Enabling Tensor Language Model to Assist in Generating High-Performance Tensor Programs for Deep Learning

Authors： Yi Zhai[1],Sijia Yang[2], Keyu Pan[3], Renwei Zhang[2],Shuo Liu[1]
Chao Liu and Zichun Ye[2],Jianmin Ji[1],Jie Zhao[4],
Yu Zhang and Yanyong Zhang[1]

Reporter: Hangshuai He

[1] University of Science and Technology of China,
[2] Huawei Technologies Co., Ltd.
[3] ByteDance Ltd.
[4] Hunan University

Primary research interests:

- Program Languages, Computer Systems, Parallel Computing
- Smart IoT, Smart Sensing, Smart Unmanned Systems for Sensing

Recent Papers:

- **[CGO' 25]** *GoFree: Reducing Garbage Collection via Compiler-inserted Freeing*
- **[TCAD' 25]** *PauliForest: Connectivity-Aware Synthesis and Pauli-Oriented Qubit Mapping for Near Term Quantum Simulation*
- **[OOPSLA' 24]** *MEA2: a Lightweight Field-Sensitive Escape Analysis with Points-to Calculation for Golang*
- **[DAC' 24]** *Crop: An Analytical Cost Model for Cross-Platform Performance Prediction of Tensor Programs*

Corresponding Author：

Professor Yu Zhang

IEEE Fellow Yanyong Zhang

Lab for Intelligent Networking and Knowledge Engineering

Outline

- *Background*
- *System Design*
- *Evaluation*
- *Conclusion*

Outline

- *Background*
- *System Design*
- *Evaluation*
- *Conclusion*

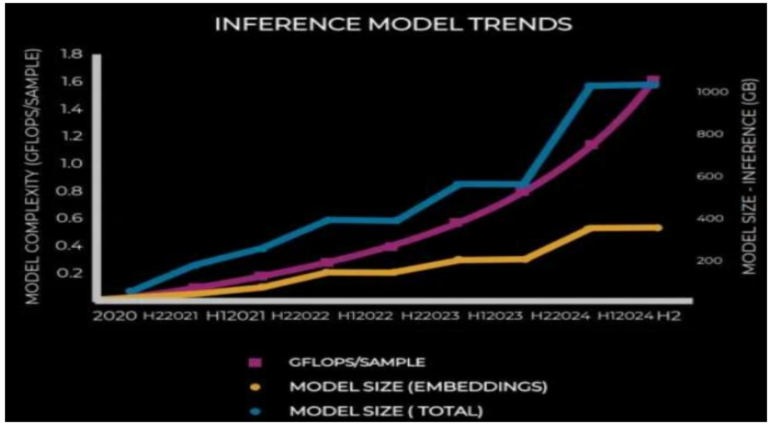
Background

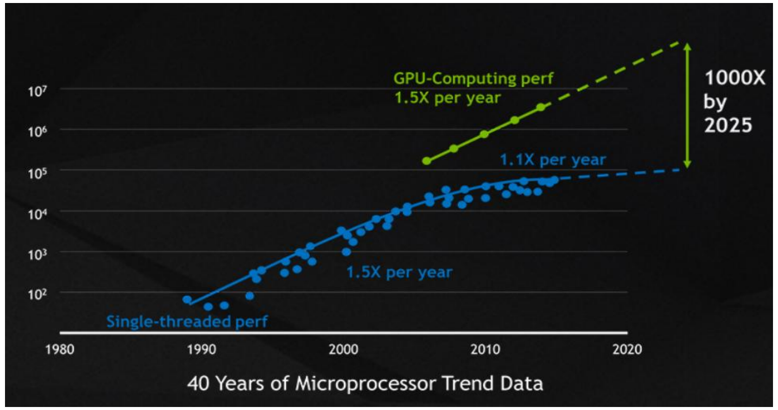
The Growing Gap:
Exponential DL Demand vs. Linear Hardware Scaling

Scale

complexity

>





Background

Low-latency execution of workloads is ***imminent***

[OSDI' 18] *TVM: An Automated End-to-End Optimizing Compiler for Deep Learning*

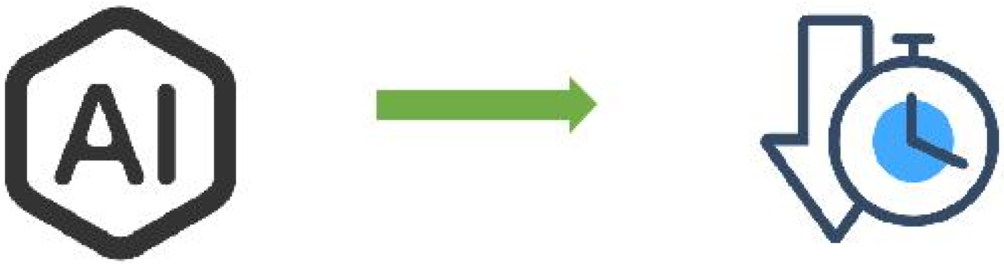
[TOG' 19] *Learning to optimize **halide** with tree search and random programs*

[ASPLOS' 20] ***Flextensor:*** *An automatic schedule exploration and optimization framework for tensor computation on heterogeneous system*

[ATC' 19] *Optimizing {CNN} model inference on {CPUs}*

Background

kernel libraries is ***costly***, tensor compilers are becoming popular



cuDNN oneDNN

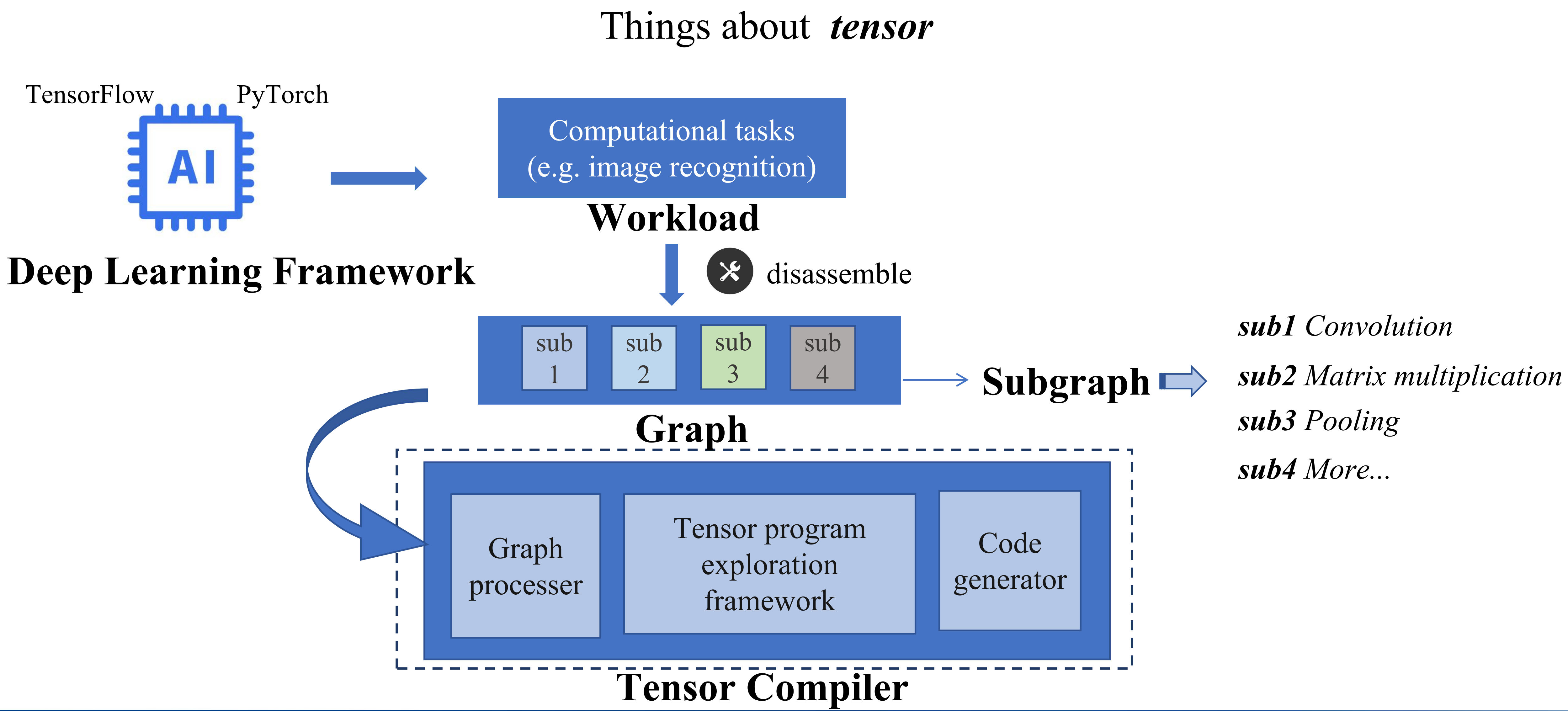
[OSDI' 18] *TVM: An Automated End-to-End Optimizing Compiler for Deep Learning*

[TOG' 19] *Learning to optimize **halide** with tree search and random programs*

[ASPLOS' 20] ***Flextensor:** An automatic schedule exploration and optimization framework for tensor computation on heterogeneous system*

