Assessment of the Effectiveness of ML-based Performance Models for Compiler Optimization

Fabrice Rastello

Assessment of the Effectiveness of ML-based Performance Models for Compiler Optimization

Fabrice Rastello
INRIA and UGA
Fabrice.rastello@inria.fr

March 3, 2024

Once upon a time...

There was a poor compiler guy. Working hard. No time for coffee
No much outcome.

…Till he saw an apparition

Introduction

ML will make him rich!
He will use his compiler expertize to optimize ML
And come talk at C4ML…

Introduction

Still working hard. No time to drink coffee...
And did not work that well (can't come to C4ML)
…So he decided ML would do the job for him

Introduction

He drinks coffee…
Came to C4ML to advocate changing the name to ML4CC4ML

Introduction. Layout

Part 1: *Advocate the change of name from C4ML to ML4CC4ML*

- Classical compiler approach (roofline model based « one-shot »)
- Througput model
- Operational intensity model
- Evaluation

Part 2: *Precise what mean by ML*

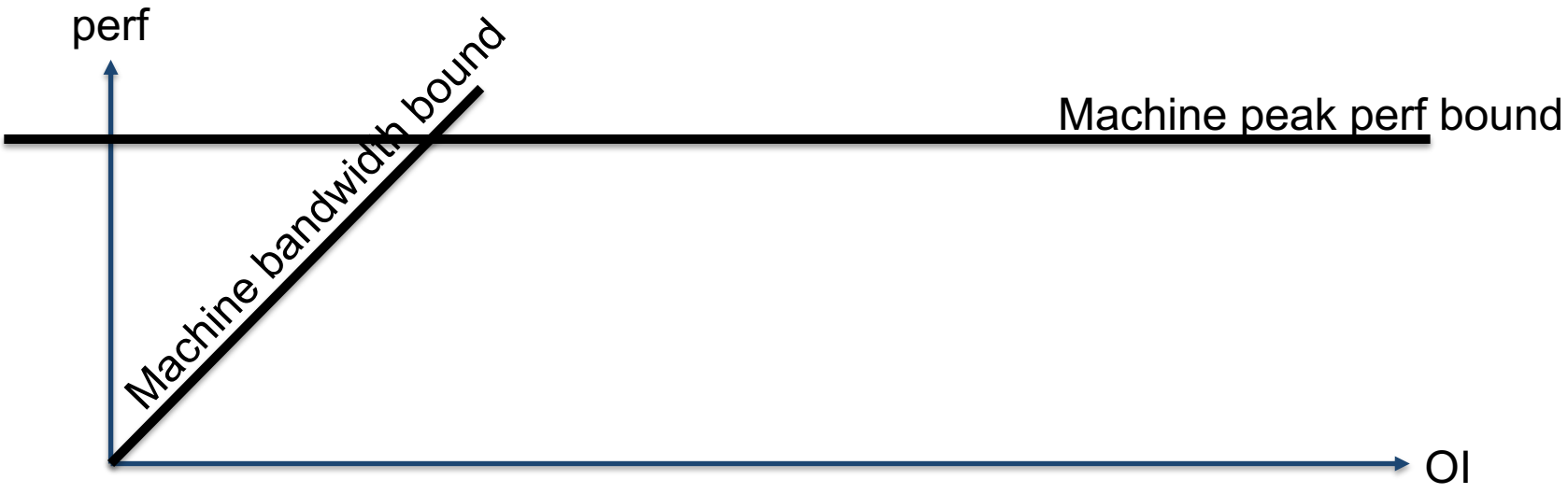
- General infrastructure. Main pillars
- Ansor as a starting point. Raised questions
- Illustration through experiments

Part 1. Approach

- Lesson from Ansor: Can exploit specific pattern of deep learning computation kernels + specific tile sizes
- Expertize in low level + data movement: Should be piece of cake

⇒ Use roofline model approach.

- peak perf == throughput model (Intel IACA)
- OI == analytically derived (polyhedral model)

