Georgia Tech School of Electrical and Computer Engineering
College of Engineering

http://synergy.ece.gatech.edu

ASTRA-SIM Description

Workload Layer

Taekyung Heo
Postdoctoral Fellow, School of ECE
Georgia Institute of Technology
taekyung@gatech.edu

Acknowledgments: William Won (GT), Srinivas Sridharan (Meta), Louis Feng (Meta), Matt Bergeron (Meta), Shengbao Zheng (Meta), Wenyin Fu (Meta), Zhaodong Wang (Meta), Sudarshan Srinivasan (Intel)

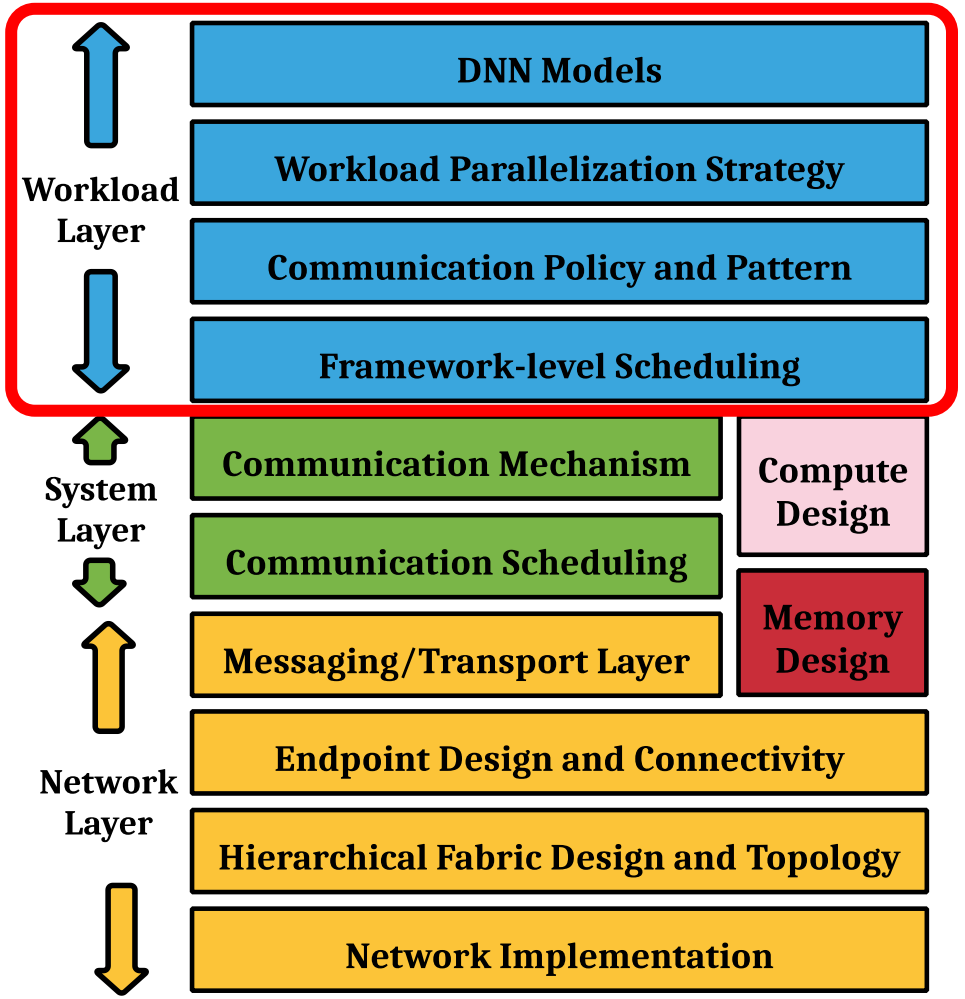
Workload Layer

ASTRA-sim can be broken down into three layers:

1. **Workload layer (the training loop):**
   - **Parallelism approach**
   - **Compute power**
   - **Communication size & type and dependency order**

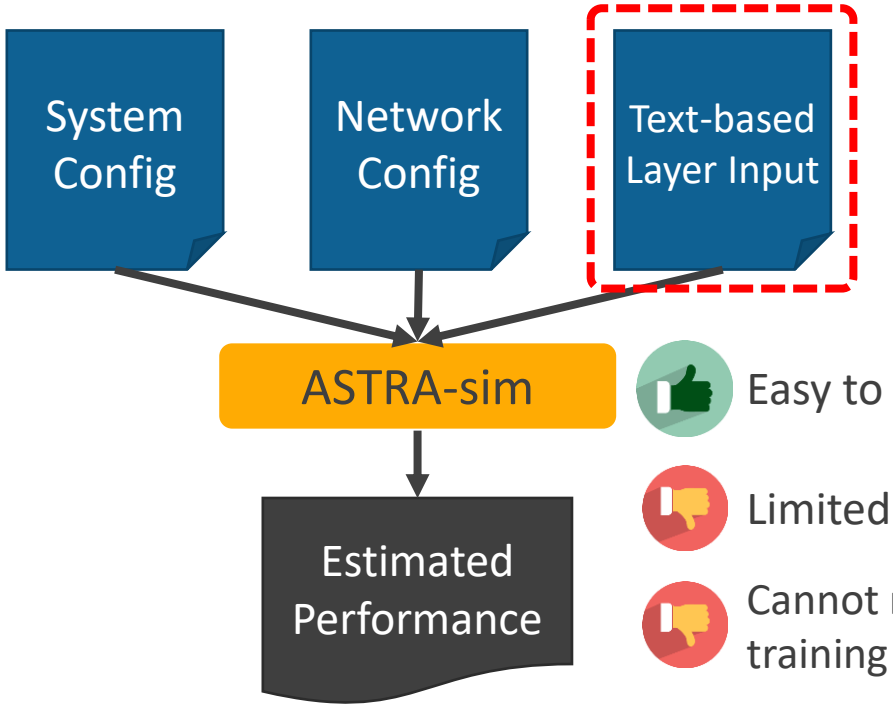
2. System layer:
   - Collective communication algorithm
   - Chunk size, schedule of collectives

3. Network layer:
   - Physical topology
   - Congestion control, communication protocol
   - Link BW, latency, buffers, routing algorithm



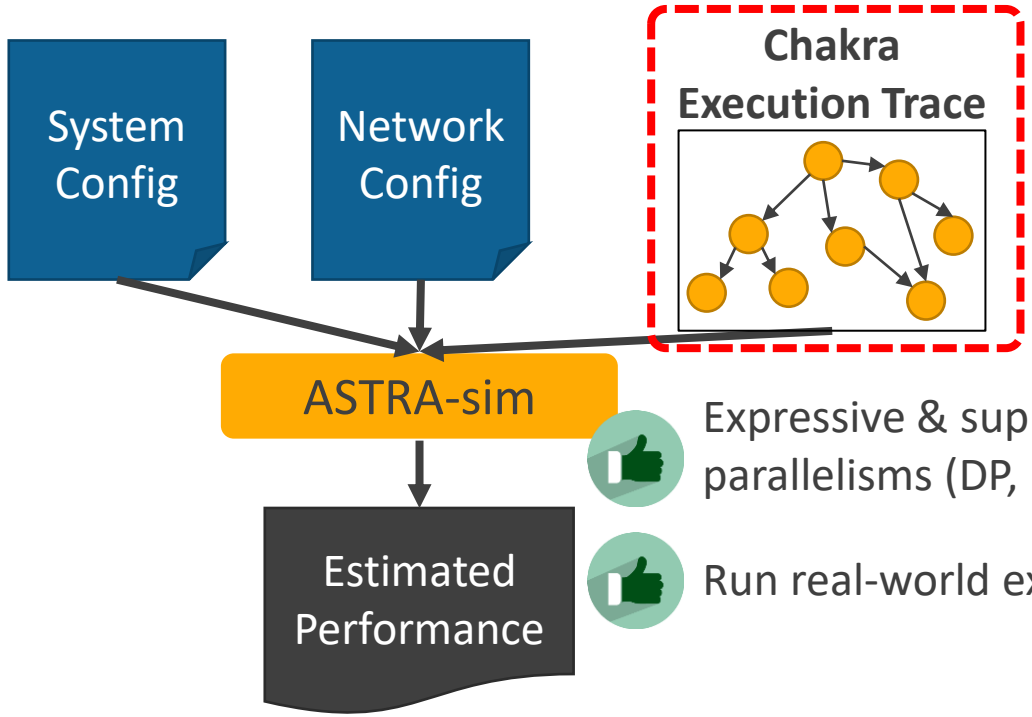
Two Types of Workload Layers

Text-based Workload Layer (ASTRA-sim 1.0, master branch)



