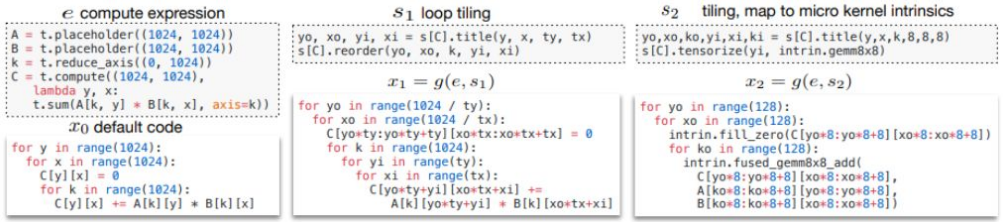
TVM study

Automating Optimization

소준혁

문제

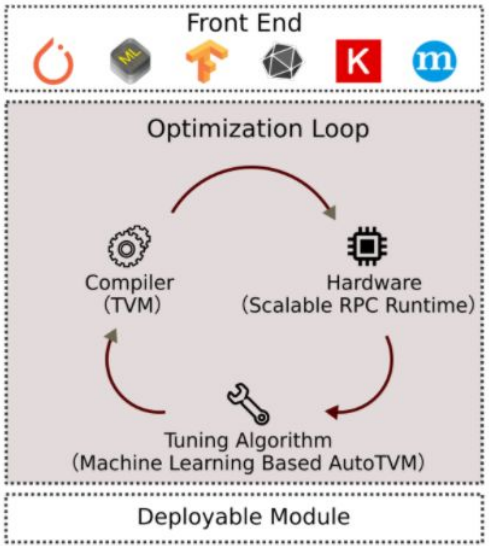
- Kernel성능에 영향을 끼치는 수많은 요소가 있음
  - Loop Order
  - Memory Access
  - Loop Unrolling
  - Tiling size
  - Shared Memory Caching
  - etc…



- 이러한 조합의 경우의 수는 거의 **억단위**로, 이들을 **Grid Search하며 Auto Tuning 하는것은 거의 불가능.**
- 그렇다고 해서, 성능을 저러한 변수들을 써서 **수식적으로 모델링 하는것도 너무 어려움.**

- -> 따라서, 이러한 문제를 ML을 사용한 optimization으로 해결하고자 함.

System Overview

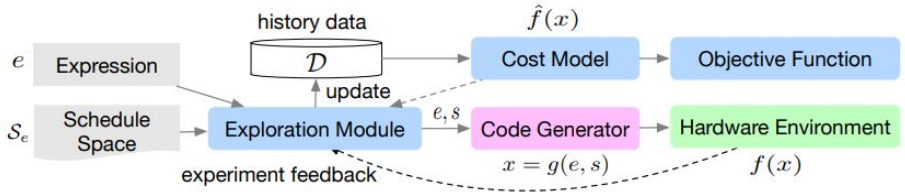


- Input : Computational Graph of Neural Net
- Output : Optimized backend kernels

- 시스템의 간단한 작동개요는 아래와 같음.
  - Tuning Algorithm내부의 **ML Model 이** Schedule Space에서 **적절한 Shedule 를 선택**함
    - Tiling size, unrolling factor ..
  - TVM이 이를 받아 **실제 코드로 컴파일.**
  - 이를 **실제 Hardware에 구동**시켜 Cost(latency)를 측정.
  - 얻은 Cost로 ML Model을 **재학습.**

