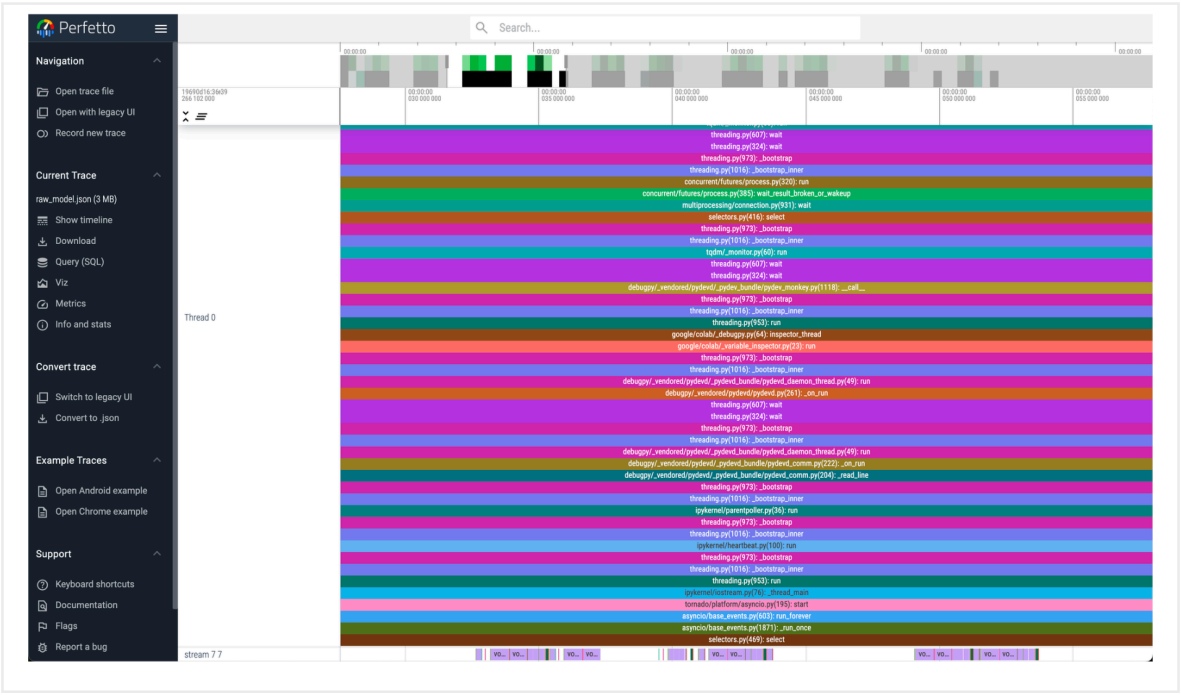
max-autotune: optimizes to produce the fastest model, but takes a very long time to compile

torch.compile(model, mode="max-autotune")



