# Work In Progress: A New Task Model for Real-Time DNNs over GPU

Mourad Dridi, Yasmina Abdeddaim, Chiara Daini

# Work In Progress: A New Task Model for Real-Time DNNs over GPU

Mourad DRIDI*, Yasmina ABDEDDAIM*, Chiara DAINI†
*Univ Gustave Eiffel, CNRS, LIGM, F-77454 Marne-la-Vallée, France
†INRIA, Paris, France

***Abstract*—Recently, deep neural networks (DNNs) have been utilized in real-time systems such as autonomous vehicles, where meeting temporal constraints is essential. However, executing such systems on CPU-GPU architectures can make scheduling analysis challenging due to the added delays caused by computing and memory contention. In addition, classic task models are not directly able to model accurately such systems. In this article, we propose a new task model called DNN Task Model (DTM). This model considers both DNN properties and GPU architecture at the same time. It allows us to distinguish between CPU and GPU tasks, provides information about the DNN application and give more accurate execution time analysis through consideration of data quality. We compute DTM from a source CUDA file and a set of real-time specifications of the system. The proposed model is extensible enough to be adopted to various DNN type applications allowing designer to compare candidate software and GPU architectures. Furthermore, we propose a graph optimization inspired by Tensor-RT.**

## I. Introduction

As the use of Deep Neural Networks (DNNs) continue to raise in the field of real-time systems, they are becoming a popular choice for applications like autonomous driving. For the inference stage, DNNs are often executed on CPU-GPU architectures. This allows a balance of computational power and energy and a faster processing of the data [3] enabling the efficient execution of the DNNs. GPU systems provide high performance by offering greater computational power than most common CPUs. However, the use of GPUs also introduces contention, which can lead to additional variability in task execution time [6].

### A. Problem Statement

Alongside the need for computational efficiency, real-time DNNs are commonly characterized by real-time requirements, which need to be satisfied to guarantee the safe and correct behavior of the system. However, current DNN applications deployed over CPU-GPU systems have poor performance in terms of execution time variation. [6] found a non-negligible variations in execution time for DNN inference, which significantly challenges the worst case execution time estimation and scheduling problem. For example, in autonomous driving applications, the majority of DNN models show variations larger than 100ms, which affects autonomous driving safety [6]. There are several sources of uncertainty that contribute to the execution time variation issue in real-time DNN applications, including the input quality such as image resolution, communication latency and contention of concurrent tasks for resources such as memory, CPU, and GPU [6]. In addition, classic task models are not immediately suitable with CPU-GPU architectures and real-time DNNs. They are not able to model simultaneously and accurately both CPU tasks and GPU tasks, streams and communication delays.

### B. Contributions

In this article, we first propose a new task model that allows designers to perform an accurate schedulability analysis in order to assess the predictability of real-time DNN applications deployed over CPU-GPU architectures. We call the proposed model DNN Task Model (DTM). The DTM model describes both the details of the GPU architecture and the DNN application. It allows us to distinguish between CPU tasks and GPU tasks. The execution time for GPU tasks takes into account the targeted data quality and its impact on the execution time analysis. Additionally, the model specifies the type of operation for each GPU task, affording deeper understanding of the DNN network and enabling optimization of the task graph. From the source CUDA file and the real-time requirements of the system, the DTM task model is computed.

The second contribution of this article is to propose a graph optimization of the generated DTM. The optimization is inspired by Tensor-RT, which is a tool for optimizing DNN inference [7]. Graph optimization of Tensor-RT may not always align with real-time constraints. Our approach takes this into consideration and prioritizes the verification of real-time constraints compliance for every possible optimization.

The remainder of the paper is organized as follows. The next section introduces background elements about CPU-GPU architecture and Tensor-RT. Section III proposes our task model. Then, Section IV presents the graph optimization of DTM. An illustrative example is given in Section V. Section VI presents related works and Section VII concludes the article.

## II. Background

### A. GPU Execution Model and CUDA file

The considered GPU execution model for a given GPU kernel involves the following steps: first, a CPU task is launched, which initiates the execution of the kernel. Then, data is transferred from the CPU memory to the GPU memory. The GPU executes the kernel. Finally, the result of the computation is then transferred back from the GPU memory to the CPU.