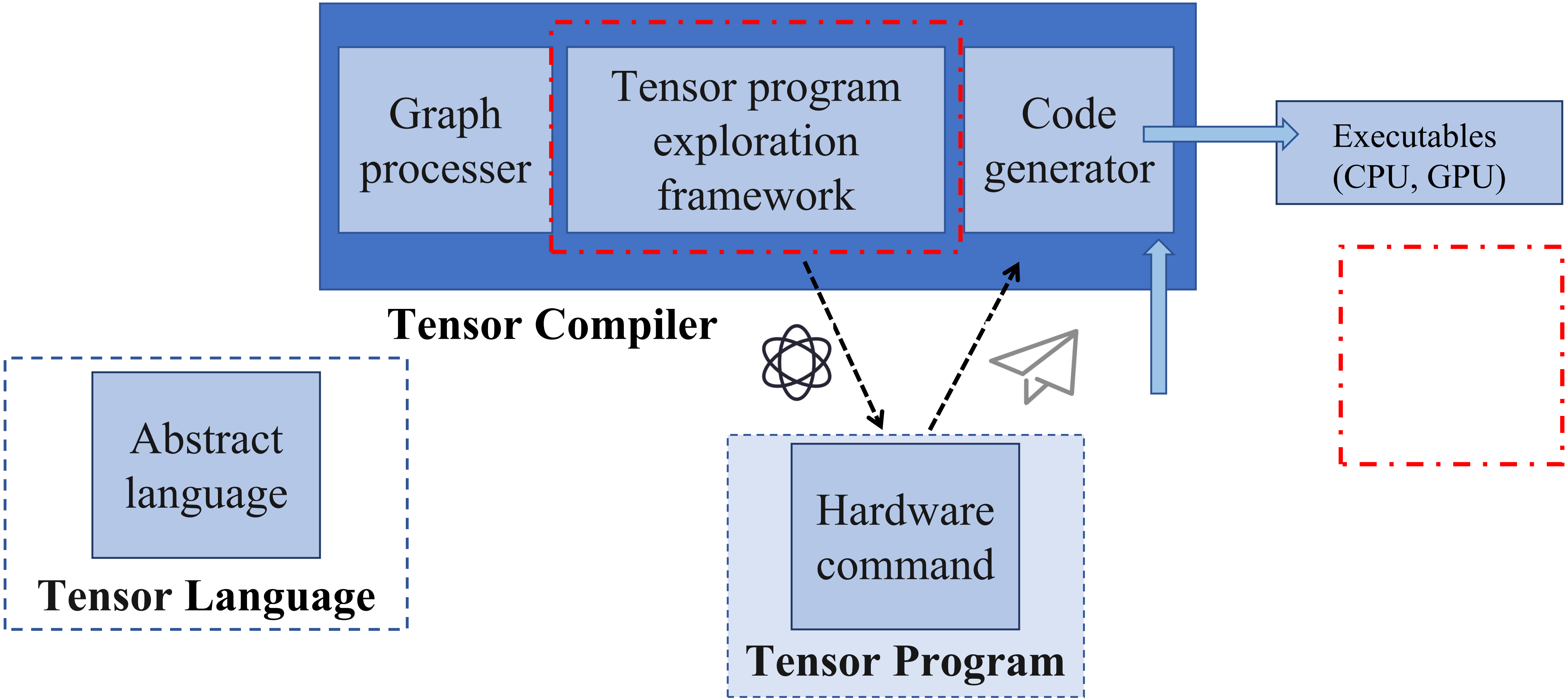
[ATC' 19] *Optimizing {CNN} model inference on {CPUs}*

Background


Things about *tensor*
TensorFlow
PyTorch
AI
Deep Learning Framework
Computational tasks
(e.g. image recognition)
Workload
disassemble
sub 1
sub 2
sub 3
sub 4
Graph
Subgraph
Graph processer
Tensor program exploration framework
Code generator
Tensor Compiler
sub1 Convolution
sub2 Matrix multiplication
sub3 Pooling
sub4 More...

Background

Things about *tensor*



Background

In a clear, structured way,to ***document*** computational processes and ***optimization*** decisions

Abstract language

```
import torch
A = torch.rand(1024, 1024)
B = torch.rand(1024, 1024)
C = torch.matmul(A, B)
```

Tensor Language

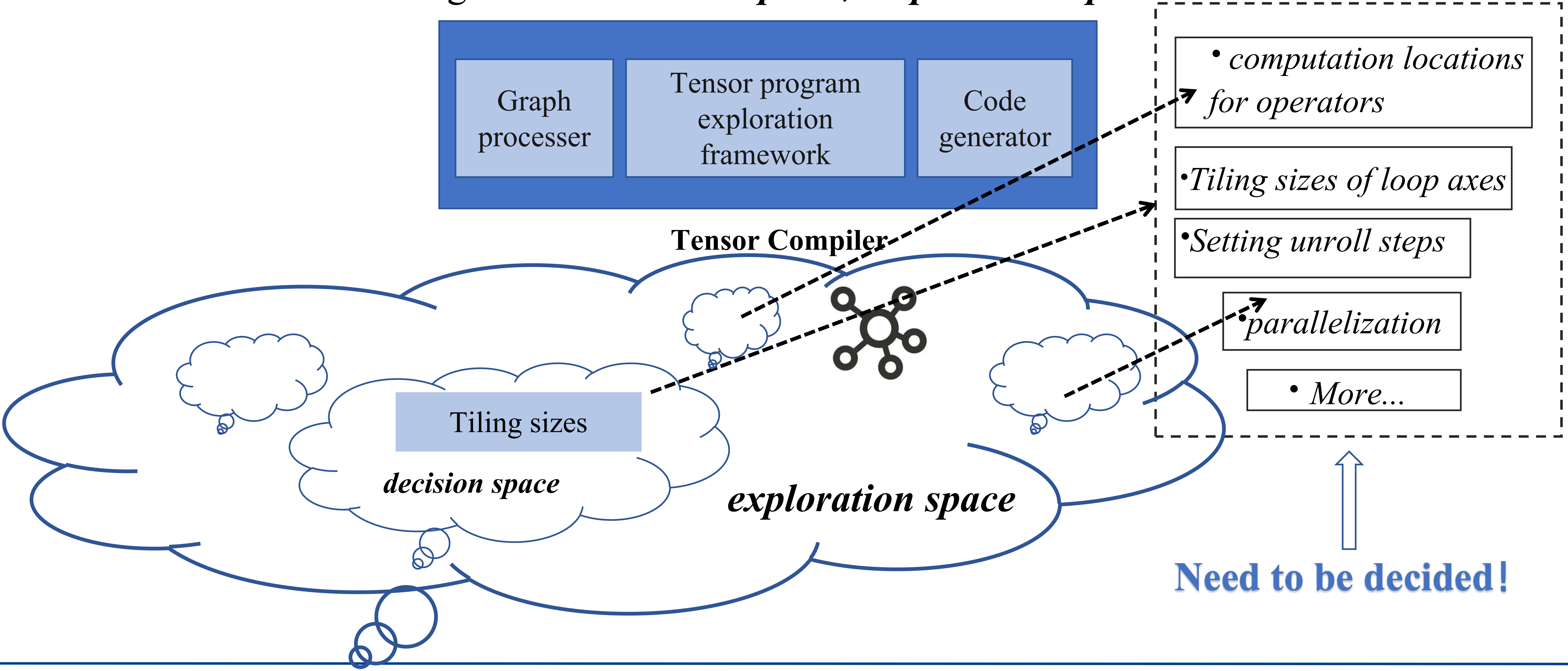
$$C = A \times B$$

MatMul(A, B) ->
Tile(32, 32) ->
Parallelize(Warp-Level) ->
Vectorize(8)

p0 p1 T_matmul_NT p2 T_add 00a059b856ac30ac172b6252254479a6 1024 1024 512 1024 1024 512 1024 512 *llvm -keys=cpu -mcpu=core-avx2 -model=i7 4 64 64 0 0 0 0 0 2* SP 2 0 1024 32 1 4 1 SP 2 4 512 8 1 4 1 SP 2 8 1024 1024 1 RE 2 0 4 1 5 8 2 6 9 3 7 FSP 4 0 0 2 FSP 4 3 1 2 RE 4 0 3 1 4 2 5 CA 2 4 3 PPT SPC 2 0 1024 32 1 4 1 SPC 2 4 512 8 1 4 1 SPC 2 8 1024 1024 1 CLS FU 4 0 1 2 3 AN 4 0 3 PRS 2 PR 2 0 auto_unroll_max_step$0 VECS

Background

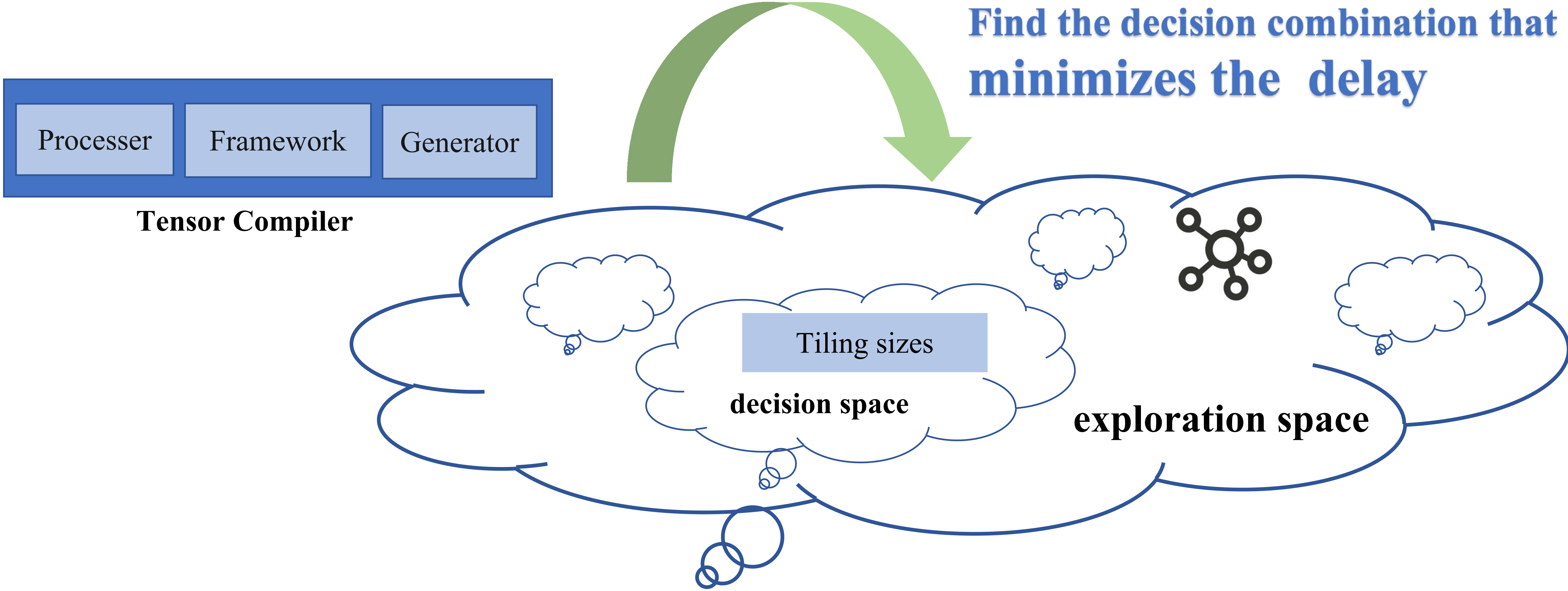
Things about ***decision space***、 ***exploration space***