nvitop (pip install nvitop) >>> to monitor Nvidia cpu/gpu live usage

Check tracing using perfetto
————————————————



https://www.youtube.com/watch?v=HvRDGi3tayY

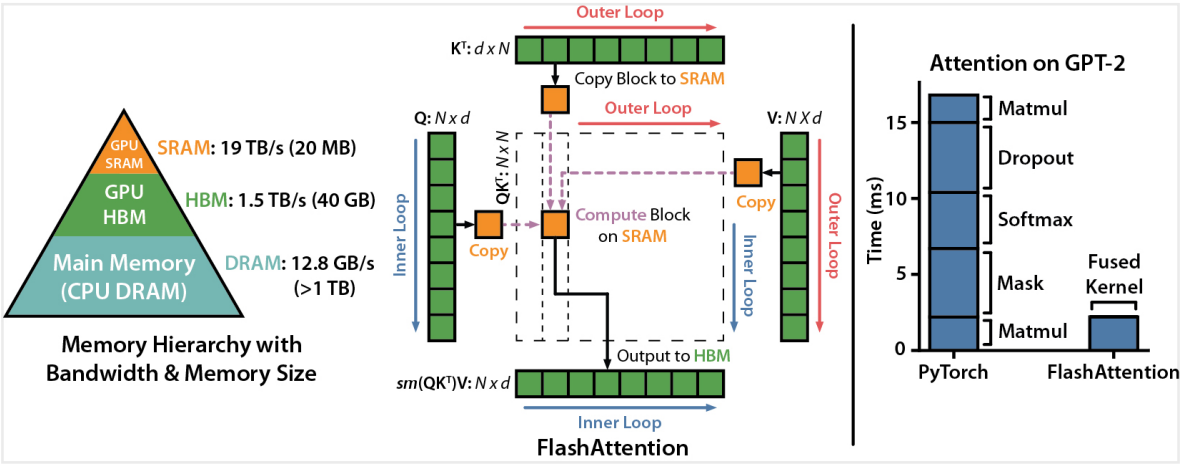
https://ggml.ai/

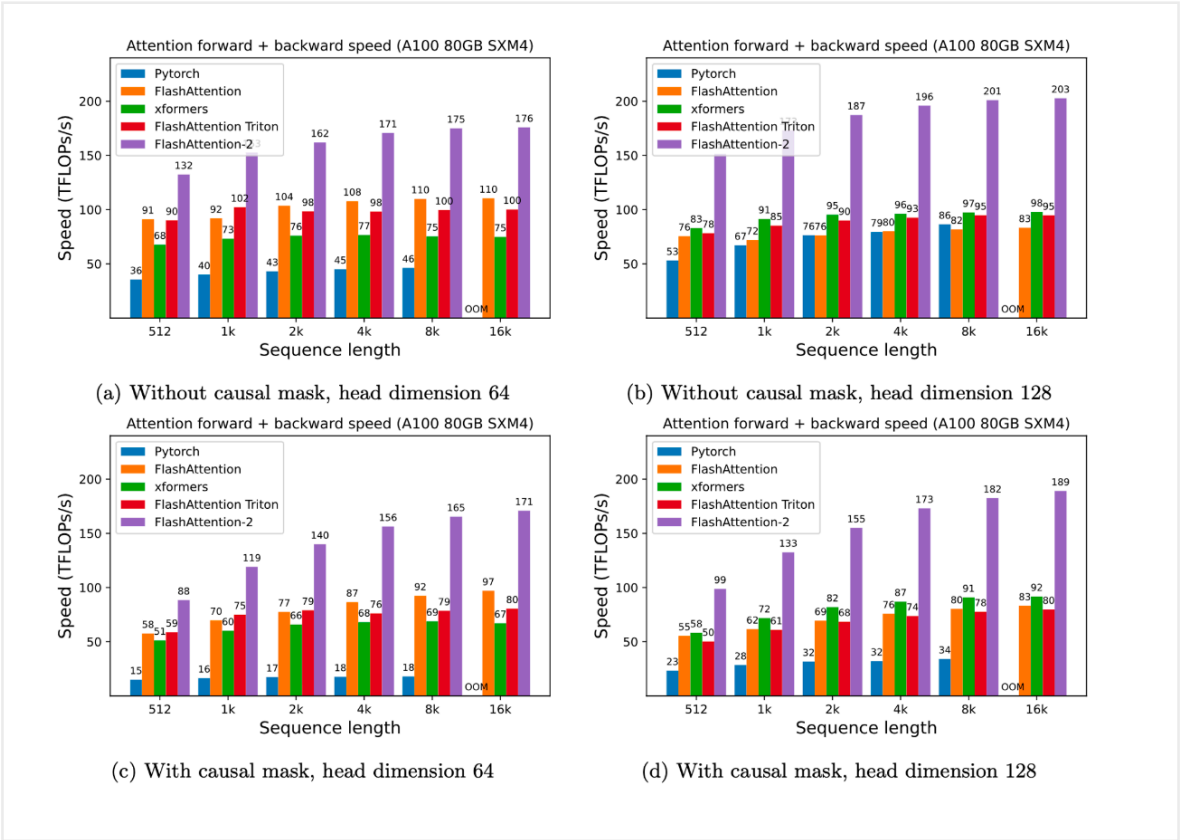
2) Flash Attention

FlashAttention-2 is a faster and more efficient implementation of the standard attention mechanism that can significantly speedup inference by:

1. additionally parallelizing the attention computation over sequence length
2. partitioning the work between GPU threads to reduce communication and shared memory reads/writes between them

FlashAttention-2 supports inference with Llama, Mistral, and Falcon models.





