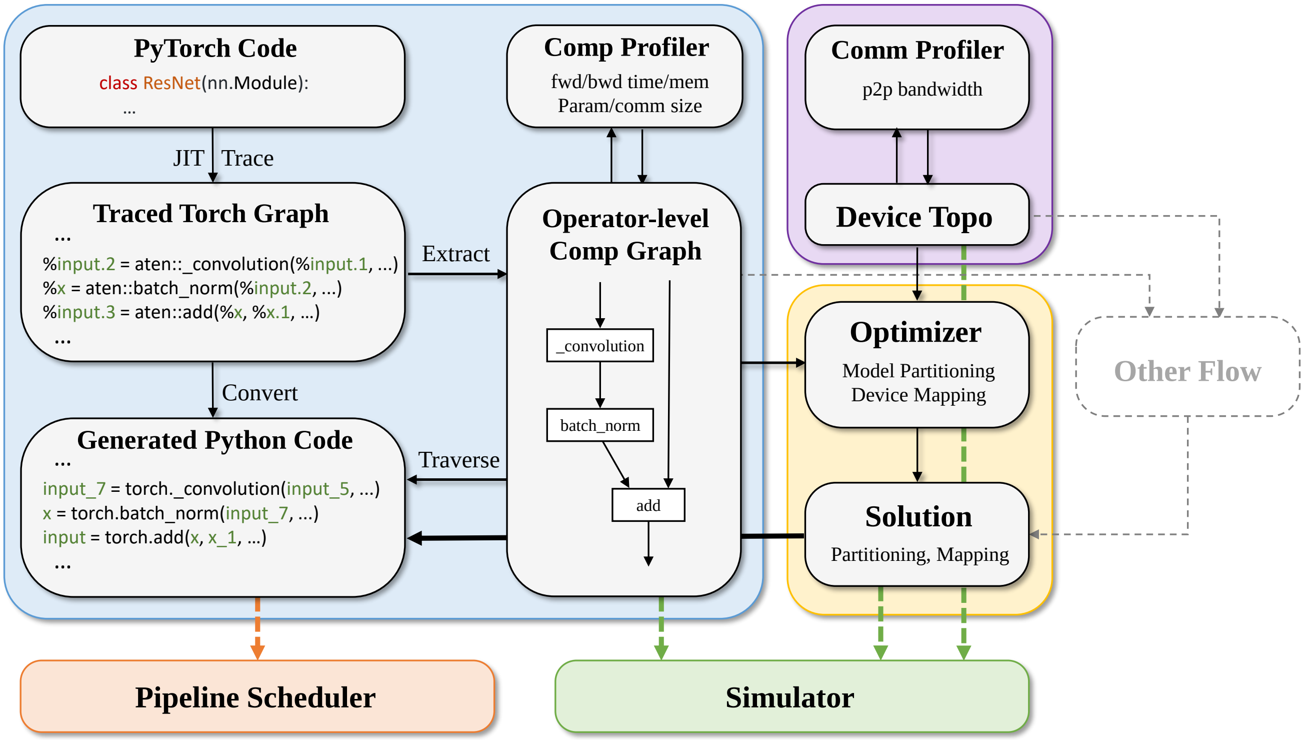

Fig. 11. The whole workflow of Parmesan.

Appendix A
System-level Implementation

We implement Parmesan with PyTorch [29], Numba [30], and NetworkX[5]. The core of Parmesan contains around 12k lines of Python. In this section, we will introduce other components in Parmesan. To simplify the notation, we will use the word *optimizer* to denote the whole optimization process (including partitioning and mapping).

Fig. 11 describes the whole workflow of Parmesan. Given a DNN model written in PyTorch [29], our graph extractor will first conduct just-in-time (JIT) tracing and automatically extract the *operator-level computation graph*. We will then launch the profiler to profile the operator attributes (including forward/backward time/memory) and compute the total parameter size of this operator (allreduce time is highly correlated to the parameter size).

As for the *device topology*, we first measure the point-to-point (*p2p*) communication overhead between every pair of devices. Then, the device topology representation, a p2p bandwidth look-up-table (LUT), is constructed based on the postal model [26], [27].

After building the operator-level computation graph and the device topology LUT, our optimizer will take them as input and output the partitioning and mapping results.