Background

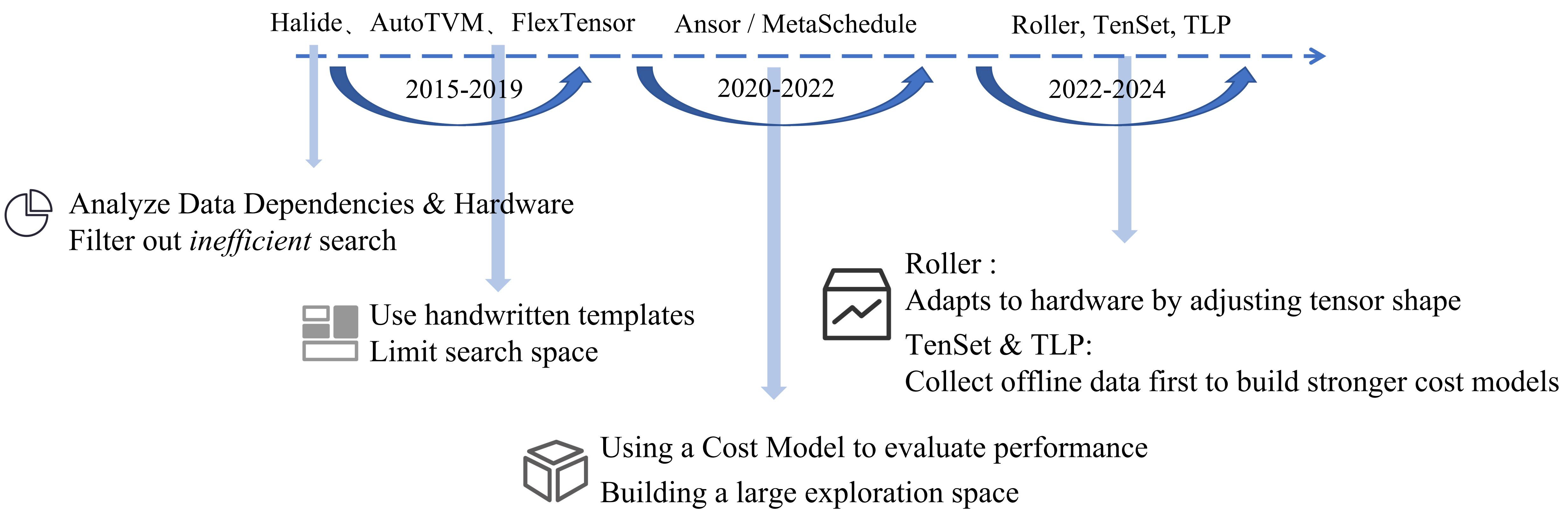
The ***objective*** of tensor program exploration framework

Find the decision combination that minimizes the delay



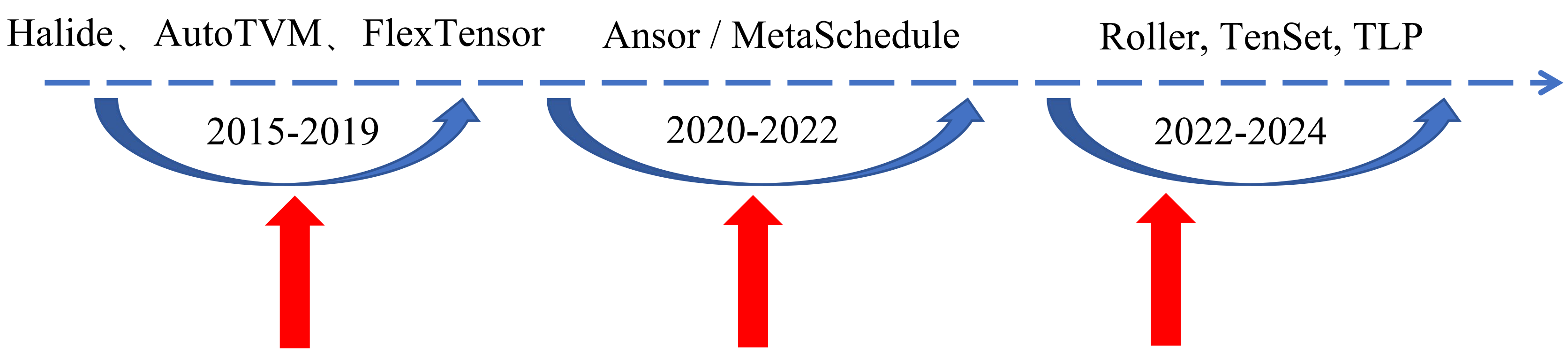
Background

Related SoA work



Background

Related SoA work



Limit [illegible] rules,
less flex [illegible] templates
Limited cost modeling capabilities

Background

Related SoA work

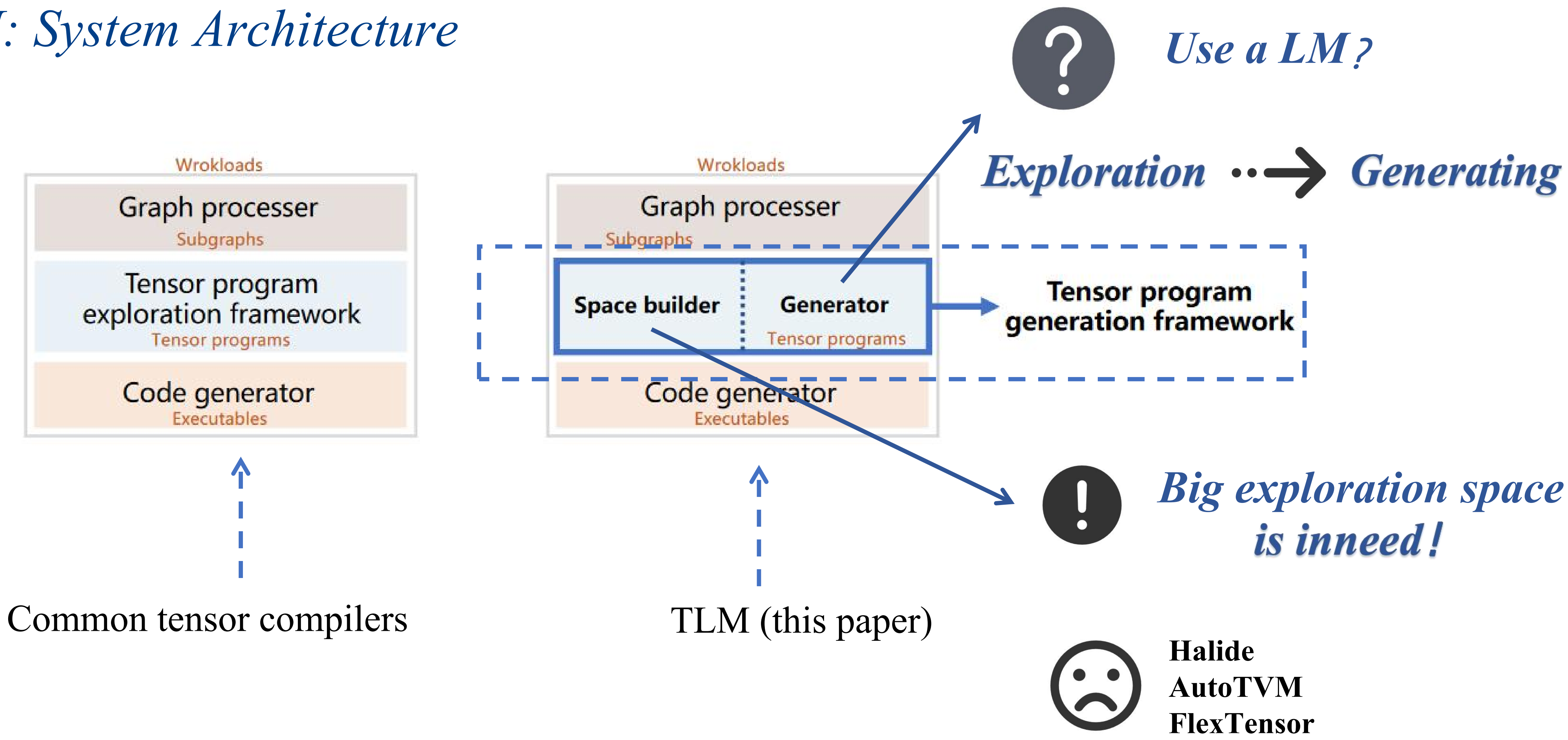
Design Concept	Exploration Orientation	Generation Orientation
Search Methods	Halide, AutoTVM（Templates、Heuristics） Roller（hardware alignment）， Ansor, MetaSchedule（random sampling）	-
Data-driven	TenSet, TLP（Offline Data + Search）	TLM（this paper）