GPU kernels are typically described in a file called a CUDA source file. This file includes different CPU and GPU kernels, which are functions that are executed both on the GPU and CPU [3]. It can also include data transfers between the CPU and GPU. Kernels are assigned into streams, which are sequences of kernels that are executed in order. Streams are used to synchronize the execution of different kernels and manage the data flow between them [4].

### B. DNN Inference and Tensor-RT

In the context of neural networks, we typically go through two phases: training and inference. In this work, we focus only on the inference phase wich is the process of using the trained model to make predictions on new data [7]. In order to make this process more efficient, it is important to optimize the DNN for inference. One tool that can be used for this purpose is Tensor-RT, which is a library developed by NVIDIA for optimizing DNNs for deployment on NVIDIA GPUs. It can be used to perform tasks such as layer fusion and precision calibration to improve DNN inference [7]. In this work, we consider an architecture consisting of multiple CPUs and a single GPU with multiple Streaming Multiprocessors (SMs). We consider also dependent periodic real-time tasks.

## III. DNN Task Model (DTM)

As shown in Fig 1(a), from the CUDA source file and real-time specifications of the system, we compute the DTM.

- The CUDA source file: outlines the software architecture of a program by specifying which functions will be executed on CPU and GPU. Additionally, it describes how data are transferred between the CPU and GPU.
- The system specification: provides information regarding the real-time constraints of the system and the quality of the data to be processed. These considerations must be taken into account during the modeling step in order to achieve an accurate scheduling analysis.

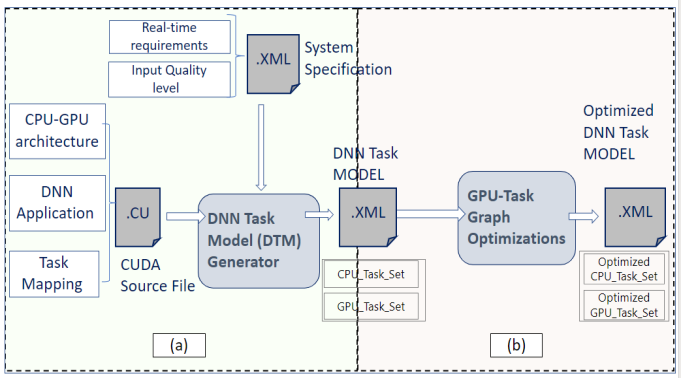


Fig. 1. General Approach: (a) DNN Task Model (b) DTM Graph Optimization

### A. Task Model

We model kernels specified in the CUDA source file as two sets of tasks $\Gamma_{CPU}$ and $\Gamma_{GPU}$. $\Gamma_{CPU}$ describes tasks executed on the CPU, while $\Gamma_{GPU}$ describes tasks executed on the GPU.

The task set $\Gamma_{CPU}$ is composed of $n$ periodic tasks $\Gamma_{CPU} = \{\tau_1, \tau_2, \ldots, \tau_n\}$. This task set is composed by two types of tasks, traditional tasks that are independent of the GPU, and tasks that are executed on the CPU to trigger the execution of a kernel on a GPU. Each task leads to an infinite sequence of jobs. Each task $\tau_i$ of $\Gamma_{CPU}$ is defined as follows: $\tau_i = \{ O_{\tau_i}, T_{\tau_i}, D_{\tau_i}, \Pi_{\tau_i}, C_{\tau_i}, Node_{\tau_i}, Next\}$ where:

- $O_{\tau_i}$ is the first release time of the task $\tau_i$.
- $T_{\tau_i}$ is the period of the task.
- $D_{\tau_i}$ is the implicit deadline of the task ($D_{\tau_i} = T_{\tau_i}$).
- $\Pi_{\tau_i}$ is the fixed priority of the task. 1 denotes the highest priority level while a larger value is a lower priority. The scheduling algorithm is not considered in this paper.
- $C_{\tau_i}$ specifies the computation time needed by the task defined as its Worst Case Execution Time (WCET). We consider a measurement-based WCET estimation.
- $Node_{\tau_i}$ identifies the CPU running the task. This parameter allows us to introduce the mapping configuration, i.e. to which CPU each task is assigned to.
- $Next$ is the precedence constraints function with:
$$Next: \{\Gamma_{CPU}, \Gamma_{GPU}\} \rightarrow \{\Gamma_{CPU}, \Gamma_{GPU}\}$$
$$\tau_i \rightarrow Next(\tau_i) = \{\tau_j \ldots \tau_k\}$$
For a given CPU task $\tau_i$, $Next(\tau_i)$ determines the set of tasks $\tau_j$ that have precedence constraints with the task $\tau_i$ where $\tau_j$ can be a GPU task or CPU task.