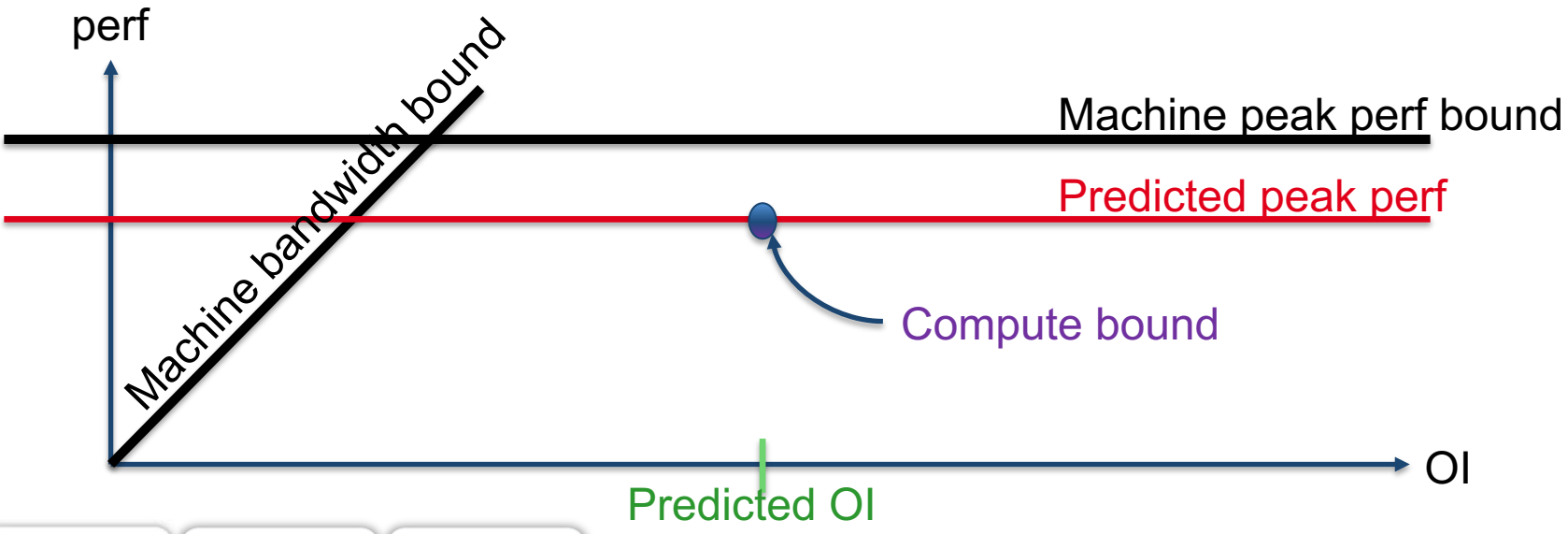


Part 1. Approach

- Lesson from Ansor: Can exploit specific pattern of deep learning computation kernels + specific tile sizes
- Expertize in low level + data movement: Should be piece of cake

⇒ Use roofline model approach.

- peak perf == throughput model (Intel IACA)
- OI == analytically derived (polyhedral model)

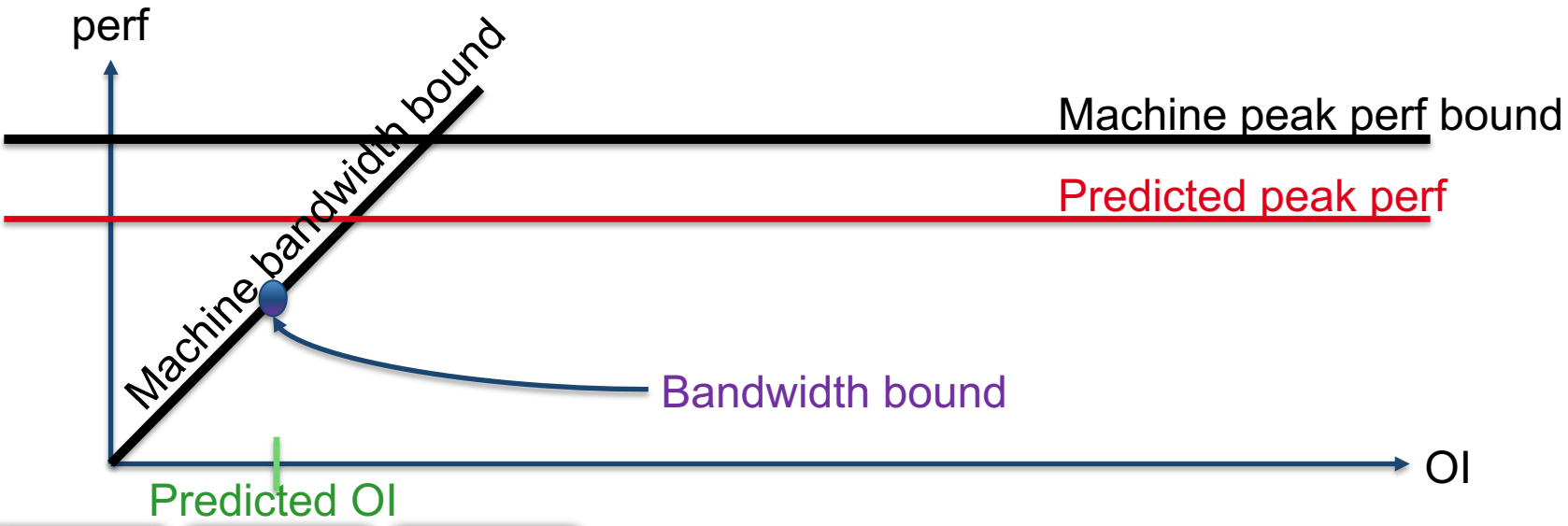


Part 1. Approach

- Lesson from Ansor: Can exploit specific pattern of deep learning computation kernels + specific tile sizes
- Expertize in low level + data movement: Should be piece of cake

⇒ Use roofline model approach.

- peak perf == throughput model (Intel IACA)
- OI == analytically derived (polyhedral model)



Part 1. Throughput model

- Basic block. Infinite loop arround. L1 resident accesses.