- Easy to write
- Limited expressiveness
- Cannot run complex training loops (pipeline)

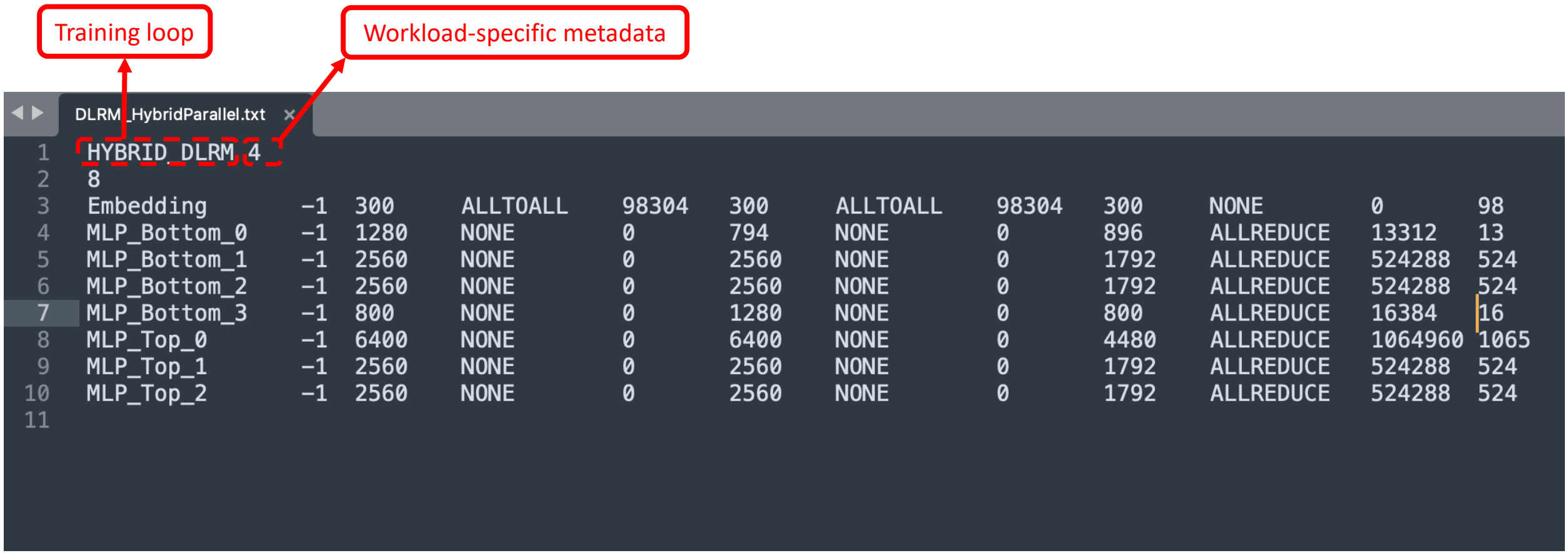
Graph-based Workload Layer (ASTRA-sim 2.0, Chakra branch)



- Expressive & supports all parallelisms (DP, MP, PP)
- Run real-world exec traces

Text-based Workload Input

- A model is described layer-by-layer



Text-based Workload Input

- A model is described layer-by-layer

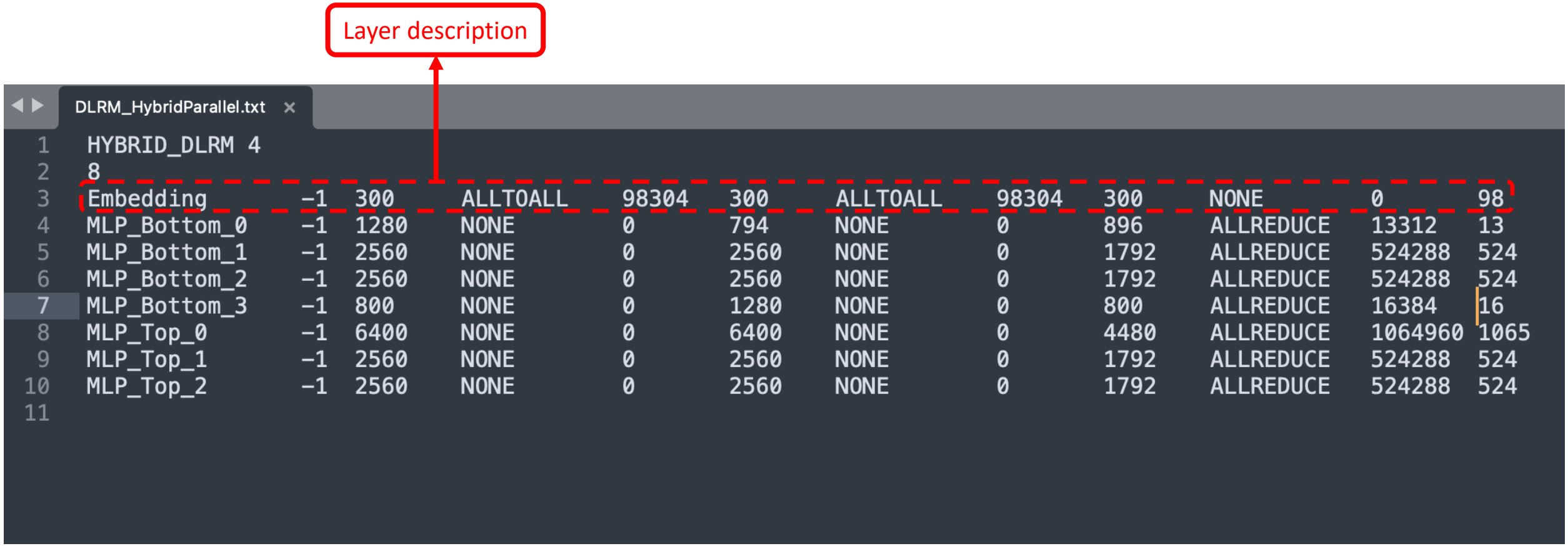
# of layers

DLRM_HybridParallel.txt

```
1  HYBRID_DLRM 4
2  8
3  Embedding       -1  300   ALLTOALL  98304  300   ALLTOALL  98304  300   NONE       0        98
4  MLP_Bottom_0    -1  1280  NONE      0      794   NONE      0      896   ALLREDUCE  13312    13
5  MLP_Bottom_1    -1  2560  NONE      0      2560  NONE      0      1792  ALLREDUCE  524288   524
6  MLP_Bottom_2    -1  2560  NONE      0      2560  NONE      0      1792  ALLREDUCE  524288   524
7  MLP_Bottom_3    -1  800   NONE      0      1280  NONE      0      800   ALLREDUCE  16384    16
8  MLP_Top_0       -1  6400  NONE      0      6400  NONE      0      4480  ALLREDUCE  1064960  1065
9  MLP_Top_1       -1  2560  NONE      0      2560  NONE      0      1792  ALLREDUCE  524288   524
10 MLP_Top_2       -1  2560  NONE      0      2560  NONE      0      1792  ALLREDUCE  524288   524
11
```