(a) Without causal mask, head dimension 64

(b) Without causal mask, head dimension 128

(c) With causal mask, head dimension 64

(d) With causal mask, head dimension 128

https://pytorch.org/blog/accelerated-pytorch-2/

https://pytorch.org/blog/pytorch2-2/

Scaled-Dot-Product-Attention by PyTorch2

BetterTransformer - by HuggingFace

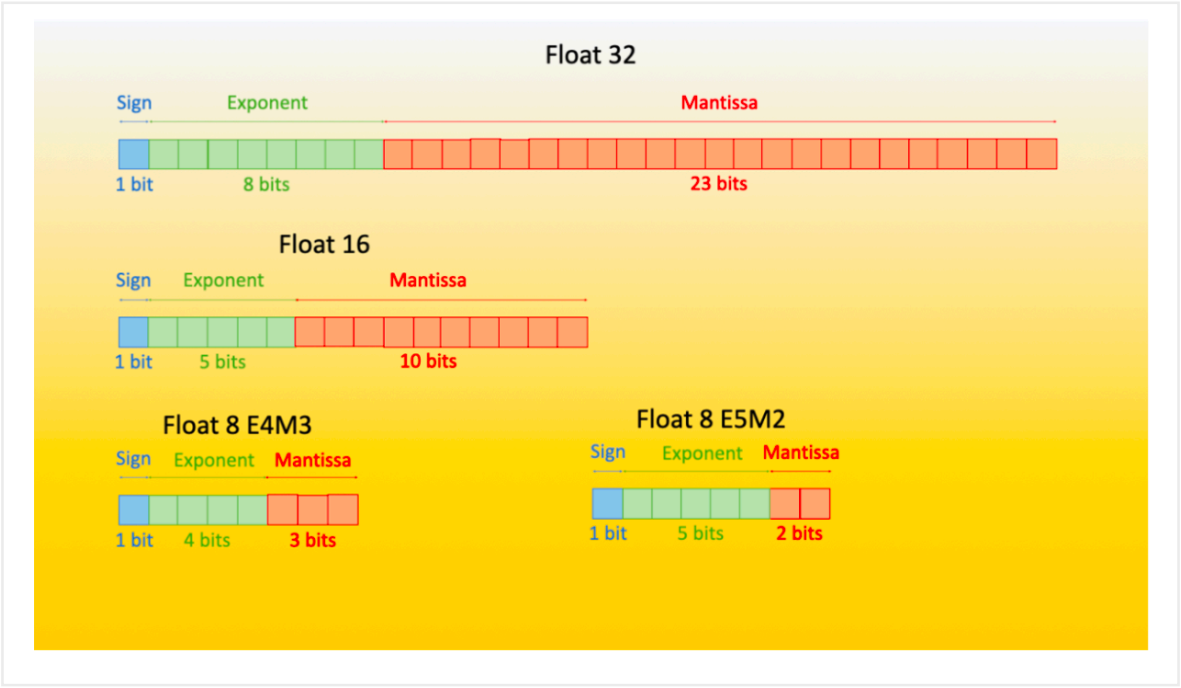
3) Quantization

- BitsandBytes

https://github.com/TimDettmers/bitsandbytes

- Lightning Fabric

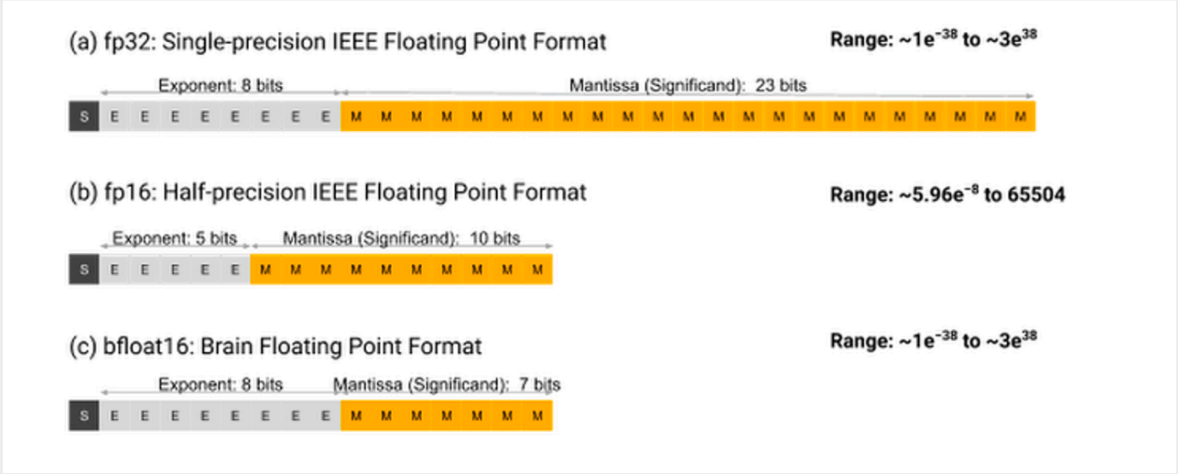
https://lightning.ai/docs/fabric/stable/api/precision.html