⇒ Predict number of cycles per iterations

- Several approaches:
  - Behavioral model (Intel IACA, uICA, LLVM-MCA)
  - Context free measure model (BHive)
  - Learned model (Ithemal, Granite)
  - Behavioral+learned model (GUS, PALMED++)

Part 1. OI model

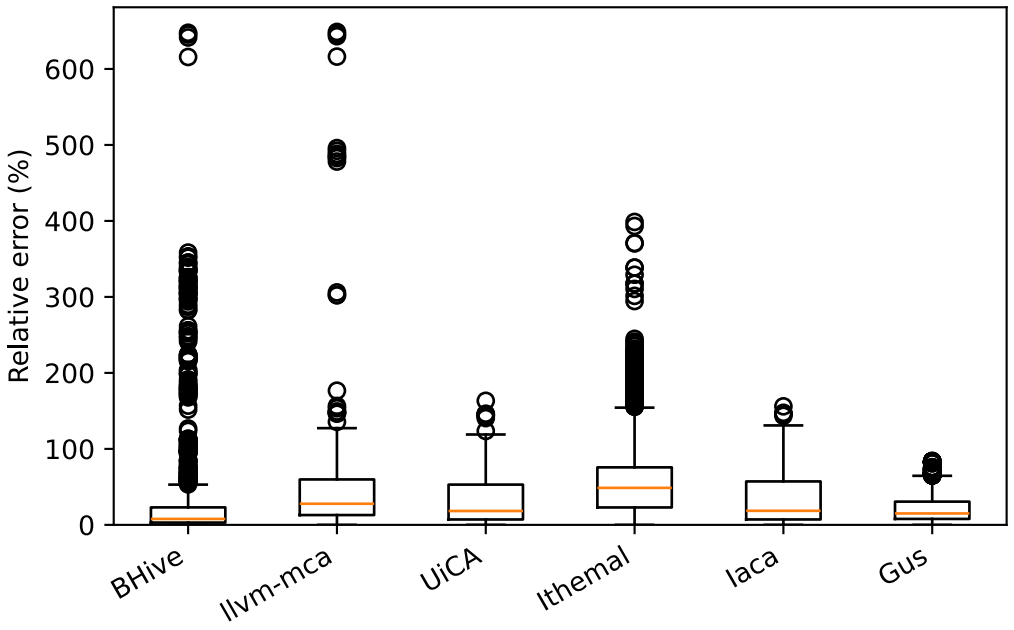
- Affine loop body. Simplified cache model (fully associative).

⇒ Approximate the ratio: #ComputedFlops / #MovedBytes

1. Compute footprint for every tile/loop level (Barvinok)
2. Find loop level that saturates given cache level.
3. Assume full reuse of overlapping data between consecutive tiles of surrounding loop.
4. Derive analytical formula (multivariate polynomial)

Part 1. Evaluation. Througput

- Benchmarking:
  - Innermost loop: Polybench. Loop transformations
  - L1 resident loops only
  - GCC with many compiler opts
  - Skylake



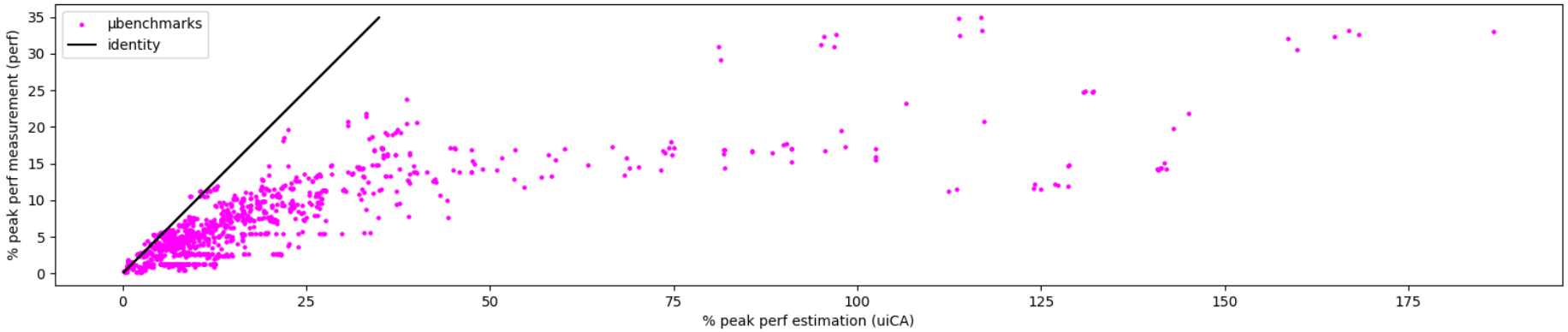
Part 1. Evaluation. Througput

- Precision of % peak perf estimation for uICA:
  - Estimated IPC
  - Dynamic analysis to normalize to Flop per iteration
  - Manual analysis to normalize to peak perf

  ⇒ Mostly over estimated

  ⇒ Cannot be used for « one-shot »