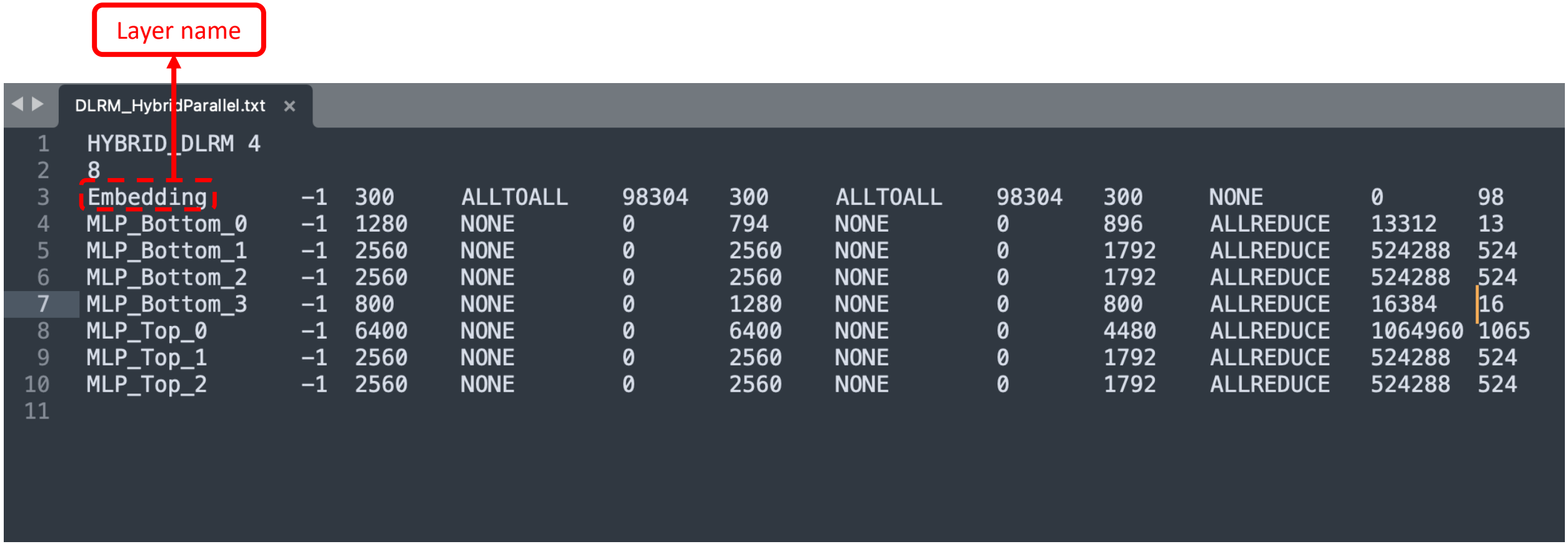
Text-based Workload Input

- A model is described layer-by-layer



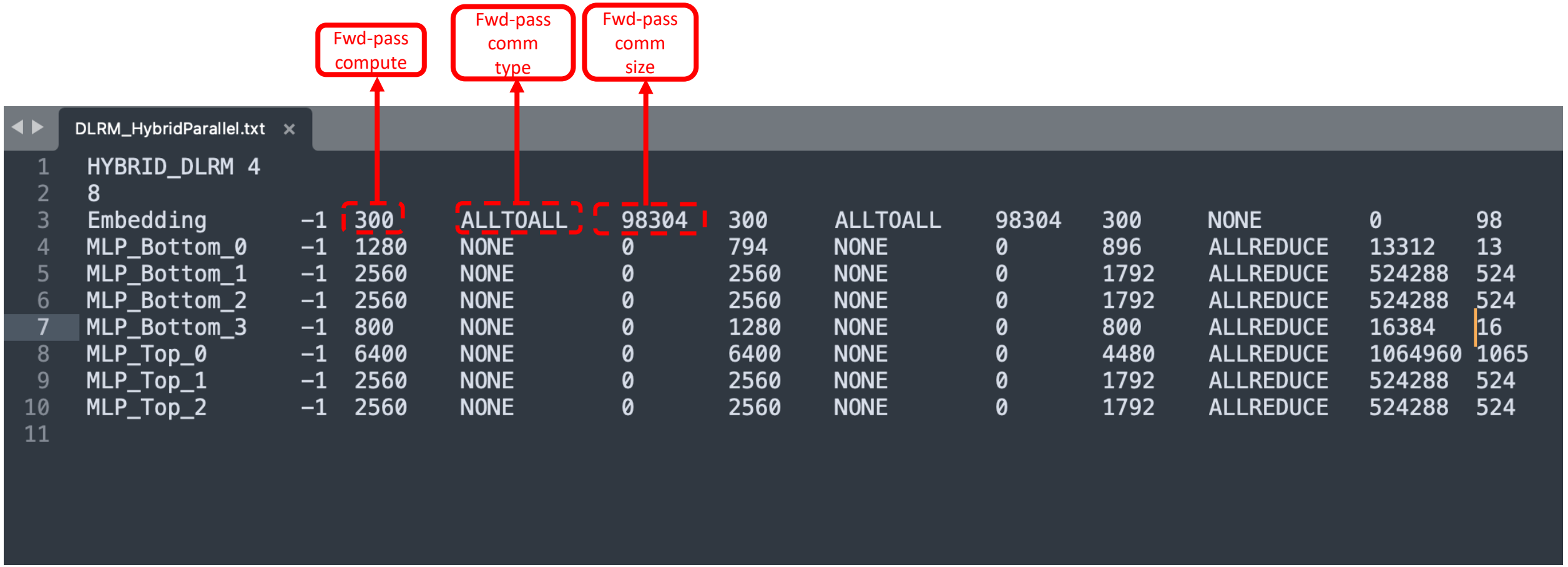
Text-based Workload Input

- A model is described layer-by-layer



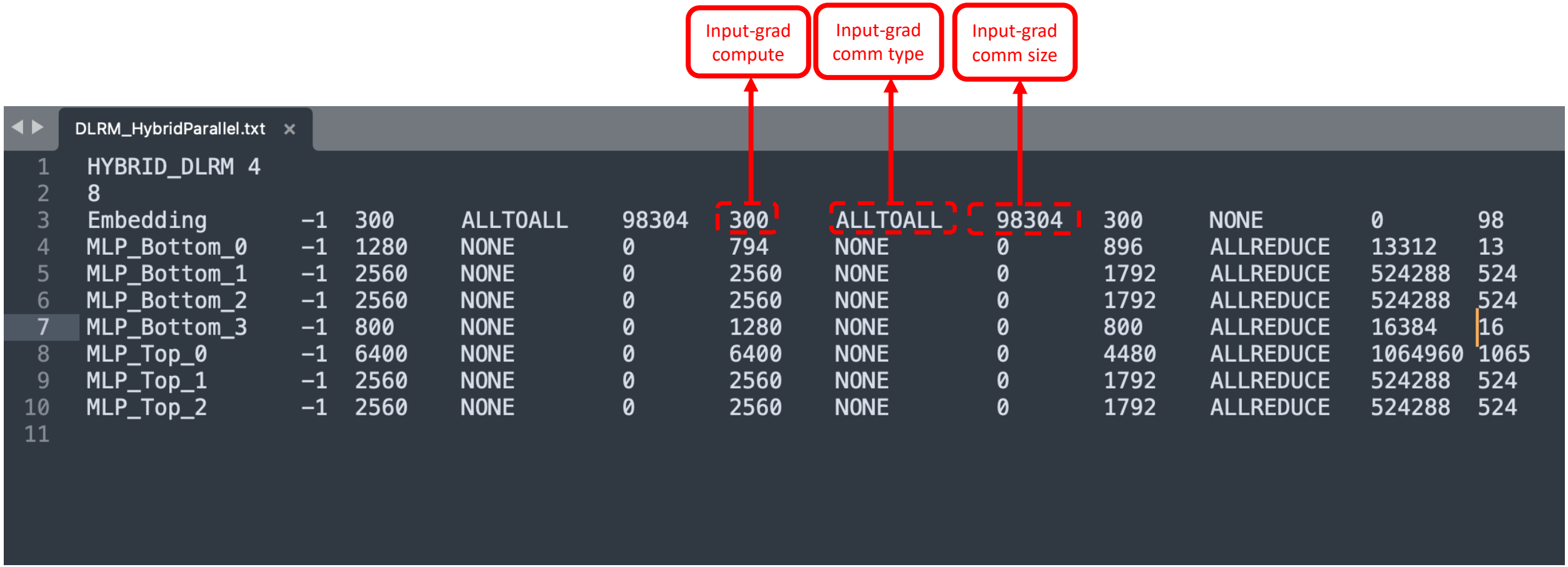
Text-based Workload Input

- A model is described layer-by-layer



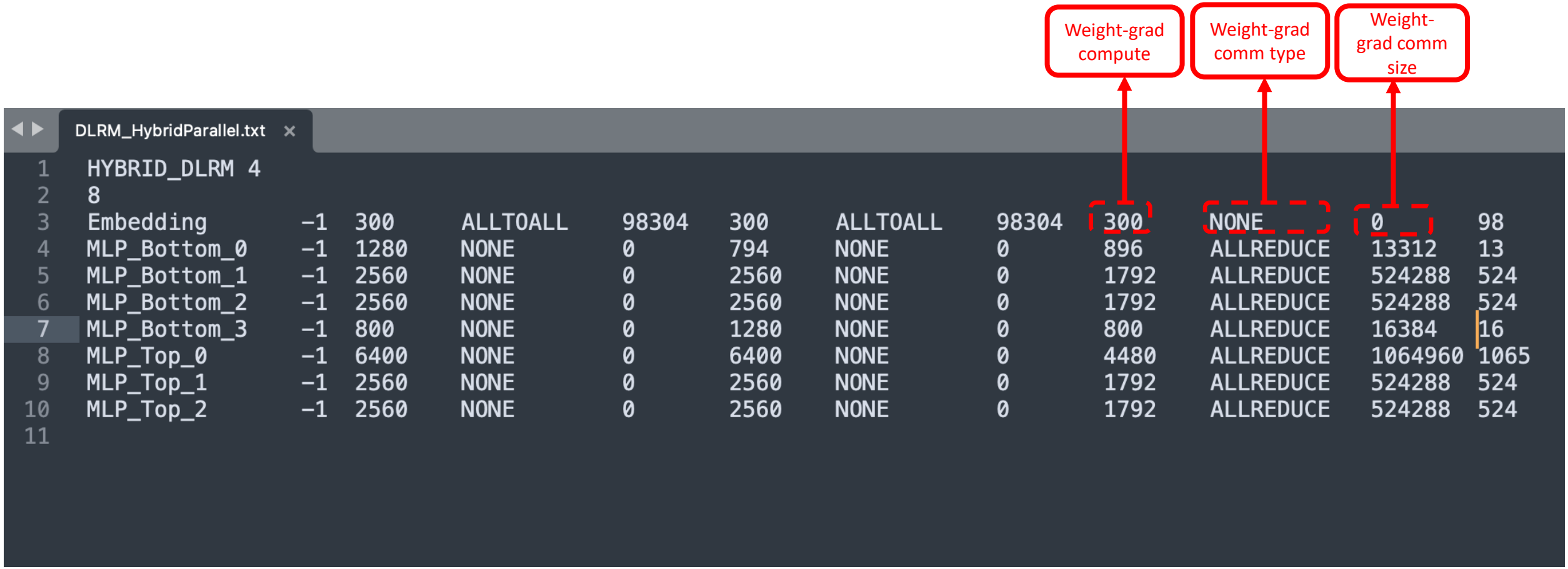
Text-based Workload Input

- A model is described layer-by-layer



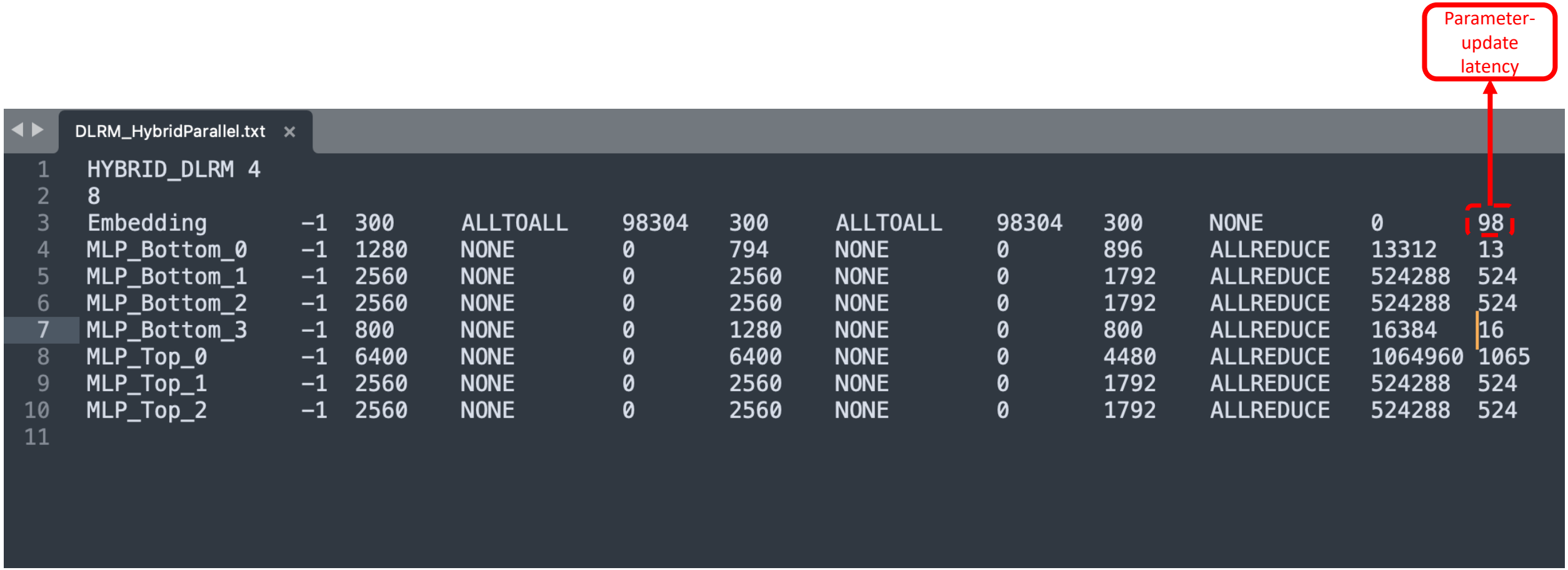
Text-based Workload Input

- A model is described layer-by-layer

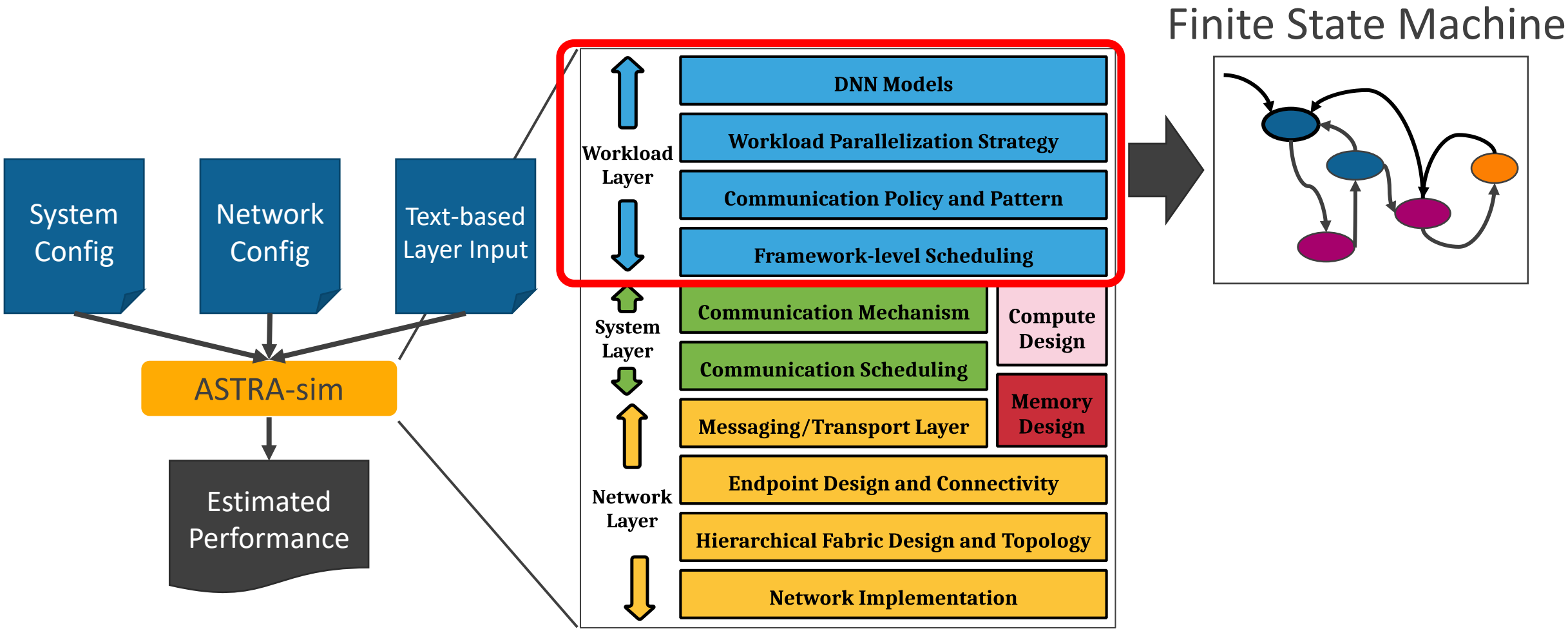


Text-based Workload Input

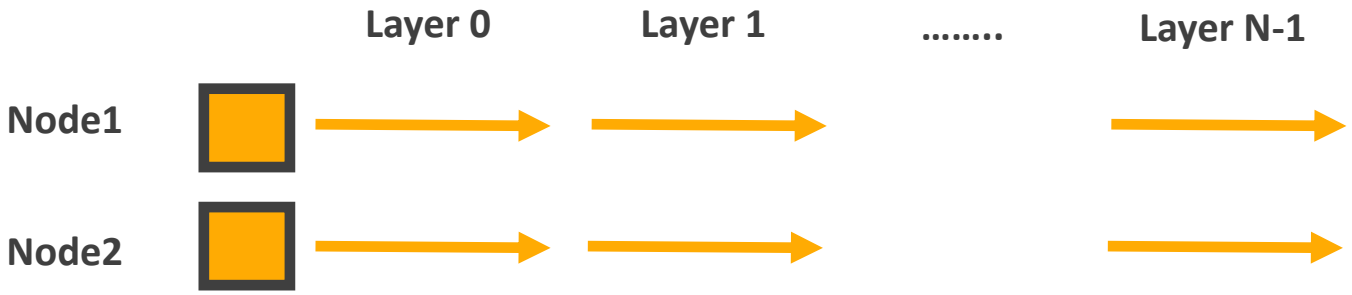
- A model is described layer-by-layer



Text-based Workload Layer



Vanilla Data-parallel Training Loop (FWD)



Flow-per-layer: 1.Compute output -> 2. go to the next layer

Inference compute

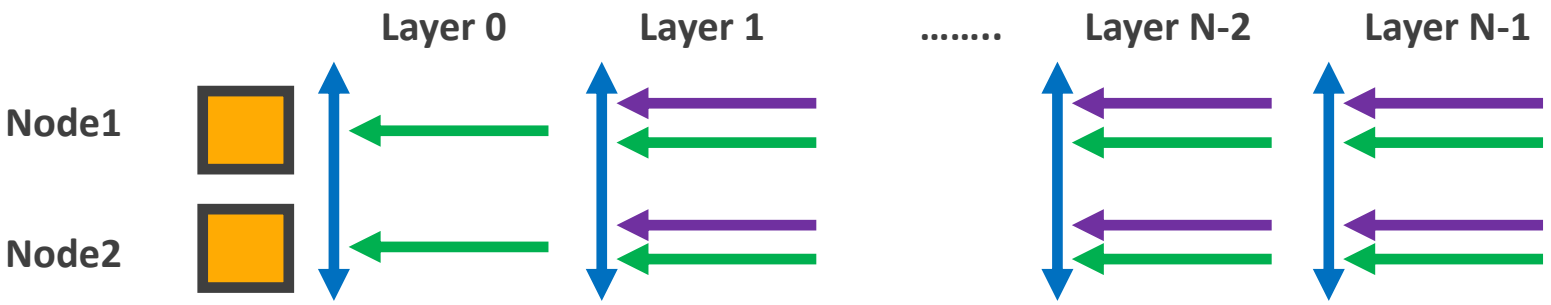