  - 이러한 **루프를 계속 반복**하여 Optimized된 Kernel을 찾게됨

Tuning Algorithm



e = Compute Expression , s= schedule config, c=cost ,D=Database

Algorithm 1: Learning to Optimize Tensor Programs

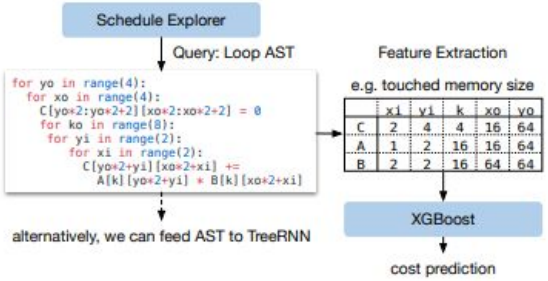
Input : Transformation space $\mathcal{S}_e$
Output : Selected schedule configuration s^*
$\mathcal{D} \leftarrow \emptyset$
while *n_trials* $<$ *max_n_trials* **do**
  `// Pick the next promising batch`
  $Q \leftarrow$ run parallel simulated annealing to collect candidates in $\mathcal{S}_e$ using energy function $\hat{f}$
  $S \leftarrow$ run greedy submodular optimization to pick $(1-\epsilon)b$-subset from Q by maximizing Equation 3
  $S \leftarrow S \cup \{$ Randomly sample ϵb candidates. $\}$
  `// Run measurement on hardware environment`
  for s ***in*** S **do**
    $c \leftarrow f(g(e,s)); \mathcal{D} \leftarrow \mathcal{D} \cup \{(e,s,c)\}$
  end
  `// Update cost model`
  update $\hat{f}$ using $\mathcal{D}$
  n_trials $\leftarrow$ n_trials $+ b$
end
$s^* \leftarrow$ history best schedule configuration

$$L(S) = -\sum_{s \in S} \hat{f}(g(e,s)) + \alpha \sum_{j=1}^{m} |\cup_{s \in S} \{s_j\}| \qquad (3)$$

Minimize cost Maximize Diversity

1. 우선 Simulated Anneling 알고리즘을 사용하여 Global Search Space에서 유망한 후보 Q를 추림.
   a. ML Model을 사용하더라도 전체 Search Space가 너무 커, 다 실행해보며 Top k개의 후보를 추릴수가 없음.
2. Q Eq.3 을 최대로하는 부분집합들 S를 생성
   a. Cost를 Minimize하면서, Diversity가 Maxmize되는 부분집합들.
3. S에 랜덤성 부여
4. 실제 HW에서 S의 cost들 측정.
5. 측정한 Cost로 ML Model을 retrain.

ML Model



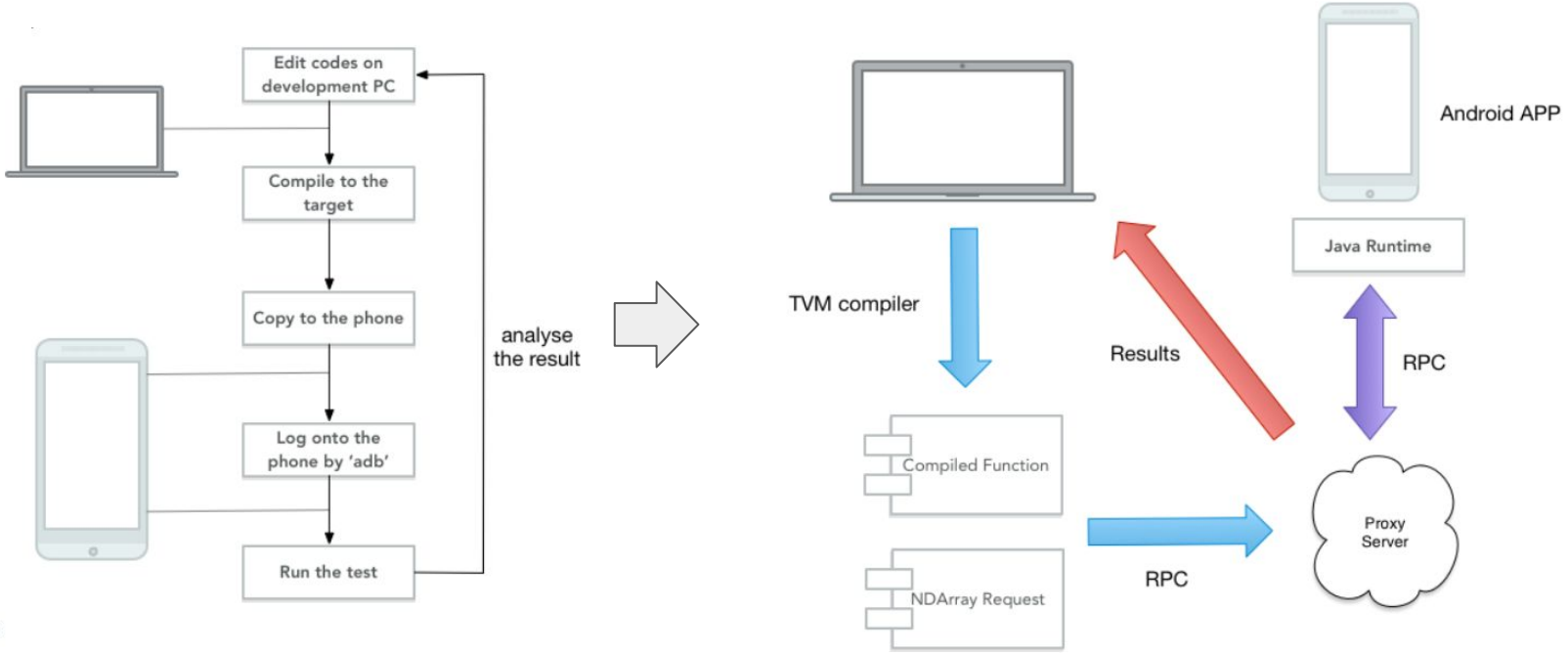
- ML Model의 사용이유는 결국 시간이 많이 소요되는 과정 (Compile -> Evaluation(on real HW)) 을 피하고자 함에 있음.
  - 따라서, ML Model의 refitting -> inference 시간이 이런 시간에 비해 크게 작아야 의미 가 있음.