Parmesan's *pipeline scheduler* and *simulator* evaluate the quality of the output solution and provide valuable information for further development. Inspired by FlexFlow [11], we develop a task-graph-based simulator to tackle the general device topology simulation problem. For the real-world evaluation, we design a GPipe [13] fashion pipeline scheduler in PyTorch 1.10.1 [29] with CUDA 11.3, and adopt NCCL 2.10.3 [31] distributed backend for both the p2p communication between pipeline stages and allreduce between the stage replicas. Note that the Python code snippets executed by our pipeline scheduler are automatically generated from the operator-level graph and the optimized solution.

Besides, Parmesan supports writing/reading computation graphs, device topologies, and optimized solutions. Thus, one can further explore some more optimization algorithms/flows and evaluate their performance based on Parmesan. Meanwhile, Parmesan's optimizer and simulator are independent of the deep learning framework. Provided the computation graph extracted by other DL frameworks (like Tensorflow [32]), Parmesan can automatically conduct model partitioning and device mapping for the given network and simulate the solution performance.

[5]https://networkx.org/

TABLE X

RUNTIME (IN SECONDS) OF DEVICE MAPPING WITH DIFFERENT (S, R) AND DEVICE TOPOLOGIES. P2P AND ALLR REFER TO OUR TWO INSTANTIATIONS OF DEVICE MAPPING RESPECTIVELY.

Topology	Alg.	(2, 8)	(4, 4)	(8, 2)	(4, 16)	(8, 8)	(16, 4)	(4, 64)	(8, 32)	(16, 16)	(4, 128)	(8, 64)	(16, 32)
2d mesh	p2p	19.8	0.6	4.0	605.5	908.1	908.0	609.7	762.0	762.0	-	-	-
	allr	0.4	0.9	0.4	1.3	1.5	1.6	611.4	6.4	48.7	-	-	-
2d torus	p2p	0.2	0.6	3.8	454.3	605.8	605.8	609.8	761.5	761.9	-	-	-
	allr	10.1	0.4	0.5	1.6	1.4	1.8	480.3	6.9	6.9	-	-	-
3d mesh	p2p	-	-	-	605.4	756.7	756.7	-	-	-	616.8	769.9	770.8
	allr	-	-	-	42.5	1.6	1.6	-	-	-	661.1	88.0	57.6
3d torus	p2p	-	-	-	325.2	454.9	756.9	-	-	-	617.0	618.9	619.2
	allr	-	-	-	1.7	1.5	1.7	-	-	-	82.1	15.5	36.5
random_blk_1	p2p	50.5	12.1	234.4	1379.4	1357.2	490.7	345.9	607.8	608.1	354.1	1221.5	1223.5
	allr	869.4	12.0	9.7	905.4	1021.5	760.1	911.0	910.5	910.7	925.0	922.6	923.3
random_blk_2	p2p	1057.8	1209.3	1119.9	455.9	757.5	1208.4	1212.8	1063.8	1214.5	1221.8	1223.6	1224.6
	allr	906.7	304.3	3.0	908.0	1361.2	646.7	1402.3	1086.3	931.3	1842.8	1091.0	1096.0
uniform_dist	p2p	681.7	320.7	478.2	987.0	781.1	1096.9	949.0	1251.4	1402.7	1282.5	1410.3	1541.0
	allr	185.5	453.9	0.6	644.7	454.1	303.1	368.1	217.6	350.7	334.9	205.8	208.6

TABLE XI

COMPARISON ON RESULTS QUALITY OF DEVICE MAPPING WITH TIMEOUT HEURISTIC TO THAT OF THE OPTIMAL VERSION (I.E., WITHOUT THE TIMEOUT HEURISTIC). FIGURES IN THE TABLE ARE THE OBTAINED OBJECTIVE VALUES OF PROBLEM (5) FOR EACH COMBINATION OF ALGORITHM AND (S, R). DIFFERENCES ARE UNDERLINED. "*" DENOTES FAILURE IN OBTAINING RESULTS WITHIN 2 HOURS. "TMO" IS SHORT FOR TIMEOUT.