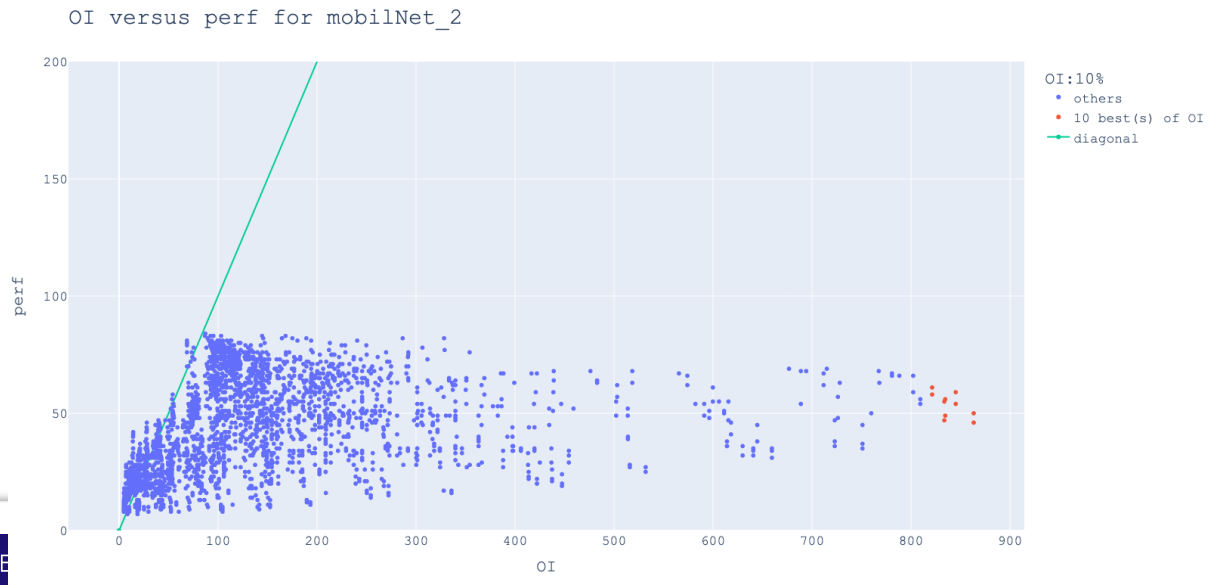
  ⇒ Empirically select micro-kernels such that *perf > 0.85 x machine peak perf*



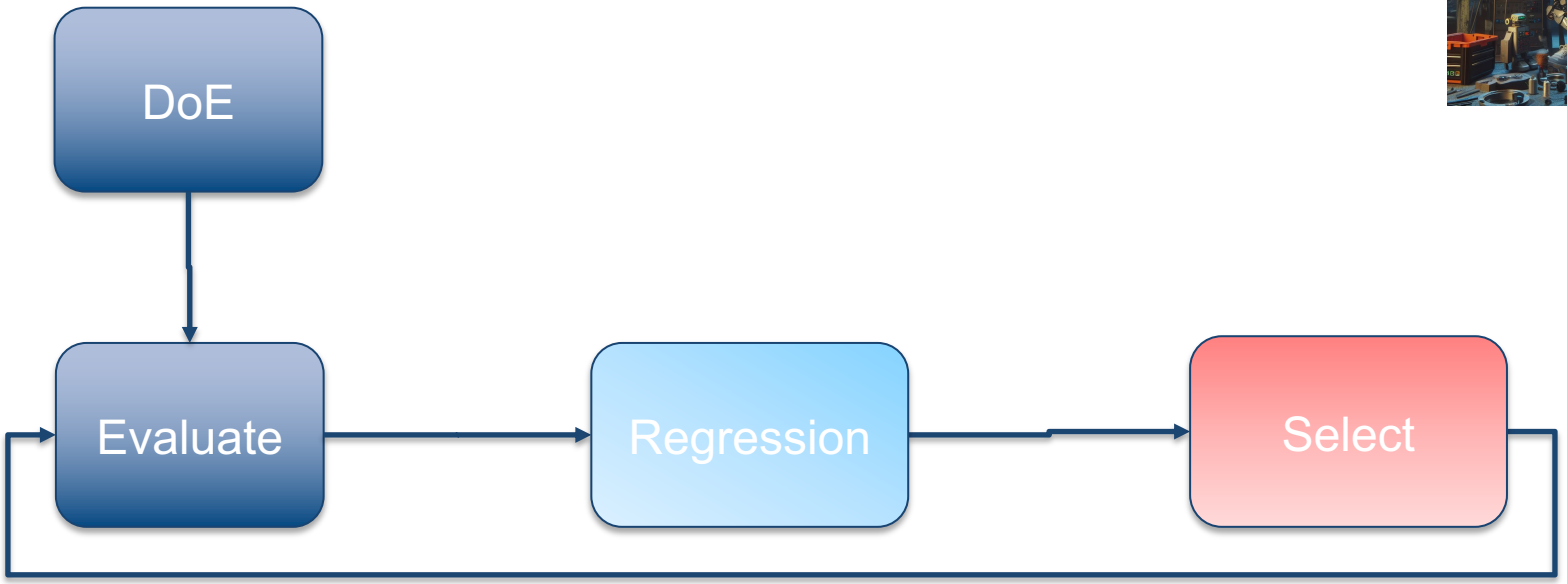
Part 1. Evaluation. OI

- Precision of OI as performance predictor
  - MobilNet_2 (did for many others)
  - Randomized tiling variants (with selected micro-kernels)
  - For each variant:
    - Compute operational intensity. Normalize to % of peak perf using actual machine bandwidth (green diagonal)
    - Measure actual perf. Normalize to ¨of peak perf
  - Scatter plot: normalized OI (x-axis) versus; normalized perf (y-axis)

