Jeff Dean - NeurIPS 2024

Advances in ML for systems

How to apply learning to make systems better?

Bitter Lesson: Search & Learning

OS, compilers, etc. doesnt use ML. It will soon.

① Learned Compiler Choices

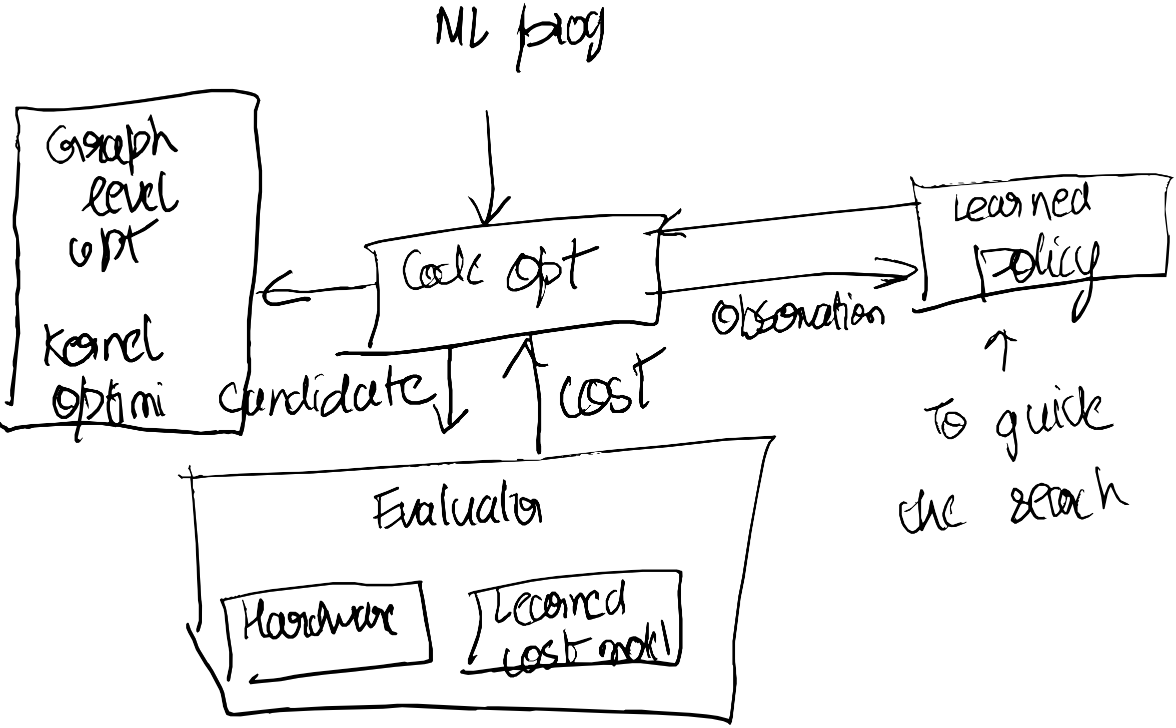
→ Good opportunity in compiler/ML land for ML. Good reward example.

→ XLA TPU Autotuner

→ Evaluator with cost model

→ Code optimizer

→ Learned policy



Eg: Operator Fusion
Combine into a single operation

Layout Assignment
→ How to layout dims of a tensor on your hardware?
→ 5 - 25% speedup possible

Learned Object Lifetime Predictions

→ tcmalloc, malloc

Allocate object
Delete object

Lifetime of object severly affects what we do with it

short lived ⇒ Put it in thread local cache

Long lived ⇒ Maybe put in central memory for easier mgmt & comms.

Important to cluster objects with similar lifetime on the same page
If diff pages, pages live till object is deleted, leading to memory wastage.

In what context was an objected allocated memory. Using this context i.e callstacks, we can approx predict the duration of the object & page life.

Treating callstacks as text, train an LSTM, can get good predictions i.e good cache hit rate

Martin Mass ⇒ Paper author.

LLAMA algorithm → Predict if object will live as long as current page, lesser or higher.

19 - 78 % ↓ in memory fragmentation!!

Learned Backtracking

-> Repeatable workload
-> Predictable allocation
-> ML Accelerator Compilation
-> Determines how to place buffers within ML accelerator

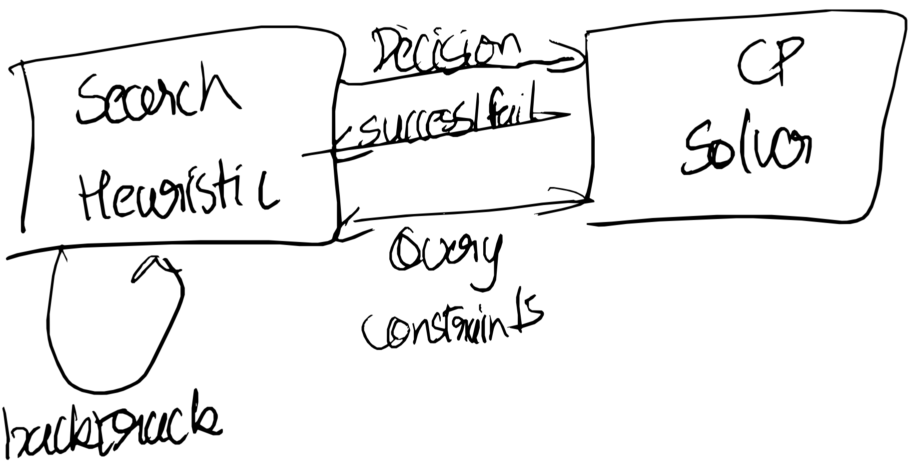
TerraAlloc

Memory allocation -> Given a sequence of fixed-size buffers, with known start & end time, place them in mem such that total used mem never exceeds capacity.

NP - Hard Problem

TeleMalloc → combine heuristics & solvers

Telemon



Backtracking issue → Don't know how far to backtrack.

Use ML. i.e Imitation Learning.

Learn from annotated scenarios

Enforce correctness by guiding solver

SmartChoices

make it easy to integrate learned choices into app code

this is multi-arm bandit setup

Paper: Smartchoices

can learn things like video to evict from cache

→ optimise thread counts

Faster inference

↓ cost & ↓ latency

Chinchilla laws ignores expected inference load when deciding model attributes before training

→ Produce high quality smaller models via overtraining

→ Use distillation to make small models (like R1)

Sparse models

→ Activate small portion of LLM at inference time

→ Gemini 1.5 Pro is a MOE model (

DiPaCo paper looking more important

Speculative Decoding

→ Enable faster decoding

① Decoding from transformers is (memory bound)

② Some tokens are easier to predict than others.

Key idea: small model generates tokens, & large model checks them in $\|^{le}$ using spare compute.

Direct app of this is inefficient, hence accept/reject stochastically

Designing a new chip

Long process

Extremely costly

Need to reduce time & cost

① Use moar machines i.e run computations in parallel

② Use ML compute

Moar compute + Parallelism

Google ⇒ 10M \$ in 21 days
↓
15 exaflops of compute.

Use end-to-end learning as much as possible

AlphaChip already used in TPU

Open source

Maybe use this for course.