Input gradient compute

Weight gradient compute

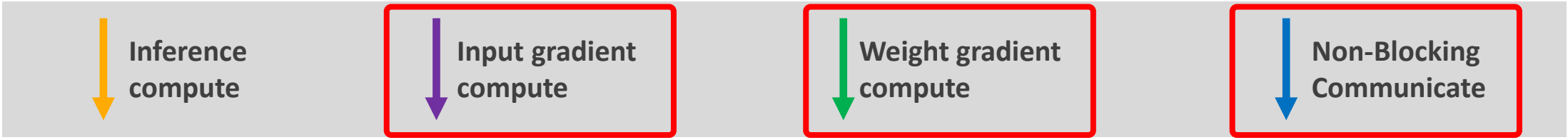
Non-Blocking Communicate

Vanilla Data-parallel Training Schedule

Vanilla Data-parallel Training Loop (BWD)



Flow-per-layer: 1.Compute weight gradient-> 2.issue weight gradient comm -> 3.compute input gradient -> 4. go to previous layer

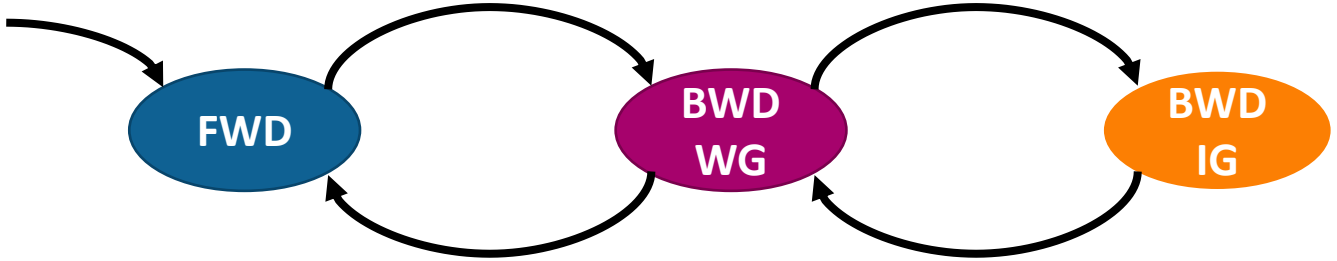


Vanilla Data-parallel Training Schedule

Vanilla Data-parallel Training Loop

Vanilla Data-parallel Training Schedule

FSM Diagram



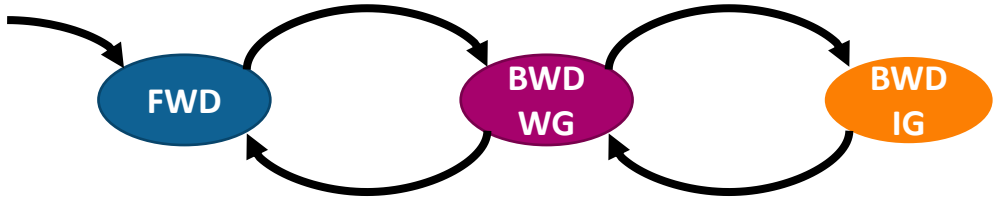
Vanilla Data-parallel Training Loop

```
void Workload::iterate_data_parallel() {
  assert(index >= 0);
  assert(index < SIZE);
  check_for_sim_end();
  if (current_state == LoopState::Forward_Pass) {
+-- 31 lines: if (!layers[index]->is_weight_grad_comm_finished_blo
    if (index >= SIZE) {
      current_state = LoopState::Weight_Gradient;
      index--;
    }
    generator->register_event(this, EventType::General, NULL, 1);
    return;
  } else if (current_state == LoopState::Weight_Gradient) {
+-- 14 lines: if (delay_loaded == false) {-----------------------
    if (index == 0) {
      pass_counter++;
      current_state = LoopState::Forward_Pass;
    } else {
      current_state = LoopState::Input_Gradient;
    }
    generator->register_event(this, EventType::General, NULL, 1);
    return;
  } else if (current_state == LoopState::Input_Gradient) {
+-- 11 lines: if (delay_loaded == false) {-----------------------
    delay_loaded = false;
    index--;
    current_state = LoopState::Weight_Gradient;
    generator->register_event(this, EventType::General, NULL, 1);
    return;
  }
}
```

- Training loop is implemented as a FSM
- `index` presents the current layer index
- `current_state` holds the current state