Outline

- *Background*
- *System Design*
- *Evaluation*
- *Conclusion*

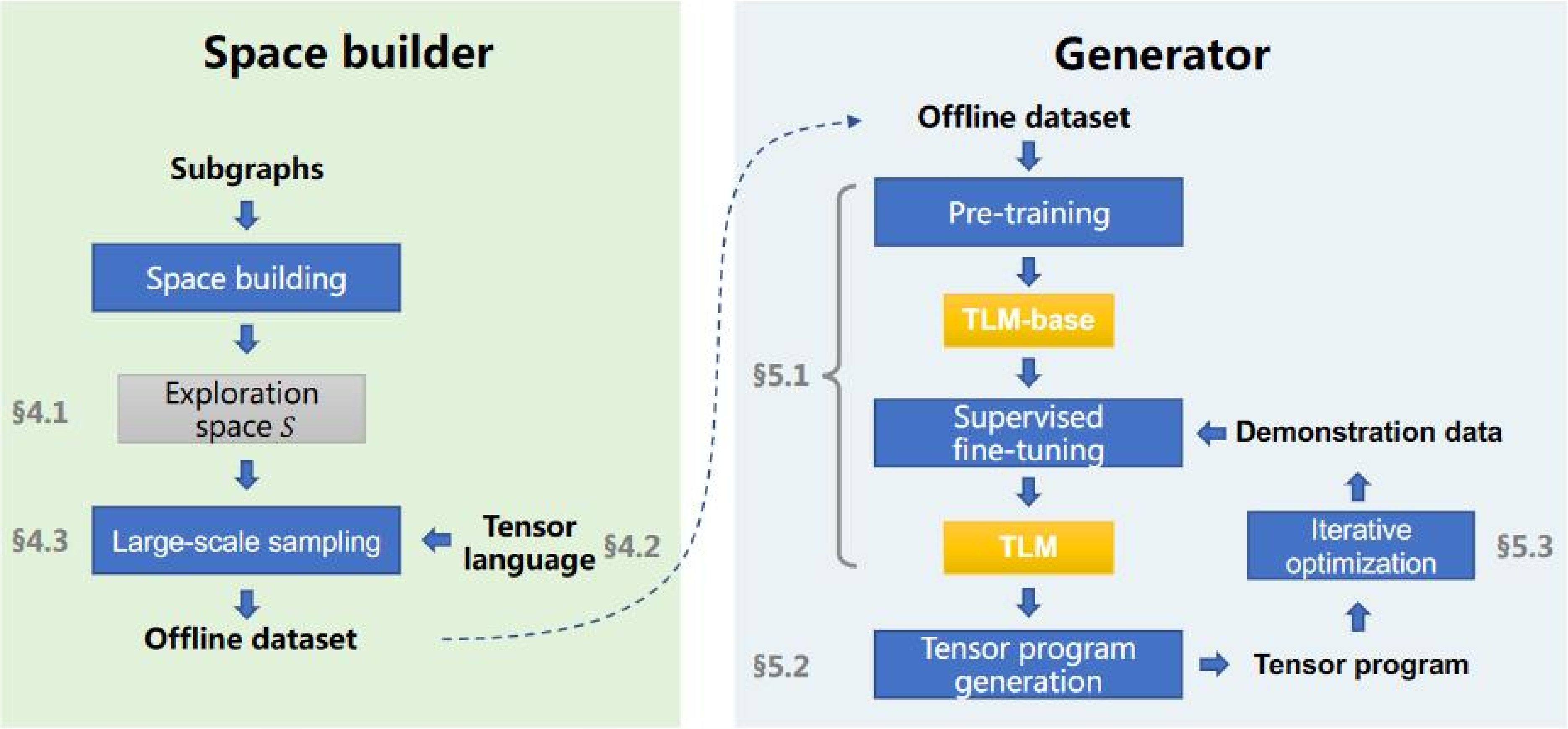
System Design

TLM: System Architecture

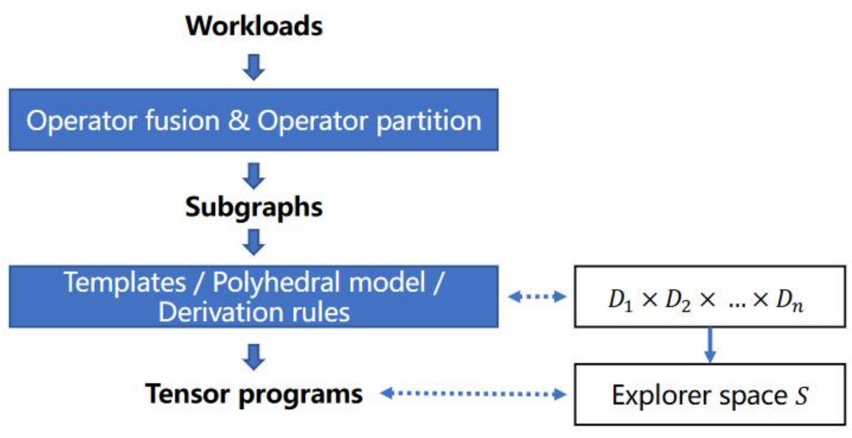


System Design

TLM: System Architecture



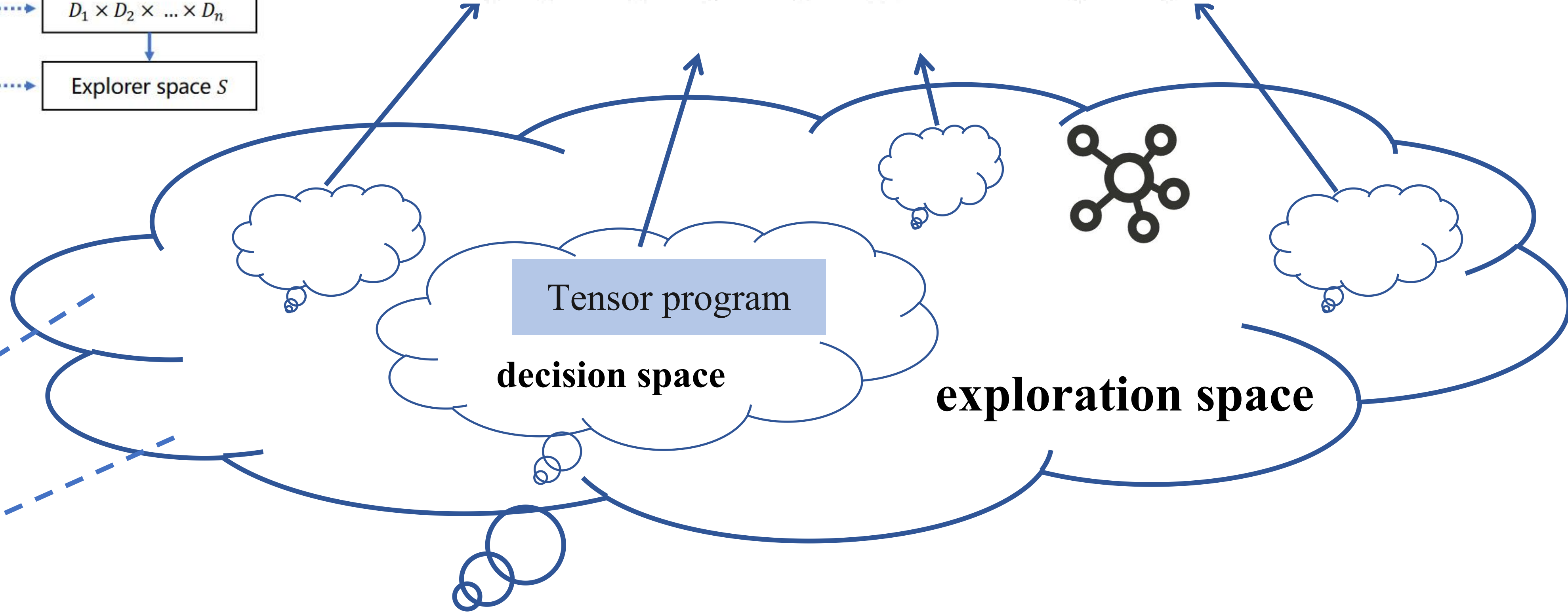
System Design



Exploration Space

Choose which ?

$$|S| = |D_1| \times |D_2| \times \ldots \times |D_n|$$



CPU： 10^6

GPU： 10^9

System Design

Tensor Language

Often exceed
10000 tokens !