Topology	Alg.	(2, 8)	(4, 4)	(8, 2)	(4, 16)	(8, 8)	(16, 4)	(4, 64)	(8, 32)	(16, 16)	(4, 128)	(8, 64)	(16, 32)
2d mesh	p2p w/ tmo	177625	98940	60715	105340	66200	48829	105340	85851	114891	-	-	-
	p2p opt	177625	98940	60715	*	*	*	*	*	*	-	-	-
	allr w/ tmo	169473	81709	47752	82287	47789	28622	86123	47766	28927	-	-	-
	allr opt	169473	81709	47752	82287	47789	28622	*	*	28927	-	-	-
2d torus	p2p w/ tmo	177625	98940	60715	104140	62067	46091	105340	85851	79498	-	-	-
	p2p opt	177625	98940	60715	*	60715	39691	*	*	*	-	-	-
	allr w/ tmo	169473	81618	47752	82287	47763	28622	82455	47794	28927	-	-	-
	allr opt	169473	81618	47752	82287	47763	28622	82455	47794	28927	-	-	-
3d mesh	p2p w/ tmo	-	-	-	104140	64715	47691	-	-	-	105340	85851	65356
	p2p opt	-	-	-	*	60715	39691	-	-	-	*	*	*
	allr w/ tmo	-	-	-	82287	47763	28622	-	-	-	82483	47830	28978
	allr opt	-	-	-	82287	47763	28622	-	-	-	82483	47830	28978
3d torus	p2p w/ tmo	-	-	-	104140	63424	47691	-	-	-	105340	81195	46091
	p2p opt	-	-	-	*	60715	*	-	-	-	*	*	*
	allr w/ tmo	-	-	-	82287	47789	28622	-	-	-	82483	47795	28978
	allr opt	-	-	-	82287	47789	28622	-	-	-	82483	47795	28978
random_blk_1	p2p w/ tmo	7848025	5561027	5983883	7067273	5329127	5285295	10617386	10335205	11163700	10661165	11511750	14437823
	p2p opt	7848025	5561027	5983883	*	*	*	*	*	*	*	*	*
	allr w/ tmo	2717063	359296	96651	2757132	123523	44337	2891041	1933769	2238277	2913360	1964333	2312136
	allr opt	2717063	359296	96651	*	*	*	*	*	*	*	*	*
random_blk_2	p2p w/ tmo	1926463	2274920	1542303	2340990	1728740	1770727	3728927	3112257	7265727	3728927	3582268	7308867
	p2p opt	*	*	*	*	*	*	*	*	*	*	*	*
	allr w/ tmo	737258	127727	49239	604081	63963	42814	1196374	661812	462752	1797052	1090929	561108
	allr opt	*	127727	49239	*	*	*	*	*	*	*	*	*
uniform_dist	p2p w/ tmo	256915	267108	187708	254602	196161	201261	255566	314564	313764	253809	447725	484707
	p2p opt	*	266651	187708	*	*	*	*	*	*	*	*	*
	allr w/ tmo	184743	102645	49154	106767	56403	40923	107532	57963	44909	107609	58195	45580
	allr opt	184743	*	49154	*	*	*	*	*	*	*	*	*

APPENDIX B
IMPACT OF TIMEOUT HEURISTIC OF DEVICE MAPPING ON RUNTIME AND QUALITY

As mentioned in Section V, a timeout heuristic is applied to speedup the searching during device mapping. In this paragraph, we make inspections on its runtime and resulted optimization quality. We conduct experiments on mapping the partitioned SemanticFPN on non-regular topologies (including grid-based and randomly generated topologies) under different (S, R) pairs. Table X shows the runtime of two instantiations of our device mapping. Our mapping algorithm successfully generates results for up to 512 devices within 40 minutes. Table XI illustrates the result quality generated by mapping with timeout by comparing them with the optimal version (i.e., without the timeout heuristic). For the cases of the grid-based topologies with the number of devices $D = S \times R \leq 16$, the searching equipped with timeout is able to produce identical min-max stage time as by the optimal version. However, due to the increasing complexity, the optimal version fails to obtain the solution within 2 hours while our device mapping with timeout heuristic successes. "-" in Table X and Table XI denotes the number of devices D ($D = S \times R$) cannot form a specific torus/mesh architecture.