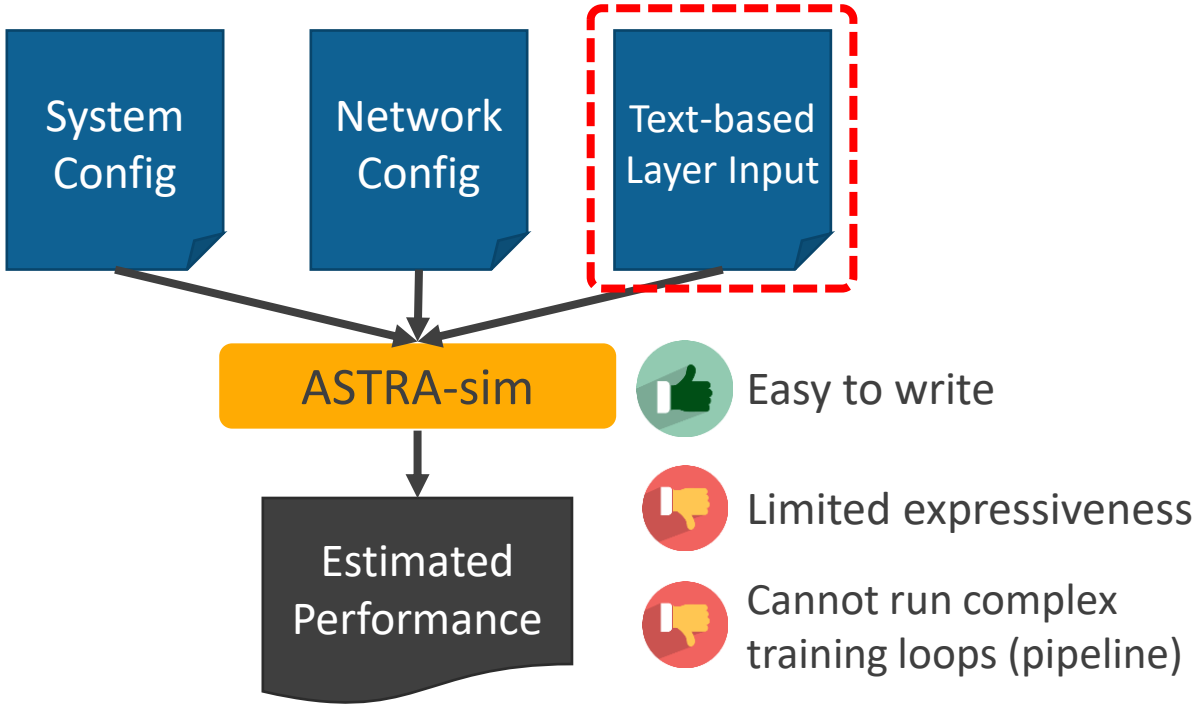
Vanilla Data-parallel Training Schedule

FSM Diagram



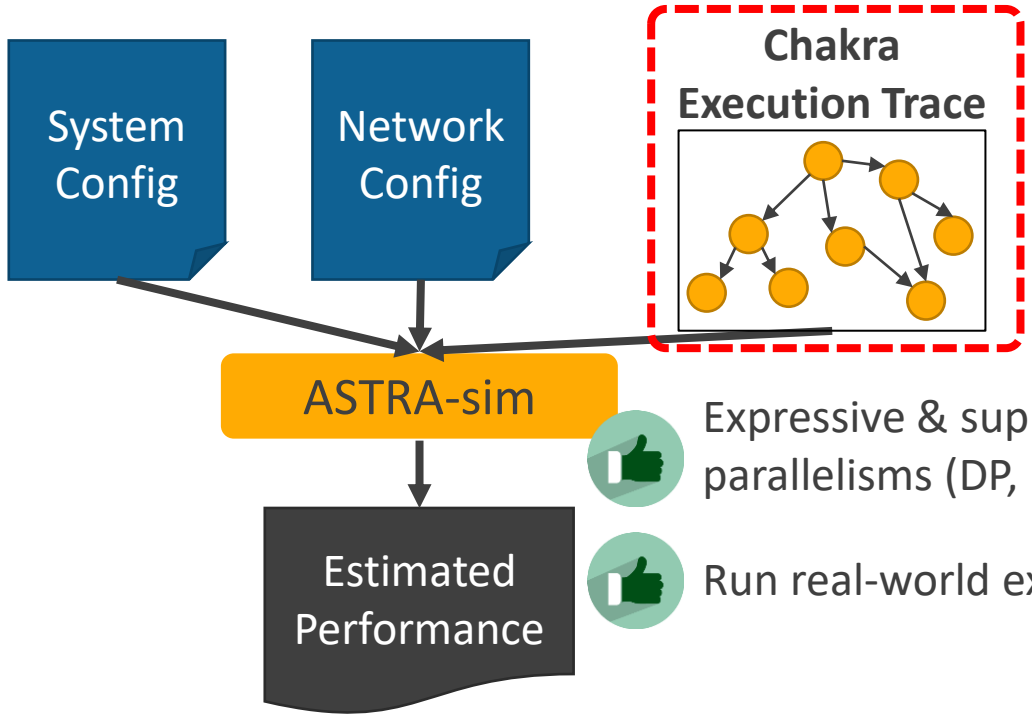
Graph-based Workload Layer

Text-based Workload Layer (ASTRA-sim 1.0, master branch)



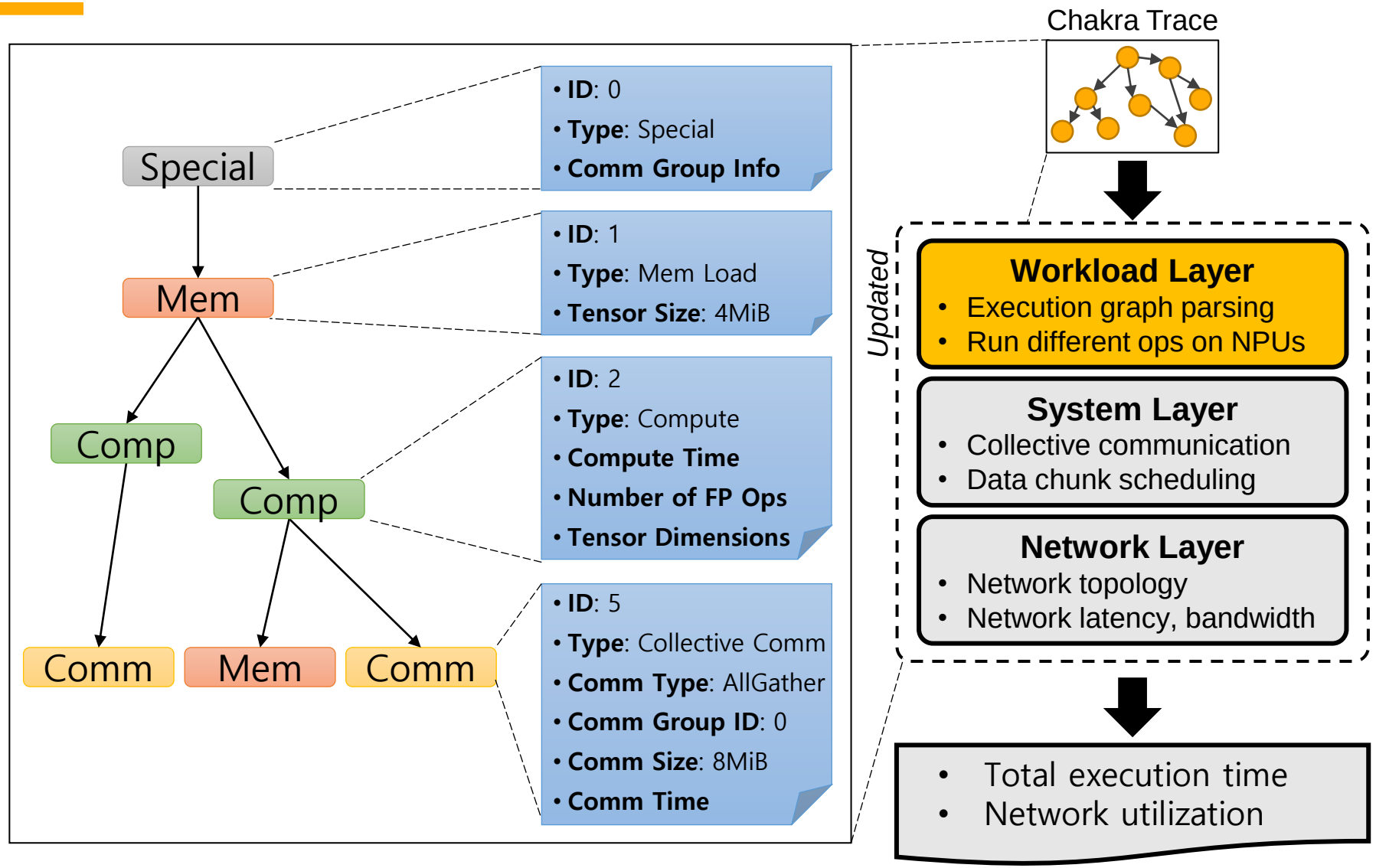
- Easy to write
- Limited expressiveness
- Cannot run complex training loops (pipeline)