Input subgraph

Hardware specifications

Decisions

Maximum
1024 tokens

System Design

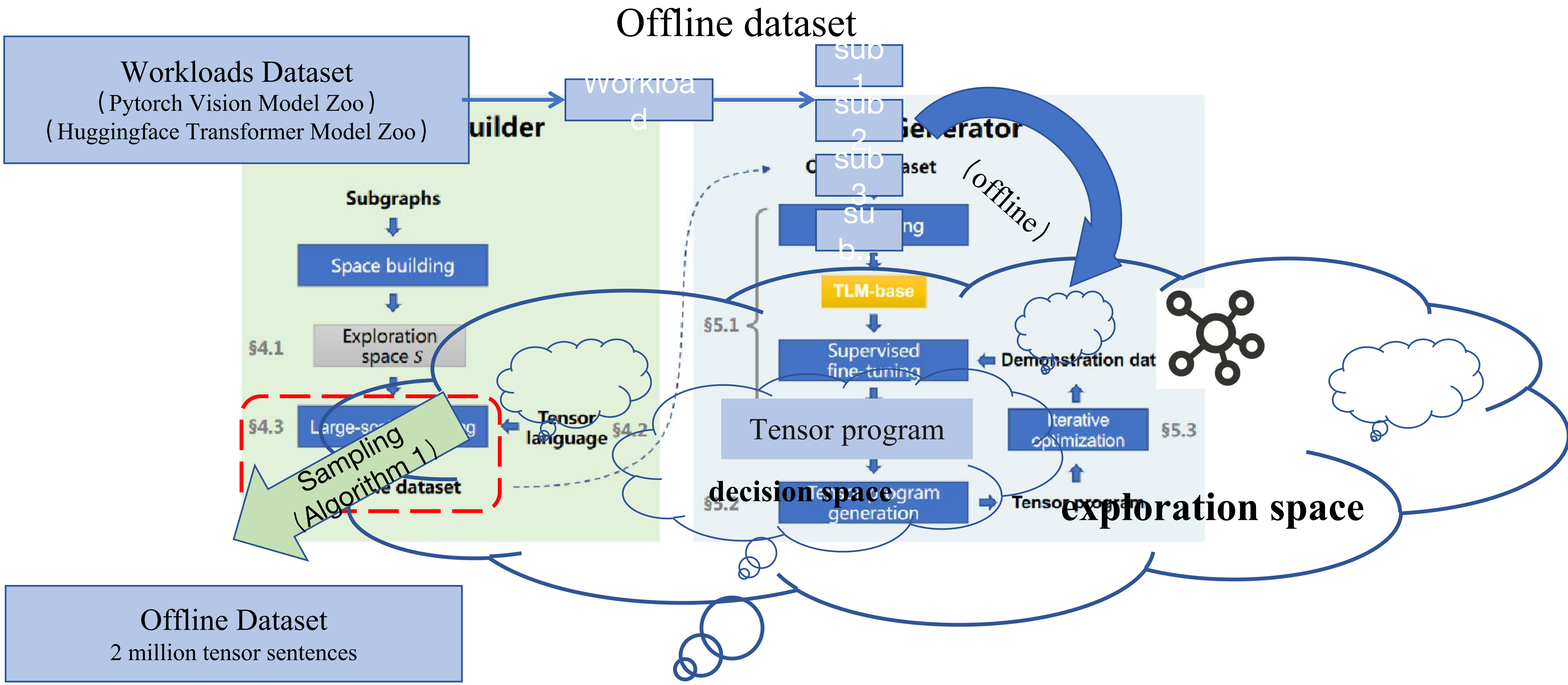
Tensor Language

Algorithm 1: Sampling tensor sentences from decision spaces.

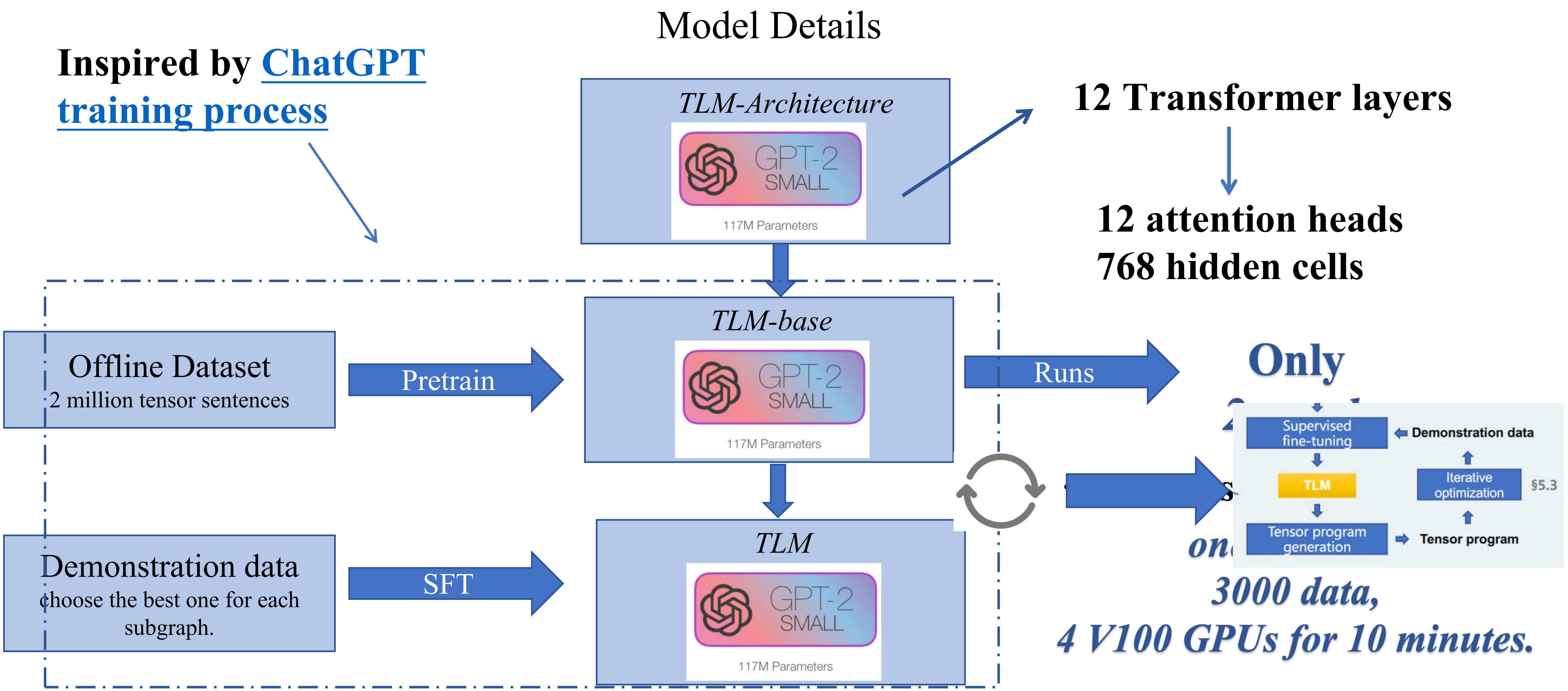
```
Func GenerateSampleData(subgraph, hardware):
    tokens = []
    ExtractTokensFromSubgraph(subgraph, tokens)
    ExtractTokensFromHardware(hardware, tokens)
    decision_spaces = DetermineDecisionSpaces(subgraph,
      hardware)
    foreach space in decision_spaces do
        switch space.type do
            case "tile_size" do
                HandleTileSizeSpace(space, tokens)
            case "unroll" do
                HandleParallelSpace(space, tokens)
            // Additional space types
            case ... do
                ...
    return tokens
Func HandleTileSizeSpace(space, tokens):
    tokens.append("split")
    tokens.extend(Serialize(space.operator))
    tokens.extend(Serialize(space.axis))
    tiles = RandomSample(space)
    tokens.extend(Serialize(tiles))
    // Other properties
```

p0 p1 T_matmul_NT p2 T_add 00a059b856ac30ac172b6252254479a6 1024 1024 512 1024 1024 512 1024 512 *llvm -keys=cpu -mcpu=core-avx2 -model=i7 4 64 64 0 0 0 0 0 2* SP 2 0 1024 32 1 4 1 SP 2 4 512 8 1 4 1 SP 2 8 1024 1024 1 RE 2 0 4 1 5 8 2 6 9 3 7 FSP 4 0 0 2 FSP 4 3 1 2 RE 4 0 3 1 4 2 5 CA 2 4 3 PPT SPC 2 0 1024 32 1 4 1 SPC 2 4 512 8 1 4 1 SPC 2 8 1024 1024 1 CLS FU 4 0 1 2 3 AN 4 0 3 PRS 2 PR 2 0 auto_unroll_max_step$0 VECS