  ⇒ Can't be used for « one-shot »

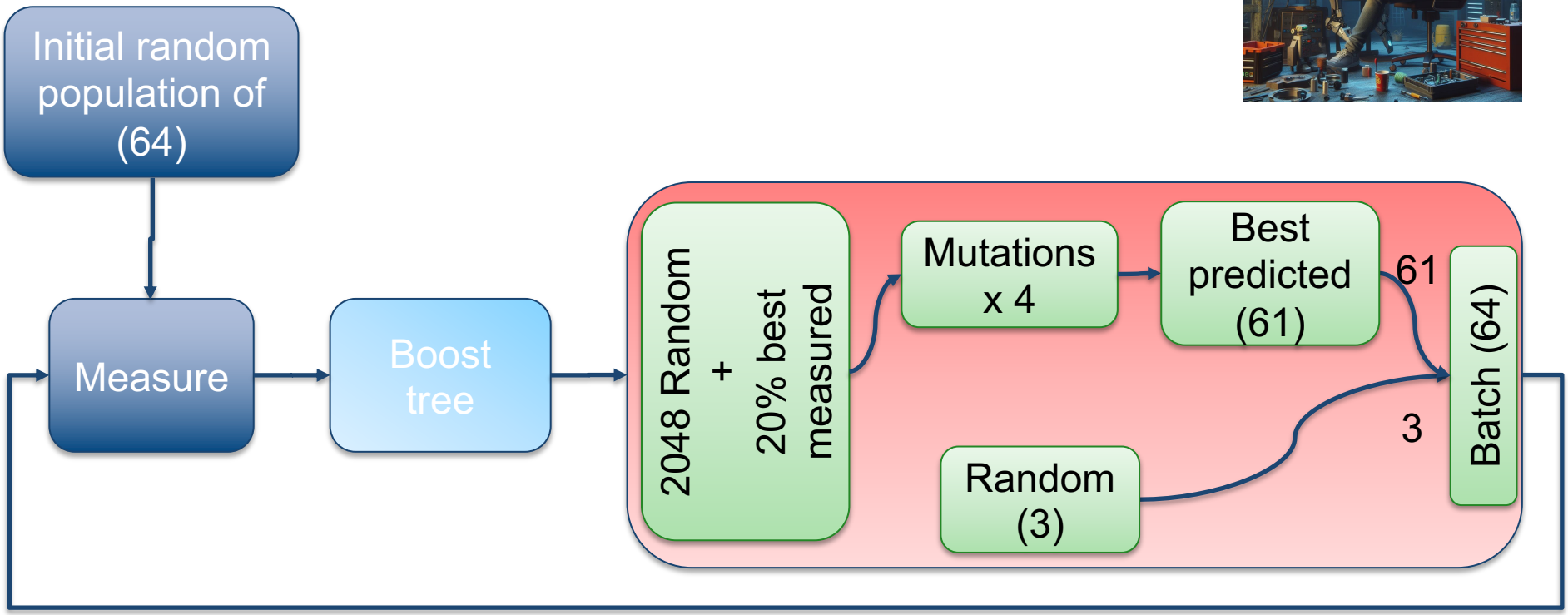


Part 2. General infrastructure.



- **Main pillars:**
  - Expertise (search space, neighbourhood, prior distribution…)
  - Learning (prediction, uncertainty)
  - Exploitation (to find the best)
  - Exploration (to improve our knowledge)

Part 2. Ansor (simplified).



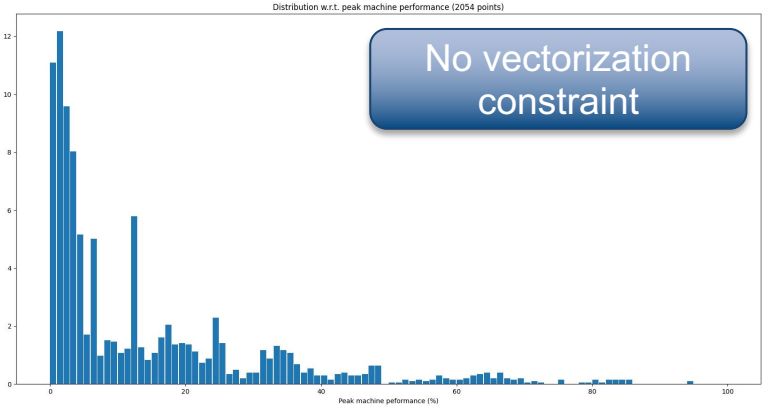
- **Remarks/Questions:**
  - Good because of high expertise?
  - Exploration only due to randomization (model that overfits)?
  - Role of model?
  - Would a model of uncertainty help?