- **FP32 (Float32):** This is the standard IEEE 32-bit floating point representation. It has 23 bits for the mantissa, 8 bits for the exponent, and 1 sign bit. FP32 offers a wide range of representable values with good precision, making it the default choice for many computations. However, it requires more memory and computational resources compared to lower-precision formats.
- **FP16 (Float16):** FP16 cuts the number of bits in half compared to FP32, with 10 bits for the mantissa, 5 bits for the exponent, and 1 for the sign. The trade-off is a much smaller range of representable numbers and reduced precision. FP16 can cause numerical issues like overflow and underflow, where very large or small numbers respectively can't be accurately represented and lead to errors such as NaN (Not a Number).
- **BF16 (BFloat16):** To address the limitations of FP16 while not compromising too much on range, BF16 uses 8 bits for the exponent

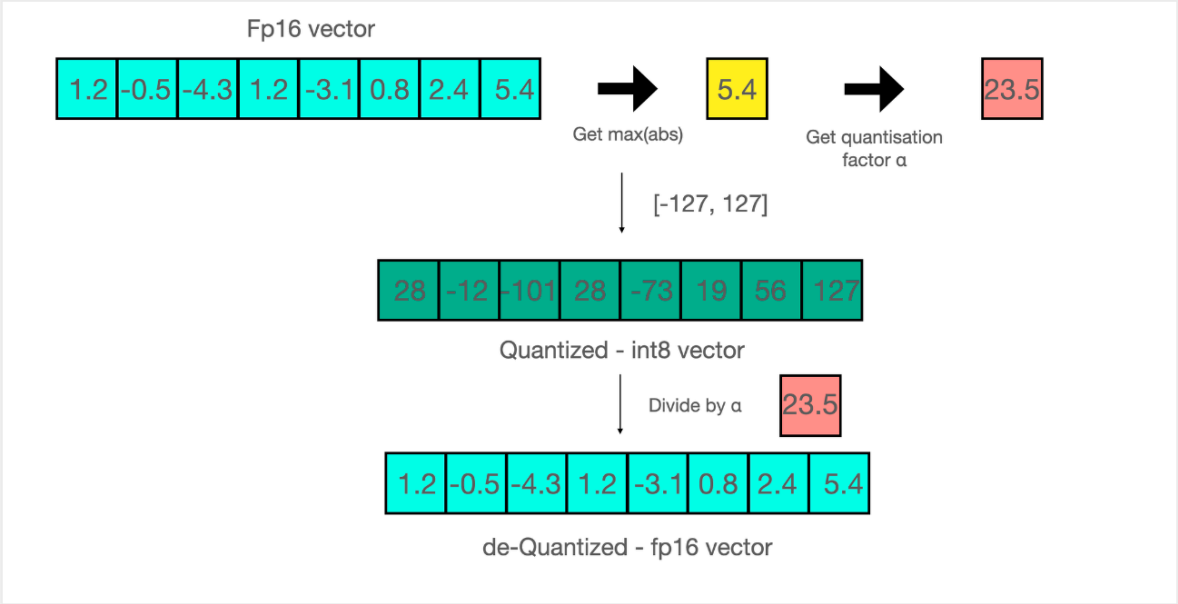
(like FP32) but only 7 bits for the mantissa. This keeps a wide dynamic range similar to FP32 but with slightly lower precision. BF16 strikes a balance that is suitable for many deep learning tasks where the wide range is more important than extreme precision.

- **TF32 (TensorFloat-32):** Exclusive to NVIDIA's Ampere architecture, TF32 offers a new format with 19 bits: 8 for the mantissa and 10 for the exponent. TF32 aims to balance range and precision by using more exponent bits than BF16 and fewer mantissa bits than FP32. It's used internally during specific GPU operations and offers the performance of FP16 with the range close to FP32.



INT8

Absmax quantization is one type of quantization that scales numerical values to fit within the range of a target data type, such as int8.



INT4

NF4(NormalFloat 4bit)

Double Quantization

Combining Flash Attention with INT8/4 quantisation

Offloading between CPU and GPU

https://huggingface.co/docs/accelerate/usage_guides/quantization

AWQ - Activation Aware Weight Quantisation

AutoAWQ package - https://github.com/casper-hansen/AutoAWQ

GPTQ

AutoGPTQ package. - https://github.com/AutoGPTQ/AutoGPTQ

4. **Graph Optimisation - ONNX conversion**
5. **kv-caching (key-value)**
6. **Tensor Parallelism**
7. **Dynamic Batching**
8. **Distillation**
9. **Pruning**