The task set $\Gamma_{GPU} = \{\phi_1, \phi_2, \ldots, \phi_m\}$ comprises $m$ periodic tasks. Each task leads to a sequence of infinite jobs. Each task $\phi_i$ of $\Gamma_{GPU}$ is defined as follows: $\phi_i = \{ O_{\phi_i}, T_{\phi_i}, D_{\phi_i}, \Pi_{\phi_i}, Next, C_{\phi_i}, Stream_{\phi_i}, Type_{\phi_i}, Data_{\phi_i}, Size_{\phi_i}\}$ where:

- Parameters $O_{\phi_i}, T_{\phi_i}, D_{\phi_i}, Next$ and $\Pi_{\phi_i}$ have the same definition as tasks of $\Gamma_{CPU}$.
- $C_{\phi_i}$ specifies the computation time needed by the task defined as its WCET. The value of this parameter is dependent on the quality of data being handled $C_{\phi_i}$(data quality). It is possible to set different levels of data quality, such as "low (L)", "medium (M)" and "high (H)", and thus assign appropriate values to this parameter, with the understanding that $C_{\phi_i}(\text{H}) > C_{\phi_i}(\text{M}) > C_{\phi_i}(\text{L})$.
- $Stream_{\phi_i}$ specifies to which stream the task $\phi_i$ is assigned. We can have multiple streams for a given system. Multiple GPU tasks can belong to the same stream.
- $Type_{\phi_i}$ specifies the type of operation that the task performs. In the context of DNN, there are several types of tasks, such as convolution, pooling and activation.
- $Data_{\phi_i}$ specifies data that is used by the task $\phi_i$.
- $Size_{\phi_i}$ specifies the number of sub tasks (blocks) of the task $\phi_i$. A GPU task can be composed of multiple sub-tasks. Each sub task models a thread block.

The $C_{\phi_i}$ parameter allows us to increase the accuracy of our analysis of execution time by considering multiple levels of quality. $Data_{\phi_i}$ and $Type_{\phi_i}$ provide more information about the considered task and allow to apply our graph optimization. $Stream_{\phi_i}$, $Next$, and $Size_{\phi_i}$ enable a more accurate scheduling analysis. The DNN processing component is not inherently periodic, but it can be used as part of a larger

periodic system that repeats the same steps (e.g. acquiring data, processing data using the DNN) at regular intervals. For that, we assume in this work that tasks belonging to the same DNN network have the same periods and deadlines.

### B. Guidelines for Generating DTM

Each function in the CUDA code is modeled as a task. By analyzing keywords and interpreting different syntax components of the code, we can identify the various characteristics and parameters of each task. For example, to identify CPU and GPU tasks, specific keywords such as "global" and "device" can be used, which indicate that the task is executed on the GPU and CPU respectively. Here are some of the most commonly used keywords in CUDA.

- **global or host** indicate that a function is is executed on either the GPU or CPU. It can be modeled as a GPU task or a CPU task, respectively.
- **cudaMemcpy** is used to transfer data between the host (CPU) and device (GPU) or between different memory spaces on the device. It can be considered in the definition of dependencies between tasks in DTM.
- **cudaStreamCreate** and **cudaMemcpyAsync** manage a stream. It can be considered in the definition of dependencies between GPU tasks.

There are many other keywords and syntax components available for specific tasks and functionality. In the final version of this work, more detailed information will be provided on all used keywords and how they are modeled in DTM.

## IV. Optimized DTM : Graph Optimization

Figure 1(b) shows the optimization we propose in this article. Using the generated model, we apply graph optimization rules in order to produce an optimized model in terms of computation time while considering real-time constraints of the system. We apply transformations to the set of GPU tasks with the aim of reducing the data transfer between the CPU and GPU. This leads to a reduction in communication delays. These optimizations can reduce memory usage, increase computation throughput, and improve overall performance.

Our optimization rules are inspired by Tensor-RT which involve merging multiple tasks into a single task when they belong to the same neural network, use the same data, and have constraints of precedence. Furthermore, tasks are fused based on operation type, such as merging Convolution task and ReLU Activation task into one. However, not all Tensor-RT optimizations are suitable for real-time systems, unlike our approach, which is specifically designed to meet these constraints. In the final version of this work, more detailed rules of graph optimization applied to DTM will be provided.