- 저자들은 Decision Tree 기반의 ML Model인 XGBoost 사용.
  - inference time 약 0.67ms (~ x1000 faster)
  - Tree GRU기반의 딥러닝 모델또한 사용해보았지만 , Accuracy는 비슷한데에비해 Latency가 두배정도 커서 XGBoost 사용.

- 학습시 Loss function으로 Lank Loss function 사용
  - MSE, RMSE, MAE.. 등과같은 간단한 Regression loss를 사용할수도 있겠지만, ML Model의 역할은 cost가 가장 작은 Top k개의 후보를 뽑는 목적이므로, Relative Speedup만 알면됨.(각 데이터들의 순위만 맞추면됨)

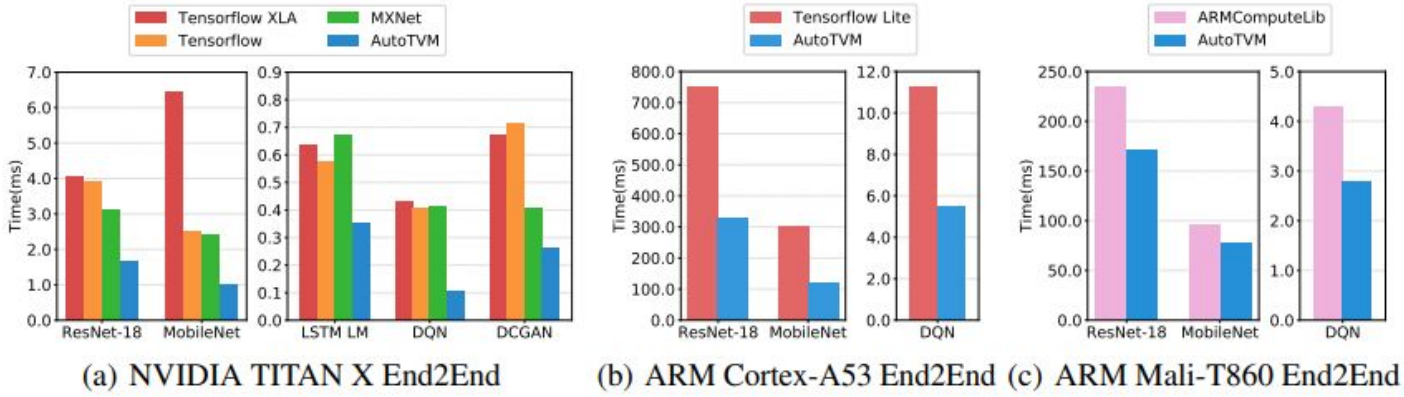
$$\sum_{i,j} \log(1 + e^{-\operatorname{sign}(c_i - c_j)(\hat{f}(x_i) - \hat{f}(x_j))}).$$

Distributed RPC



- Optimization과정에서 결국 HW에서 실제 latency 측정이 필요.
- 하지만 이러한 과정은 매우 귀찮고 번거로움.
- RPC는 이러한 과정을 쉽게 도와주는 Interface.
  - Host Computer에서 측정할 함수를 RPC Server에 보내기만 하면, 자동으로 Cross-compile후 Mobile device(혹은 Server)에서 시간을 측정해 결과를 받을 수 있음.

Benchmark results



(a) NVIDIA TITAN X End2End (b) ARM Cortex-A53 End2End (c) ARM Mali-T860 End2End

- NVidia GPU(Server), ARM Cpu, GPU(Mobile)모두 TVM이 가장 좋은 성능을 보임(2019)