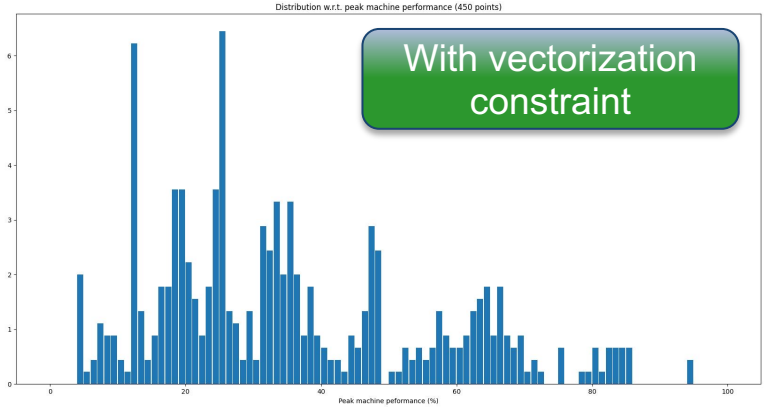
Part 2. Discussion: Expertise

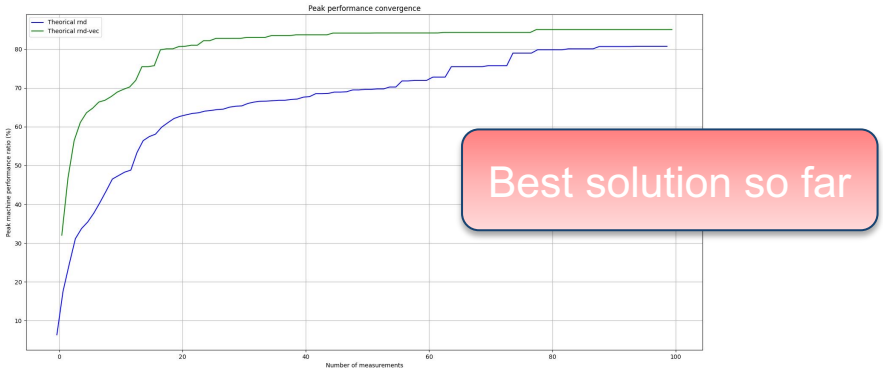
- Performance distribution on search space w/o vectorization imposed.

⇒ 50 measures: 20% faster

⇒ Performance distribution on search space matters!



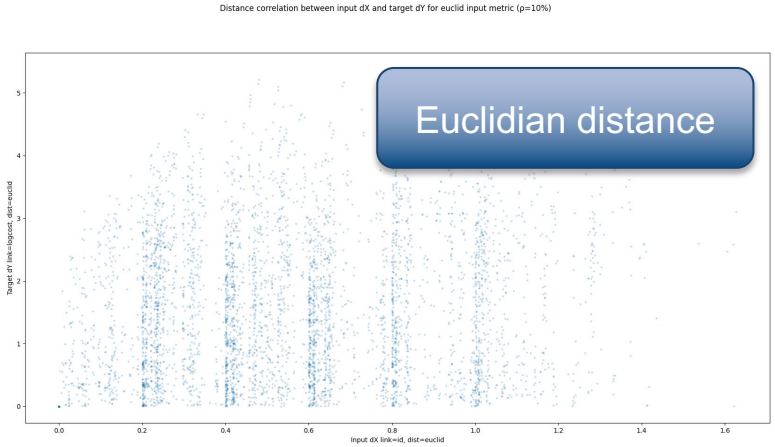


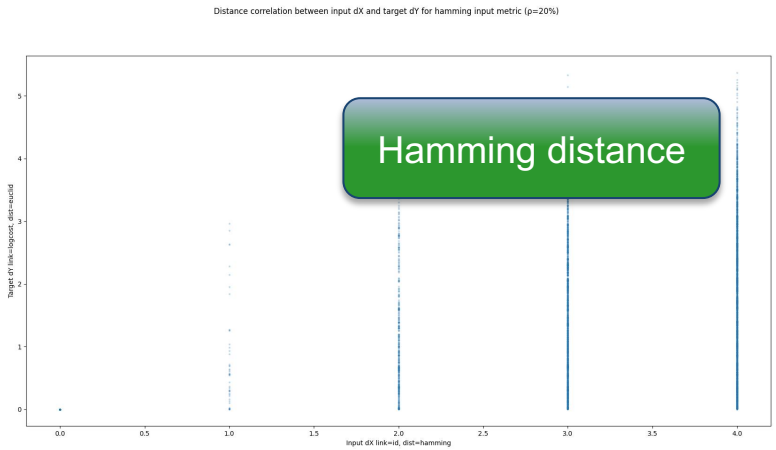


Part 2. Discussion: Expertise

- Neighborhood / prior on distance (Euclidian vs Hamming)
- Distance (x-axis) and performance (y-axis)