Graph-based Workload Layer (ASTRA-sim 2.0, Chakra branch)



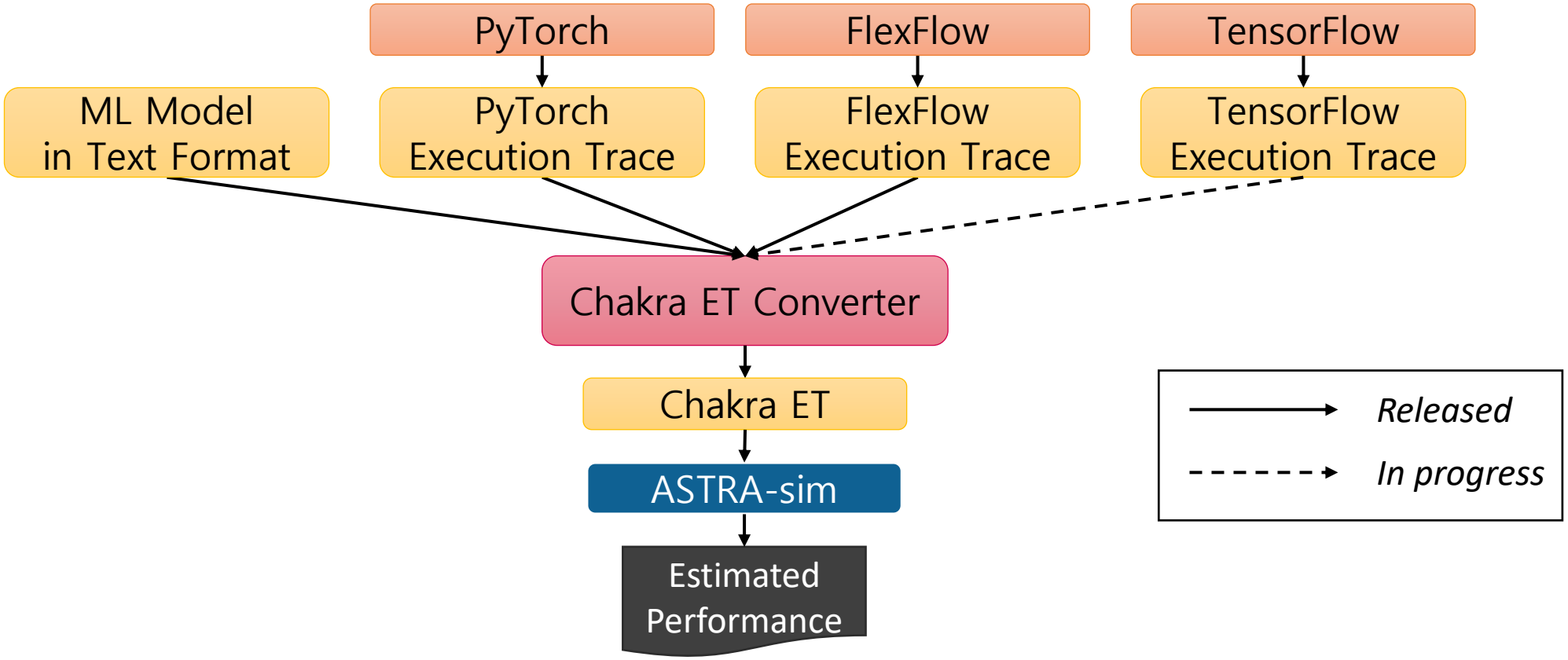
- Expressive & supports all parallelisms (DP, MP, PP)
- Run real-world exec traces

Chakra Execution Trace



Execution Trace Converter

- We provide a converter to support text input files and any other execution traces (ETs)



Running Synthetic Traces on ASTRA-sim

```
$ git clone --recurse-submodules git@github.com:astra-sim/astra-sim.git
$ cd astra-sim
$ git checkout Chakra
$ git submodule update --init --recursive
$ cd extern/graph_frontend/chakra/
$ git checkout main
$ cd -
$ ./build/astra_analytical/build.sh -c

$ cd extern/graph_frontend/chakra/eg_generator
$ cmake CMakeLists.txt && make -j$(nproc)
$ ./eg_generator

$ cd -
$ ./run.sh
```

Running Synthetic Traces on ASTRA-sim

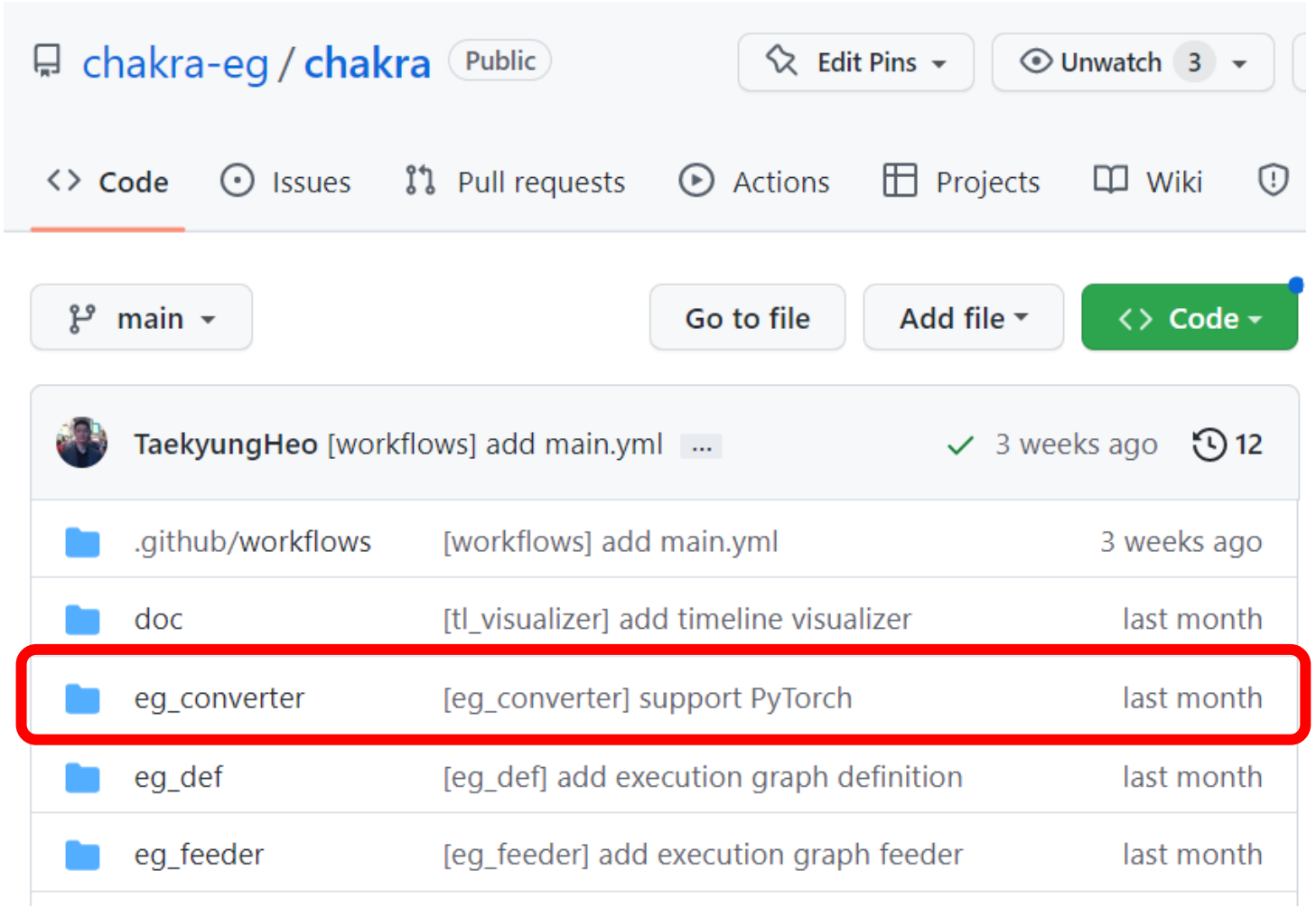
```
void twoCompNodesDependent(int num_npus, int num_dims) {
  DependencyGraph *dg;
  Node *node1, *node2;

  for (int npu_id = 0; npu_id < num_npus; ++npu_id) {
    dg = new DependencyGraph(getFilename(string(__func__), npu_id));

    node1 = dg->addNode(ChakraProtoMsg::COMP_NODE);
    node1->set_name("COMP_NODE");
    node1->set_simulated_run_time(5);

    node2 = dg->addNode(ChakraProtoMsg::COMP_NODE);
    node2->set_name("COMP_NODE");
    node2->set_simulated_run_time(5);

    dg->assignDep(node1, node2);

    dg->flushEG();
    delete dg;
  }
}
```