## V. Example of a real-time DNN over CPU-GPU architecture

In this section, we give an example of CUDA code for one DNN inference application and one CPU task. The DNN application is composed of 4 GPU kernels with precedence constraints, using the same stream. Kernel 1 named DNN_Kernel_1 uses data1 and data2. kernel 2 named DNN_kernel_2 uses data1. kernel 3 named DNN_kernel_3 uses data1, and kernel 4 named DNN_kernel_4 uses data1 and data2.

```
#include <cuda_runtime.h>
__global__ void DNN_Kernel_1(int *data1,
       int *data2) {
   // GPU code here
  *data1 = *data1;
  *data2 = *data2;}
__global__ void DNN_Kernel_2(int *data1) {
  // GPU code here
  *data1 = *data1;}
__global__ void DNN_Kernel_3(int *data1) {
  // GPU code here
  *data1 = *data1;}
__global__ void DNN_Kernel_4(int *data1,
  int *data2) {    // GPU code here
  *data1 = *data1;
  *data2 = *data2;}
int main() {
  int *data1, *data2, *data3;
  cudaMalloc(&data1, sizeof(int) * 100);
  cudaMalloc(&data2, sizeof(int) * 100);
  cudaMalloc(&data3, sizeof(int) * 100);
  cudaStream_t stream;
  cudaStreamCreate(&stream);
  dim3 block(10,1,1);
  dim3 grid(3,1,1);
  DNN_Kernel_1<<<grid, block, 0, stream>>>(data1,
      data2);
  DNN_Kernel_2<<<grid, block, 0, stream>>>(data1);
  DNN_Kernel_3<<<grid, block, 0, stream>>>(data1);
  DNN_Kernel_4<<<grid, block, 0, stream>>>(data1,
      data2);
  cudaStreamSynchronize(stream);
  // CPU code here
  kernel5(data3);
  cudaStreamDestroy(stream);
  cudaFree(data1);
  cudaFree(data2);
  cudaFree(data3);
  return 0;}
void kernel5(int *data3) { // CPU code here
  *data3 = *data3;}
```

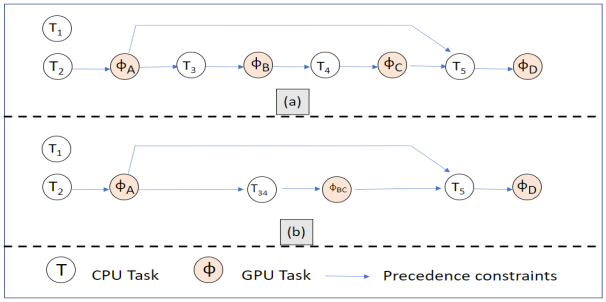


Fig. 2. (a) Task graph of DTM – (b) Task graph of the optimizated DTM

By applying DTM to this code example, we will generate the following task model. For DNN inference application, we will have 8 dependent tasks, 4 GPU tasks ($\phi_A$, $\phi_B$, $\phi_C$, $\phi_D$) and 4 CPU tasks ($\tau_2$, $\tau_3$, $\tau_4$, $\tau_5$). For the second application, we will have one independent CPU task ($\tau_1$). In Fig 2 (a) and in Table I, we give more details about the generated DNN task model for this example. Task parameters are given as an illustration to showcase our approach. This example is based

on a DNN (4 layers: Normalization, Convolution, ReLU, Fully Connected (FC)) which can be used for image classification.

| $\Gamma_{CPU}$ | $\tau_1$ | $\tau_2$ | $\tau_3$ | $\tau_4$ | $\tau_5$ |
|---|---|---|---|---|---|
| (O;T=D) | (0;20) | (0;100) | (0;100) | (0;100) | (0;100) |
| $\Pi$ | 2 | 1 | 1 | 1 | 1 |
| C | 4 | 5 | 5 | 5 | 5 |
| Node | CPU1 | CPU1 | CPU1 | CPU1 | CPU1 |
| Next | | $\tau_A$ | $\tau_B$ | $\tau_C$ | $\tau_D$ |