$\Rightarrow$ No correlation with naive Euclidian distance

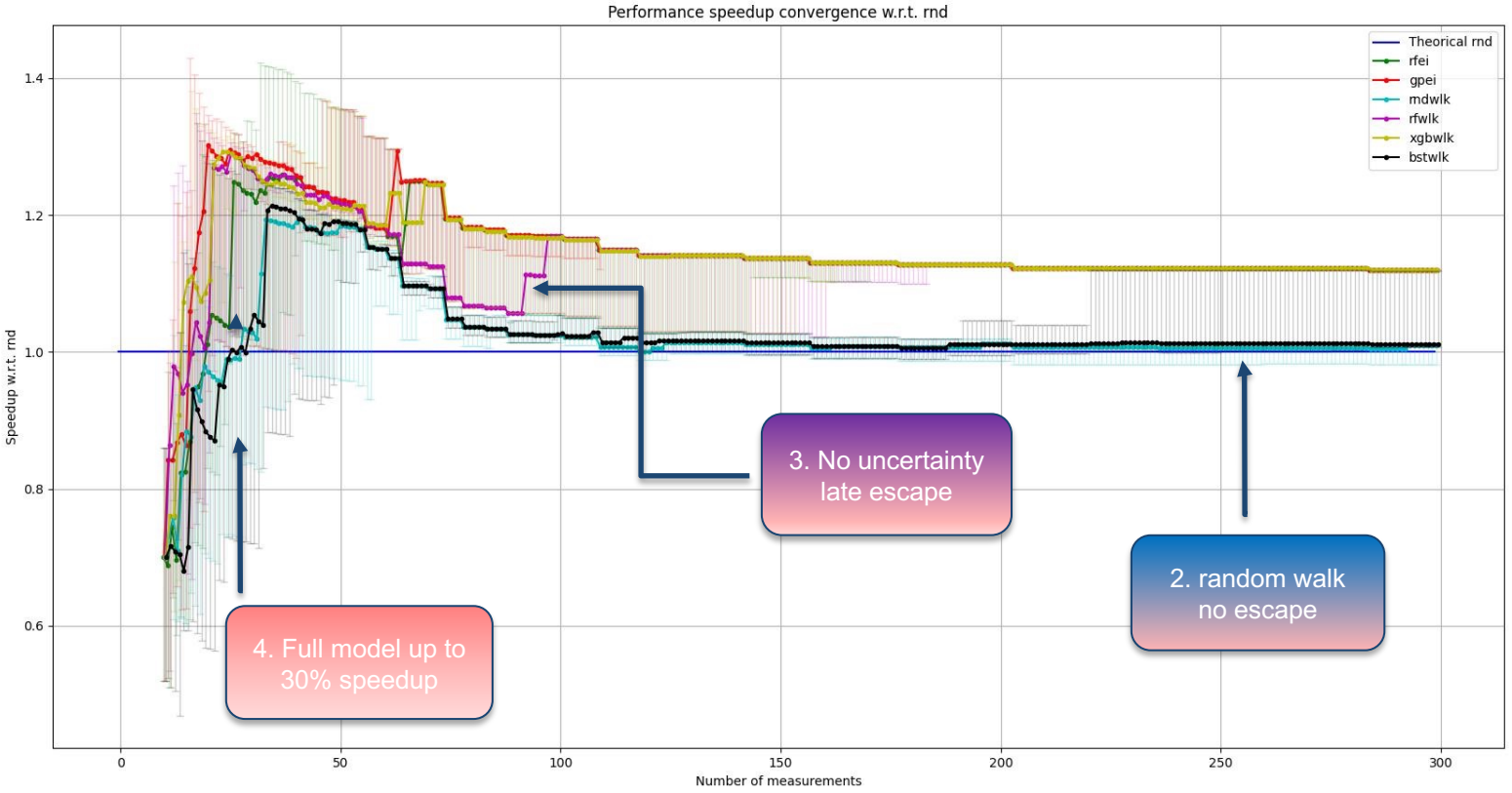




Part 2. Discussion. Explore/exploite

Exploration/exploitation: prediction model & uncertainty model

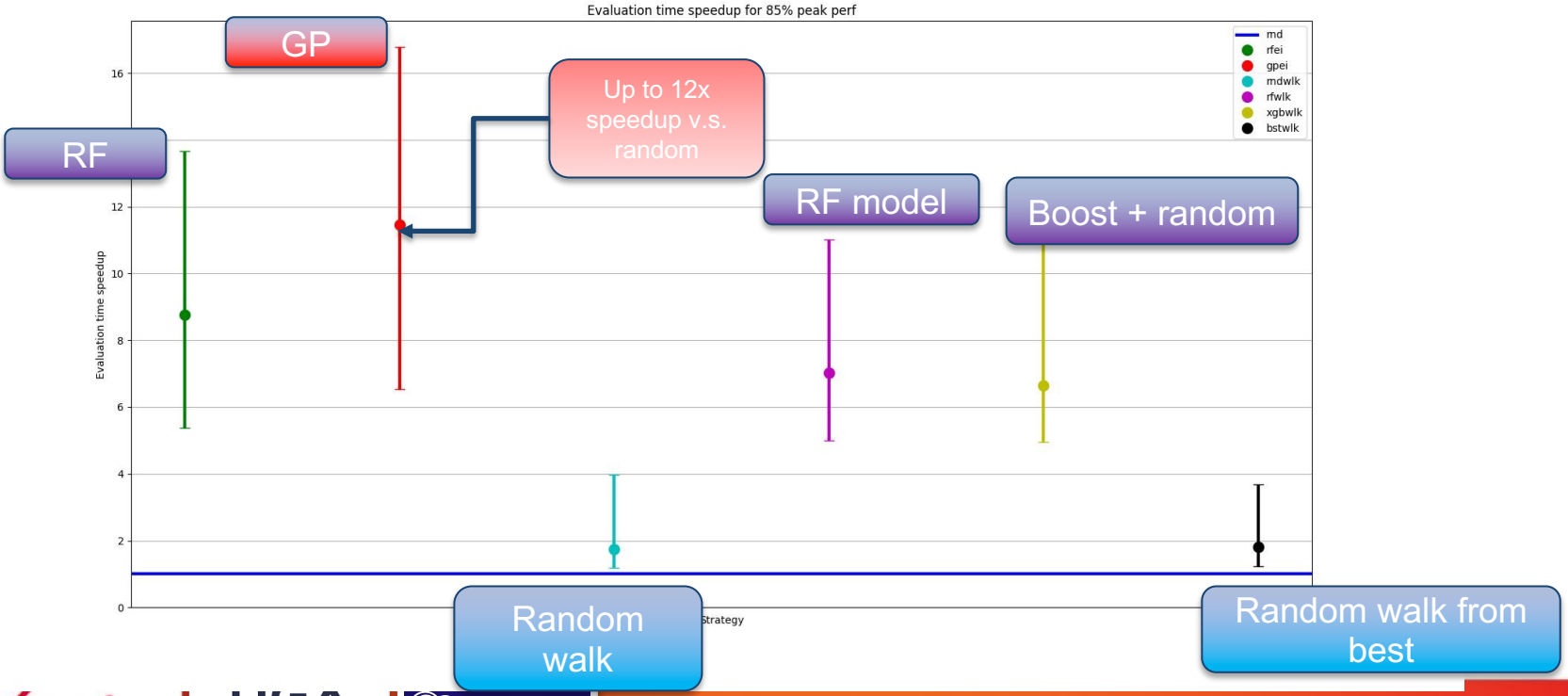
1. Pure random: no model (**naive** exploration)
2. Random walk **from best**: (exploration from initial sampling; local exploration from random walk; exploitation by starting from the best)
3. Walk driven by **prediction** model (RF, XGB): idem but exploits the model
4. Walk driven by prediction+**uncertainty** model (RF, GP): better exploration



Part 2. Discussion. Explore/exploit

Exploration/exploitation: prediction model & uncertainty model

1. Pure random: no model (**naive** exploration)
2. Random walk **from best**: (exploration from initial sampling; local exploration from random walk; exploitation by starting from the best)
3. Walk driven by **prediction** model (RF, XGB): idem but exploits the model
4. Walk driven by prediction+**uncertainty** model (RF, GP): better exploration



Conclusion.

Concluding remarks:

- Sequential optimization seems great! We need compiler infrastructure to allow for « transactional partial lowering & execution ». Possible in MLIR?
- Expertize is crucial. Hardness is not due to size of space but to distribution of performance and « smoothness »