System Design

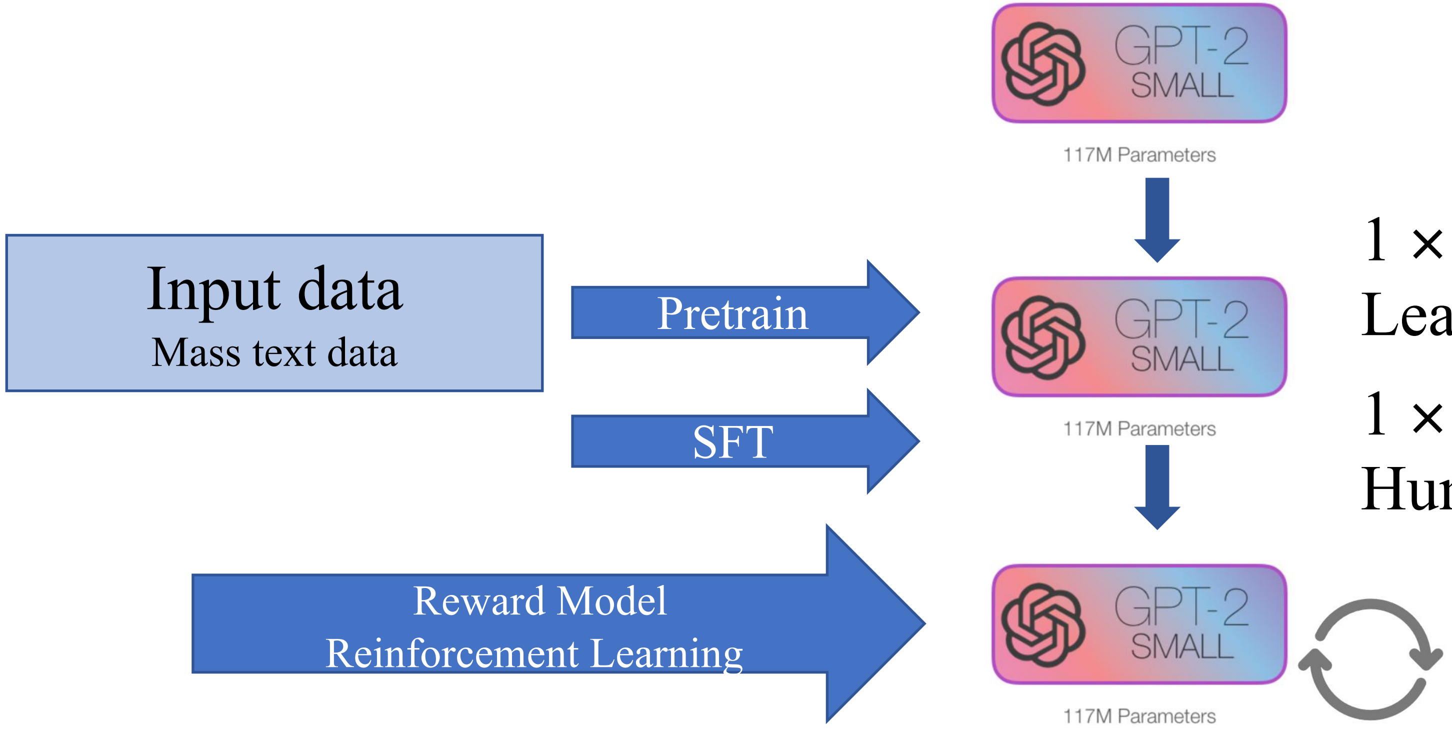


System Design



System Design

ChatGPT training process



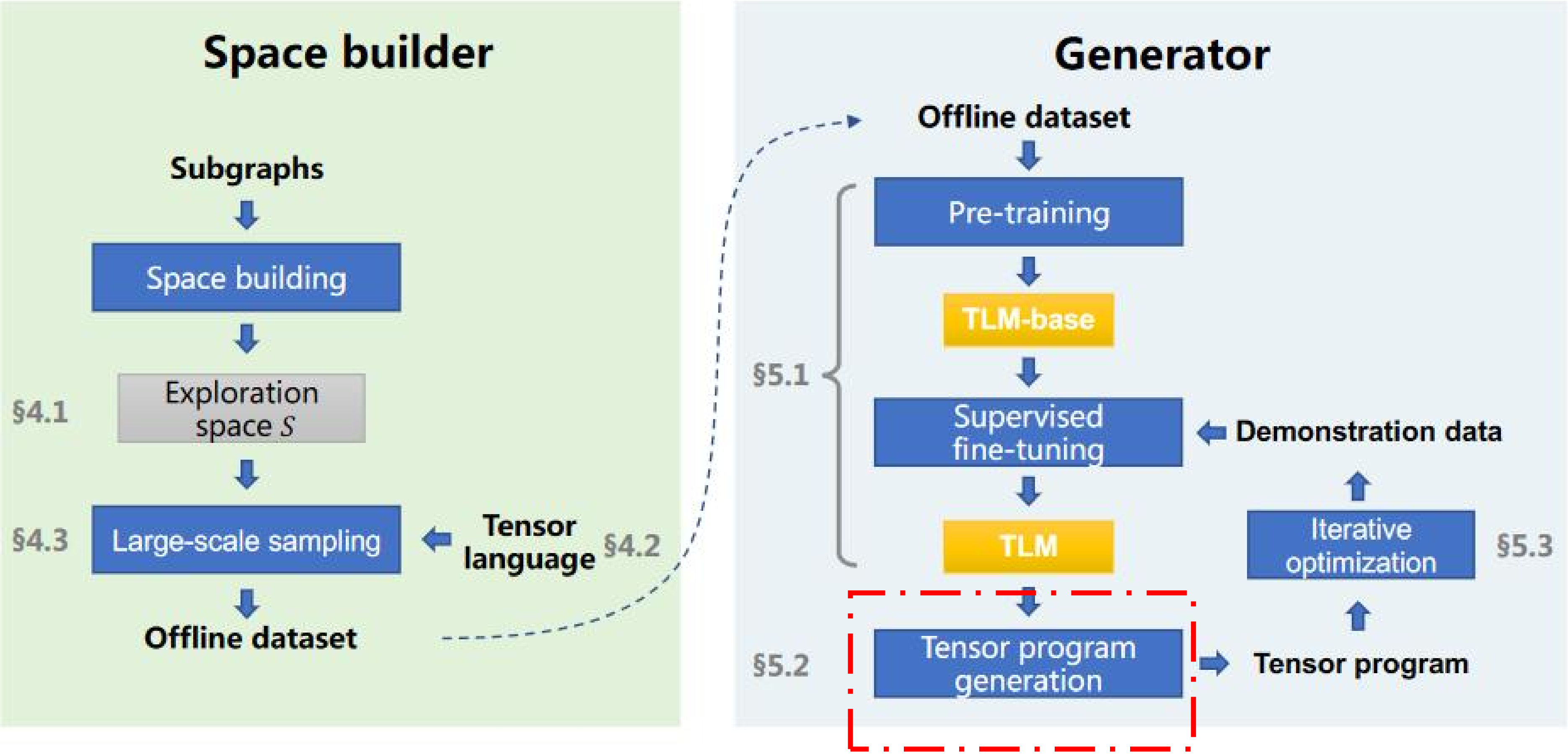
1 × Pre-training:
Learning ***language patterns***

1 × SFT:
Human Demonstration Tuning

····**n times (Reward adjustments)**

System Design

Tensor Program Generation



System Design

Tensor Program Generation

Extract token from subgraph and hardware initialize program

Traversing the decision space

Each time TLM generates a decision, such as "tile size = 32", the framework checks if it is valid!

Algorithm 2: Generating tensor programs aided by TLM in decision-making.

```
Func GenerateTensorProgram(subgraph, hardware):
    tokens = []
    ExtractTokensFromSubgraph(subgraph, tokens)
    ExtractTokensFromHardware(hardware, tokens)
    program = GetInitProgram(subgraph, hardware)
    decision_spaces = DetermineDecisionSpaces(subgraph,
      hardware)
    foreach space in decision_spaces do
        switch space.type do
            case "tile_size" do
                ApplyTileSize(space, tokens, program)
            case "unroll" do
                ApplyParallel(space, tokens, program)
            // Additional space types
            case ... do
                ...
    return program
Func ApplyTileSize(space, tokens, program):
    tokens.append("split")
    tokens.extend(Serialize(space.operator))
    tokens.extend(Serialize(space.axis))
    response_tokens = TLM(tokens)
    tiles = ConvertTokensToTiles(response_tokens)
    if not CheckValidTiles(space, tiles) then
        raise Exception("Invalid Tensor Program")
    tokens.extend(Serialize(tiles))
    program.apply(space.operator, space.axis, tiles)
    // Other properties
```

Tensor Program Generation

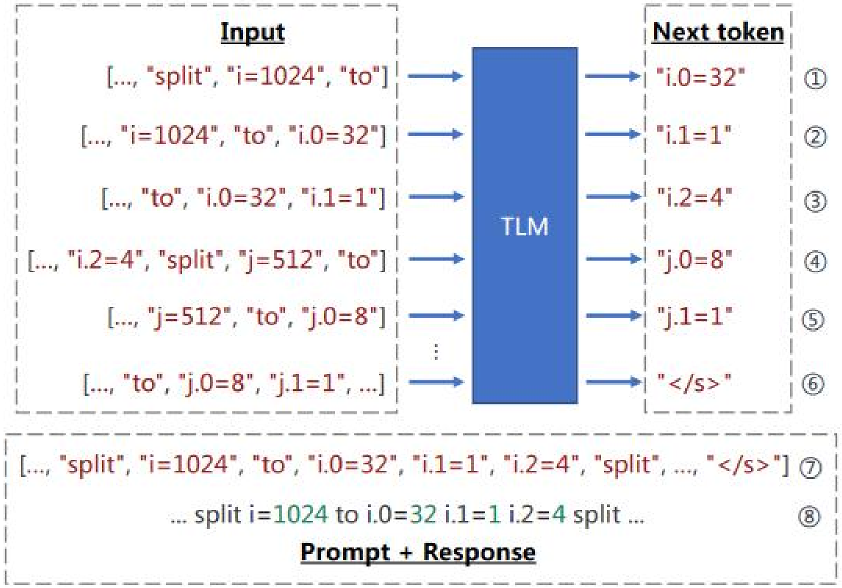


Figure 7: Generating a tensor sentence for a matrix multiplication operator with dimensions $m = 1024$, $n = 512$, and $k = 1024$, with the tiling size component depicted therein.

p0 p1 T_matmul_NT p2 T_add 00a059b856ac30ac172b6252254479a6 1024 1024 512 1024 1024 512 1024 512 llvm -keys=cpu -mcpu=core-avx2 -model=i7 4 64 64 0 0 0 0 0 2 SP 2 0 1024 32 1 4 1 SP 2 4 512 8 1 4 1 SP 2 8 1024 1024 1 RE 2 0 4 1 5 8 2 6 9 3 7 FSP 4 0 0 2 FSP 4 3 1 2 RE 4 0 3 1 4 2 5 CA 2 4 3 PPT SPC 2 0 1024 32 1 4 1 SPC 2 4 512 8 1 4 1 SPC 2 8 1024 1024 1 CLS FU 4 0 1 2 3 AN 4 0 3 PRS 2 PR 2 0 auto_unroll_max_step$0 VECS

Algorithm 2: Generating tensor programs aided by TLM in decision-making.

```
Func GenerateTensorProgram(subgraph, hardware):
    tokens = []
    ExtractTokensFromSubgraph(subgraph, tokens)
    ExtractTokensFromHardware(hardware, tokens)
    program = GetInitProgram(subgraph, hardware)
    decision_spaces = DetermineDecisionSpaces(subgraph,
      hardware)
    foreach space in decision_spaces do
        switch space.type do
            case "tile_size" do
                ApplyTileSize(space, tokens, program)
            case "unroll" do
                ApplyParallel(space, tokens, program)
            // Additional space types
            case ... do
                ...
    return program
Func ApplyTileSize(space, tokens, program):
    tokens.append("split")
    tokens.extend(Serialize(space.operator))
    tokens.extend(Serialize(space.axis))
    response_tokens = TLM(tokens)
    tiles = ConvertTokensToTiles(response_tokens)
    if not CheckValidTiles(space, tiles) then
        raise Exception("Invalid Tensor Program")
    tokens.extend(Serialize(tiles))
    program.apply(space.operator, space.axis, tiles)
    // Other properties
```