Running Synthetic Traces on ASTRA-sim

```
➜  astra-sim git:(Chakra) ./run.sh
[CostModel] Adding (inter-Node, Link) bandwidth: 1, radix: 1, count: 2016 (added cost: $8064)
ring of node 0, id: 0 dimension: local total nodes in ring: 64 index in ring: 0 offset: 1total nodes in ring: 64
ring of node 0, id: 0 dimension: local total nodes in ring: 64 index in ring: 0 offset: 1total nodes in ring: 64
ring of node 0, id: 0 dimension: local total nodes in ring: 64 index in ring: 0 offset: 1total nodes in ring: 64
ring of node 0, id: 0 dimension: local total nodes in ring: 64 index in ring: 0 offset: 1total nodes in ring: 64
sys[0] finished, 10 cycles
sys[1] finished, 10 cycles
sys[2] finished, 10 cycles
sys[3] finished, 10 cycles
sys[4] finished, 10 cycles
sys[5] finished, 10 cycles
sys[6] finished, 10 cycles
sys[7] finished, 10 cycles
sys[8] finished, 10 cycles
sys[9] finished, 10 cycles
```

Execution Trace Converter

https://github.com/chakra-eg/chakra



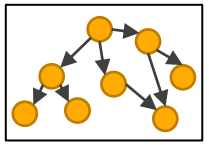
Converting Text Inputs to Chakra Traces

MLP_ModelParallel.txt

```
1 MODEL
2 6
3 layer_64_1_mlp0 -1  32291 ALLGATHER 37632 32291 ALLREDUCE 37632 12864 NONE  0 3229
4 layer_64_1_mlp1 -1  7488  ALLGATHER 65536 7488  ALLREDUCE 65536 3648  NONE  0 749
5 layer_64_1_mlp2 -1  7488  ALLGATHER 65536 7488  ALLREDUCE 65536 3456  NONE  0 749
6 layer_64_1_mlp3 -1  14144 ALLGATHER 147456  14144 ALLREDUCE 147456  10368 NONE  0 1414
7 layer_64_1_mlp4 -1  7488  ALLGATHER 65536 7488  ALLREDUCE 65536 3648  NONE  0 749
8 layer_64_2_mlp5 -1  9984  ALLGATHER 65536 9984  ALLREDUCE 65536 3456  NONE  0 998
```

Chakra ET Converter

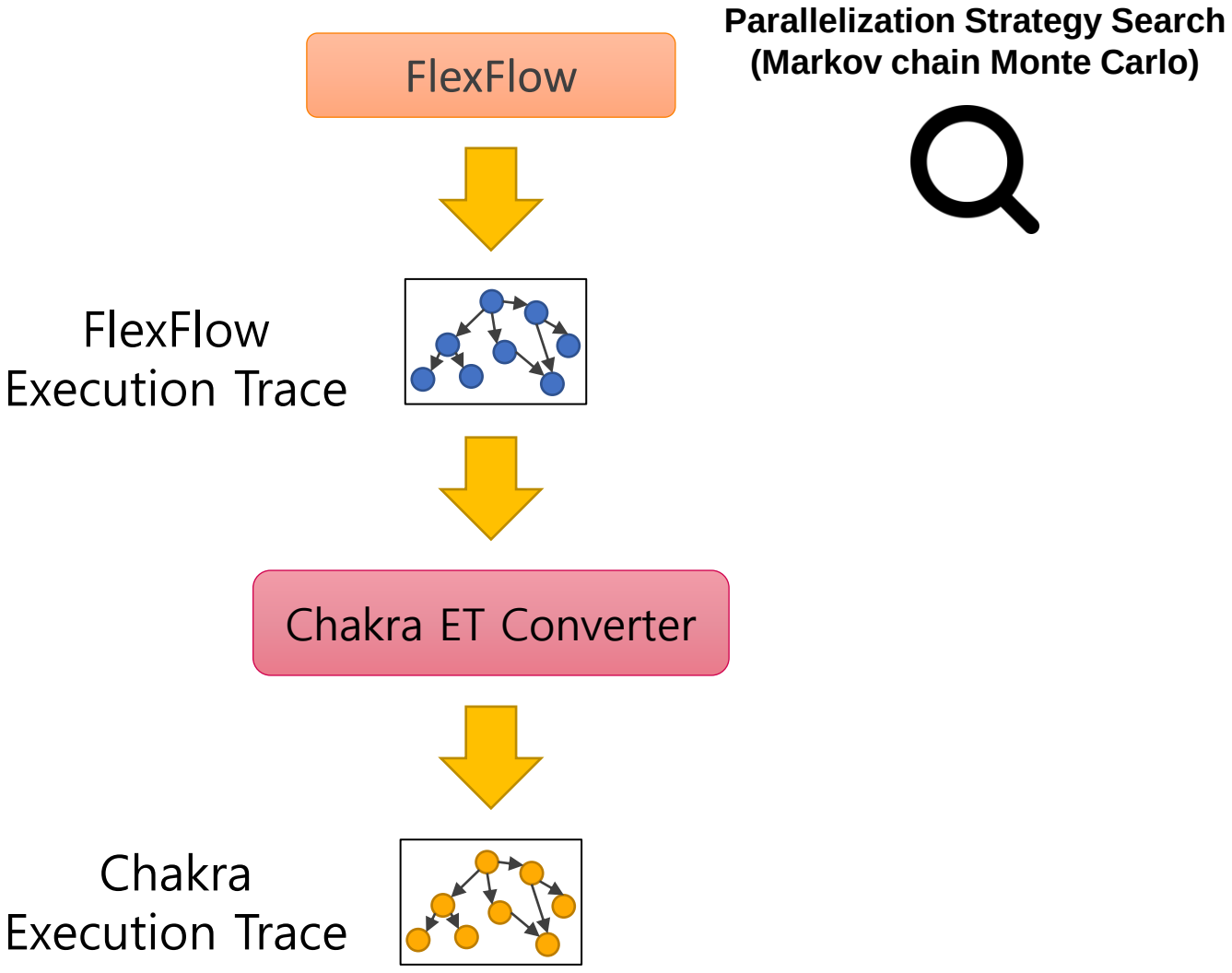
Chakra
Execution Trace



Converting Text Inputs to Chakra Traces

```
$ cd extern/graph_frontend/chakra/
$ git checkout tutorial
$ sudo python setup.py install

$ python -m eg_converter.eg_converter₩
    --input_type Text₩
    --input_filename eg_converter/input_files/Text/MLP_ModelParallel.txt₩
    --output_filename eg_converter/MLP_ModelParallel₩
    --num_npus 64₩
    --num_dims 1₩
    --num_passes 1

$ cd -
$ ./run.sh
```

Converting FlexFlow Traces to Chakra Traces



Converting FlexFlow Traces to Chakra Traces

```
$ python -m eg_converter.eg_converter₩
    --input_type FlexFlow₩
    --input_filename eg_converter/input_files/FlexFlow/alexnet.64.graph₩
    --output_filename eg_converter/AlexNet₩
    --num_npus 64₩
    --num_dims 1₩
    --npu_frequency 1000

$ cd -
$ ./run.sh
```