- Hyperparameters for trading exploration/exploitation is challenging (more than actual precision of model)

Did not talk (but have some idea):

- Do not evaluate precision of a model (MSE & Kendall tau) but evaluate its ability to take good decisions (precision, recall as a function of targetted performance)
- Expose improvement of optimizer as a Pareto normalized to pure random
- Multi-armed bandit is a different objective
- Generalizing to optimizing the full graph on multi-accelerators with distributed memory is challenging

References.

Thanks to:

- Christophe Guillon, Hugo Pompougnac, Guillaume Iooss, Mariana Vargas Vieyra, P. Sadayappan, Albert Cohen, DALL.E

References:

- *Assessment of the Effectiveness of ML-based Performance Models for Compiler Optimization*. arXiv 2024 (to appear)
- *Performance bottlenecks detection through microarchitectural sensitivity*. arXiv 2024
- *CesASMe and Staticdeps: static detection of memory-carried dependencies for code analyzers*. arXiv 2024
- *Autotuning Convolutions Is Easier Than You Think*. ACM TACO 2023
- *PALMED: Throughput Characterization for Superscalar Architectures*. CGO 2022
- *IOOpt: automatic derivation of I/O complexity bounds for affine programs*. PLDI 2021
- *Analytical cache modeling and tilesize optimization for tensor contractions*. SC 2019