System Design

Iterative optimization

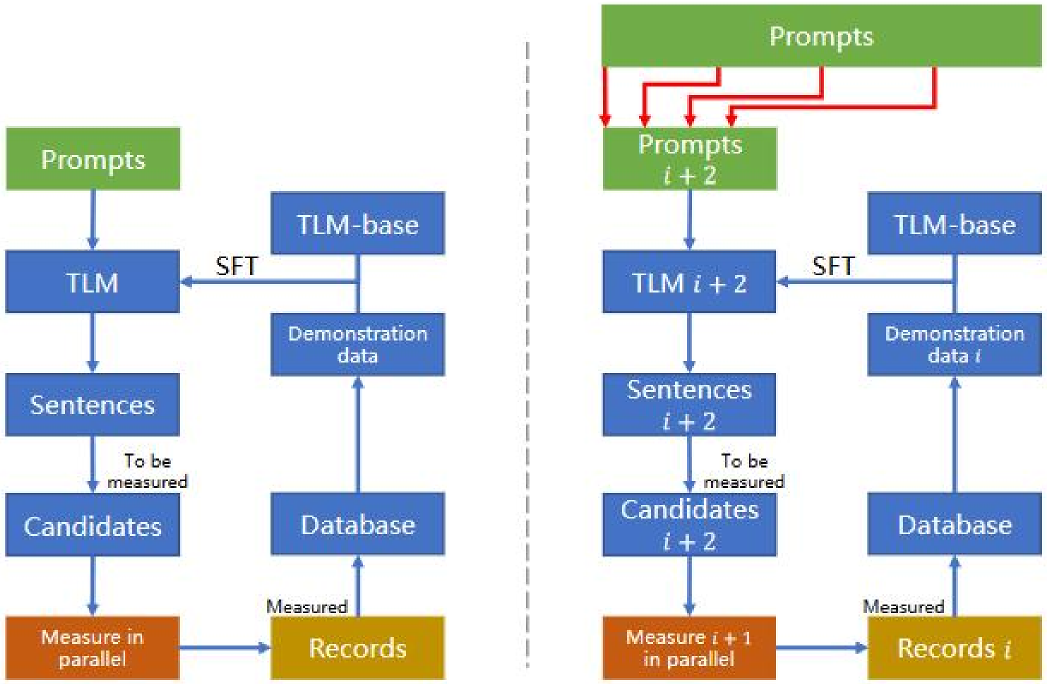


Figure 8: Flowchart of the iterative optimization process.

System Design

Why “i + 2”

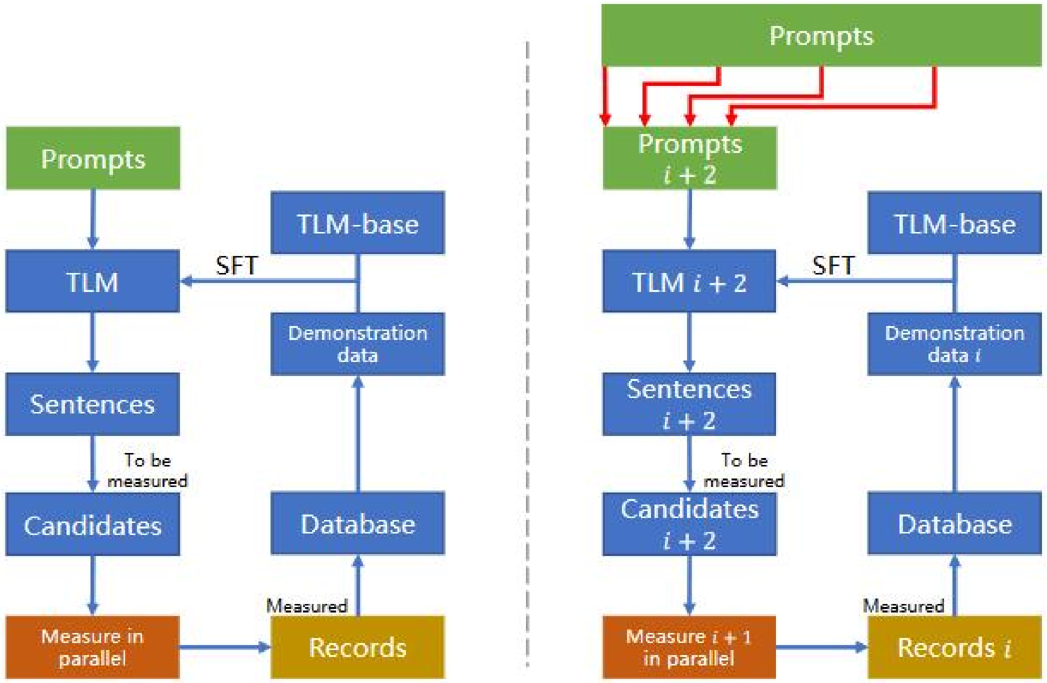
instead of “i + 1” or “i”?

use batch i
to generate

use data i-1
to fine tune

T = i

The resulting model will be used at time T = i + 1 so named TLM i+1



时间	测量进程（左）	训练进程（右）	TLM
T=0	测量 Batch 0，获得示例数据 0	**训练还没开始**	**TLM-base**
T=1	测量 Batch 1，获得示例数据 1	用示例数据 0 训练，得到 TLM 2	**TLM 2**
T=2	测量 Batch 2，获得示例数据 2	用示例数据 1 训练，得到 TLM 3	**TLM 3**
T=3	测量 Batch 3，获得示例数据 3	用示例数据 2 训练，得到 TLM 4	**TLM 4**

System Design

Iterative optimization

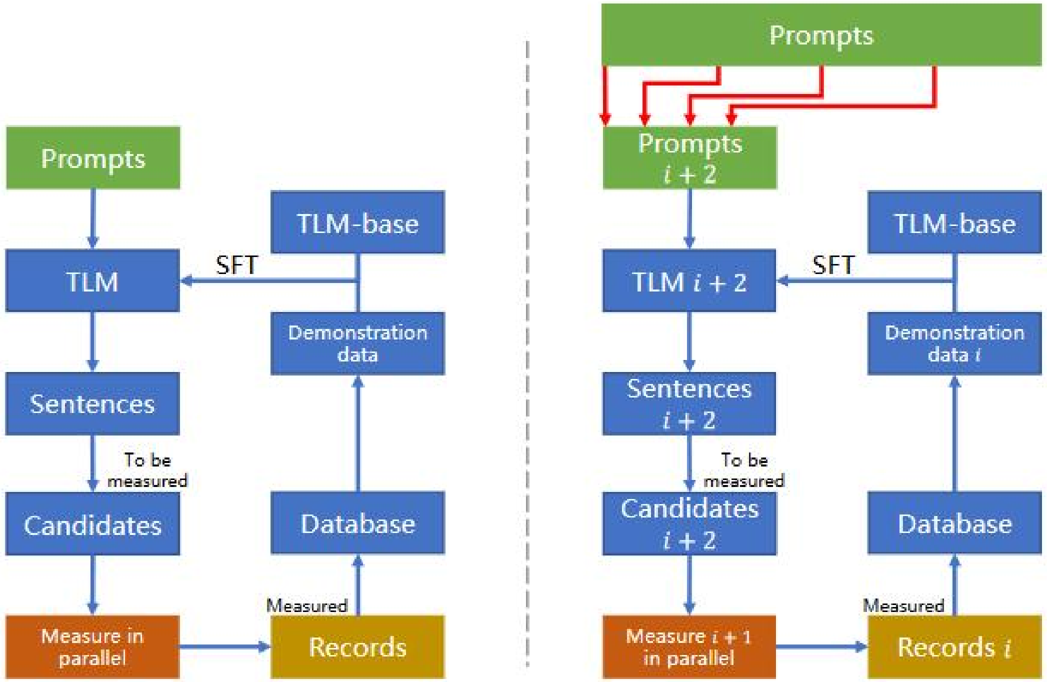


Figure 8: Flowchart of the iterative optimization process.

Outline

- *Background*
- *System Design*
- ***Evaluation***
- *Conclusion*

Evaluation

Experimental Settings

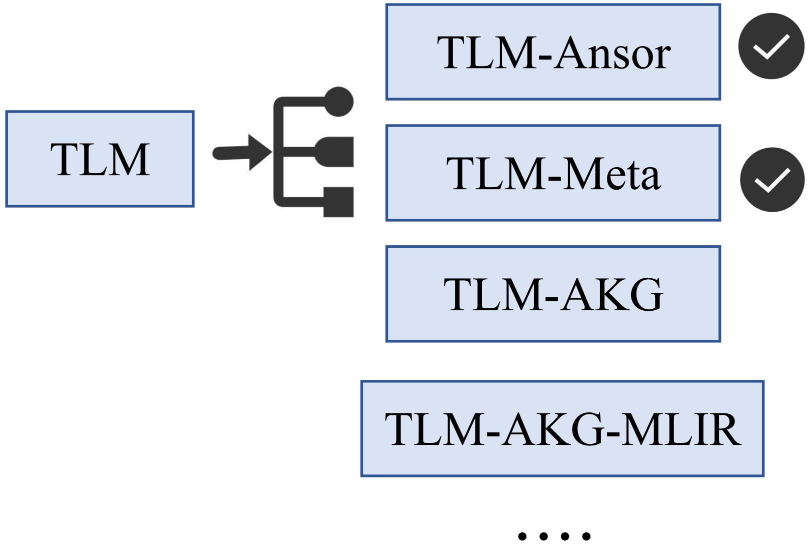


Table 1: Workloads in the TLM test set.

Model	Input shape	Model	Input shape
ResNet-50 [32]	[1, 3, 224, 224]	DenseNet-121 [33]	[8, 3, 256, 256]
MobileNet-V2 [34]	[1, 3, 224, 224]	BERT-large	[4, 256]
ResNeXt-50 [35]	[1, 3, 224, 224]	Wide-ResNet-50 [36]	[8, 3, 256, 256]
BERT-base [37]	[1, 128]	ResNet3D-18 [38]	[4, 3, 144, 144, 16]
BERT-tiny	[1, 128]	DCGAN [39]	[8, 3, 64, 64]
GPT-2	[1, 128]	LLAMA [40]	[4, 256]

CPU: 4-core Intel i7-10510U, 16GB RAM, AVX2 support, Ubuntu 20.04.

GPU: 48-core Intel Xeon Gold 6226, 376GB RAM, 4 x 32GB NVIDIA V100, Ubuntu 20.04, CUDA 11.6, cuDNN 8.4.0.

Evaluation

Convergence Behavior of Demonstration Data

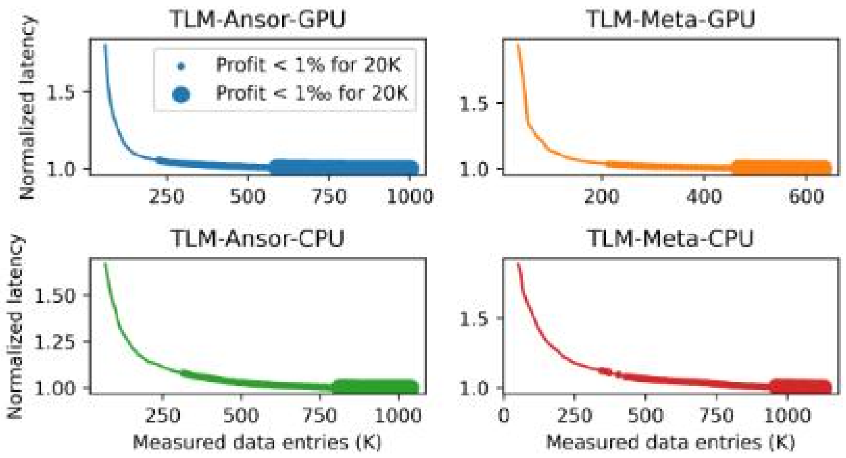


Figure 9: Demonstration data convergence curves of TLM-Ansor and TLM-Meta on the GPU and the CPU.

Convergence： as a performance improvement of less than 1% after measuring 20,000 pieces of data (all subgraphs).

Evaluation

Subgraph Benchmark

Table 2: The overall speedup for the 23 TLM-Ansor subgraphs. The higher the overall speedup, the better. In the table, "Times" represents the measurement times for each subgraph.

		Ansor			TLM-Ansor
	Times	64	1K	10K	10K
TLM-Ansor	1	1.26	0.98	0.92	0.85
	10	1.40	1.08	1.03	0.95
	16	1.43	1.10	1.04	0.96
	32	1.45	1.12	1.06	0.98
	64	1.45	1.12	1.06	0.98
	1K	1.46	1.13	1.07	0.99
	10K	1.48	1.14	1.08	1.00
Ansor	10K	1.37	1.06	1.00	0.92

Table 3: The overall speedup for the 40 TLM-Meta subgraphs.

		MetaSchedule			TLM-Meta
	Times	64	1K	10K	1K
TLM-Meta	1	1.00	0.69	0.68	0.67
	10	1.41	0.96	0.95	0.94
	16	1.45	1.00	0.99	0.97
	32	1.46	1.01	1.00	0.97
	64	1.49	1.02	1.01	0.99
	1K	1.50	1.02	1.01	1.00
MetaSchedule	10K	1.48	1.01	1.00	0.99

Evaluation

End-to-End Workload Benchmark

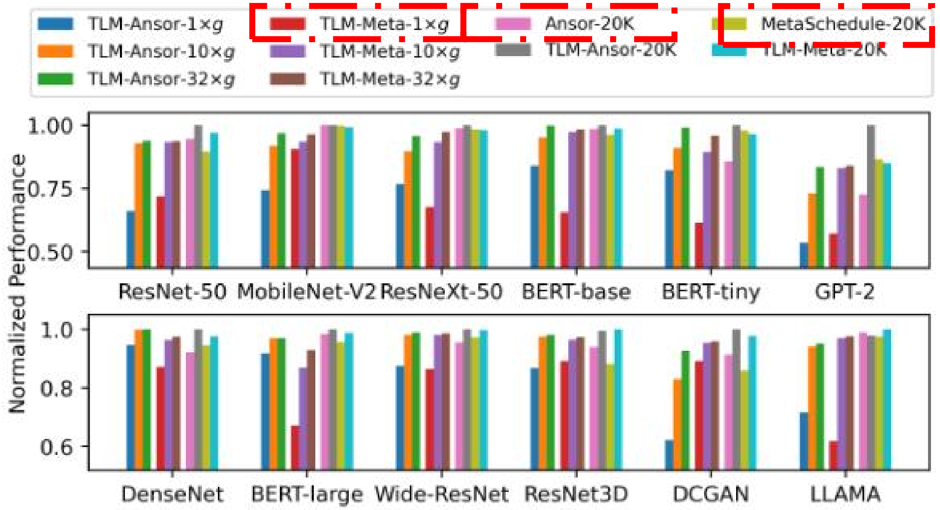


Figure 10: Workload inference performance comparison with Ansor/MetaSchedule on V100.

Evaluation

Comparison with TensorRT/PyTorch on V100

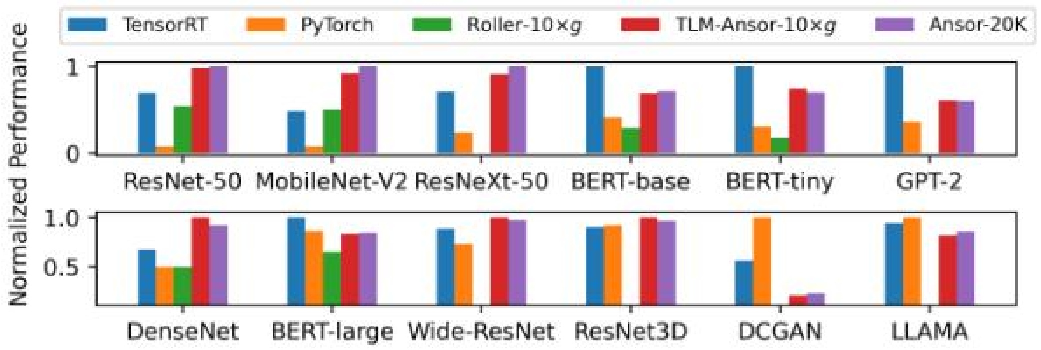


Figure 13: Workload inference performance comparison with TensorRT/PyTorch on V100.

Convergence ： as a performance improvement of less than 1% after measuring 20,000 pieces of data (all subgraphs).

Outline

- *Background*
- *System Design*
- *Evaluation*
- ***Conclusion***

Conclusion

Enabling Tensor Language Model to Assist in Generating High-Performance Tensor Programs for Deep Learning

- **Finding new scenarios to introduce LM.**
- **An LM was trained to apply the new scenario.**
- **To accompany this, a tensor language was designed and very solid framework work was done.**
- **Innovative optimization for very time-consuming traditional scenarios**
- **In the experimental part, the validation was done mainly for the efficiency, low demand, and low training time mentioned.**

Conclusion

➢ 这个paper有什么问题，基于这个paper还能做什么？

- 硬件适配不足：
  - TLM 在 BERT、LLAMA、DCGAN 等任务上落后于 TensorRT/PyTorch，因其生成的张量程序未充分利用硬件特性（如 V100 的 TensorCore、HBM2 带宽）（在矩阵乘法和卷积主导的任务中，TLM 延迟高，竞争力不足。）
  - 【如果参考类似TLM的构造，可以将硬件特性融入张量句子生成，引入硬件感知损失函数？等方面作为一个创新点】

- 实验没有考虑子图间依赖，都是随机挑选种类。
  - 【如果参考TLM的设计的话，还可以，扩展 TLM 预测全局调度策略（？），优化子图间通信（GPU 内存拷贝、共享）。
  - 对于subgraph 引入子图依赖图（dependency graph），TLM 预测整体延迟最优配置。

Conclusion

- 这个paper提到的idea，能不能用在自己的方向/project上面？
  - 文章用到的都是 V100 GPU 和 Intel CPU，没有测试边缘设备，也没有测试过其他硬件（如 NPU、ARM 架构的 CPU）
  - 也许可以优化张量语言，适配 ARM 或 RISC-V 架构。
  - 在边缘场景，也许可以利用 1×g 测量的特性，减少边缘编译时间，支持低功耗推理。
  - 测评指标也许可以加入功耗指标。
- 这个paper能不能泛化？
  - 对于端侧设备来说，尤其是agent，可能优化空间不多。但如果对于具身智能来说，可能需要处理多任务，那么也许可以设计一个多任务张量调度生成框架，统一优化张量程序生成、任务调度和内存分配。
    - 针对 TLM 的离线数据依赖（L1），设计在线自适应机制，利用端侧运行时反馈微调模型。

Enabling Tensor Language Model to Assist in Generating High-Performance Tensor Programs for Deep Learning

Thank you !

Authors： Yi Zhai[1],Sijia Yang[2], Keyu Pan[3], Renwei Zhang[2],Shuo Liu[1]
Chao Liu and Zichun Ye[2],Jianmin Ji[1],Jie Zhao[4],
Yu Zhang and Yanyong Zhang[1]

Reporter: Hangshuai He

[1] University of Science and Technology of China,
[2] Huawei Technologies Co., Ltd.
[3] ByteDance Ltd.
[4] Hunan University

Enabling Tensor Language Model to Assist in Generating High-Performance Tensor Programs for Deep Learning

Authors： Yi Zhai[1],Sijia Yang[2], Keyu Pan[3], Renwei Zhang[2],Shuo Liu[1]
Chao Liu and Zichun Ye[2],Jianmin Ji[1],Jie Zhao[4],
Yu Zhang and Yanyong Zhang[1]

Reporter: Hangshuai He

[1] University of Science and Technology of China,
[2] Huawei Technologies Co., Ltd.
[3] ByteDance Ltd.
[4] Hunan University