| $\Gamma_{GPU}$ | $\phi_A$ | $\phi_B$ | $\phi_C$ | $\phi_D$ |
|---|---|---|---|---|
| (O;T=D ;$\Pi$ ) | (0;100;1) | (0;100;1) | (0;100;1) | (0;100;1) |
| C(L, M, H) | (6;8;10) | (6;8;10) | (6;8;10) | (6;8;10) |
| Node | GPU1 | GPU1 | GPU1 | GPU1 |
| Next | $\tau_3$ | $\tau_4$ | $\tau_5$ | |
| Stream | stream | stream | stream | stream |
| Data | 1,2 | 1 | 1 | 1.2 |
| Type | Normal | Conv | ReLU | FC |
| Size | 3 | 3 | 3 | 3 |

TABLE I
TASK PARAMETERS

Now we apply graph optimization on the generated DNN task model. GPU tasks $\phi_B$ and $\phi_C$ utilize the same data (data1) and are part of the same DNN application. As such, we suggest merging these two GPU tasks into a unified task $\phi_{BC}$.

As a result of merging the two previous GPU tasks, there will be one attached CPU task ($\tau_{34}$) instead of two separate tasks ($\tau_3$ and $\tau_4$). Fig 2 (b) illustrates the optimized graph.

## VI. Related Works

There are many previous research works on the execution of real-time DNNs inference over CPU-GPU at different levels.

**Task model:** In [10] the authors propose a periodic graph-based task model and a method to reduce the graph response time bounds by merging graph nodes. [5] presents the HPC-DAG Task Model, a workload-based task model, which allows specifying real-time tasks deployed over heterogeneous embedded platforms. [1] defines Multi-segment suspension based task model. Suspension is used for the offload mechanism such as memory copy and migration on the GPU. These models represent significant efforts to support real-time systems on GPU. However,

**Scheduling framework:** [3], [2], [4] present scheduling frameworks that have been proposed to schedule DNNs over a CPU-GPU architecture. However, models used in these frameworks are not optimally adapted for DNN applications deployed over CPU-GPU architectures.

**DNN modifications:** [4], [9], [10] propose techniques for the modification of a neural network that can improve its usage and throughput. For example, using lower-resolution images in object detection context.

DTM is distinct from most previous models. The main difference lies in our approach, which aims to accurately model both DNN and CPU-GPU properties simultaneously. DTM introduces new parameters such as data, type, size, and stream, which improve the accuracy of modeling DNN applications over CPU-GPU. These parameters enable more precise estimation of execution time while considering the quality of the processed data and allow to apply graph optimizations. Furthermore, our proposed model is automatically generated from a NVIDIA CUDA file. Finally, DTM comes with an adapted graph optimization inspired by Tensor-RT.

## VII. Conclusion

In this article, we introduce DTM, a novel task model designed to model real-time DNN deployed over a CPU-GPU architecture. DTM takes into account all relevant factors that can affect the execution time of tasks, allowing for more accurate assessments of task schedulability. Additionally, we propose a graph optimization technique to speed up the inference of DNN while satisfying real-time system constraints.

In the future, we plan to provide a detailed explanation of DTM's rules and the specifics of the graph optimizations implemented. Additionally, we aim to integrate DTM and its optimizer into the Cheddar real-time system simulation tool.

We will implement DTM into Cheddar which is a GPL framework that provides a scheduling simulator, schedulability tests and various features related to the design and the scheduling analysis of real-time systems [8]. CheddarADL allows the users to describe both the software and the hardware parts of the system they expect to analyze [8]. We will extend CheddarADL to implement DTM.

## References


[1] Daniel Casini, Paolo Pazzaglia, Alessandro Biondi, Marco Di Natale, Optimized partitioning and priority assignment of real-time applications on heterogeneous platforms with hardware acceleration, Journal of Systems Architecture, Volume 124, 2022, 102416, ISSN 1383-7621,

[2] W. Kang, K. Lee, J. Lee, I. Shin and H. S. Chwa, "LaLaRAND: Flexible Layer-by-Layer CPU/GPU Scheduling for Real-Time DNN Tasks," 2021 IEEE Real-Time Systems Symposium (RTSS), Dortmund, DE, 2021, pp. 329-341, doi: 10.1109/RTSS52674.2021.00038.

[3] Y. Xiang and H. Kim, "Pipelined Data-Parallel CPU/GPU Scheduling for Multi-DNN Real-Time Inference," 2019 IEEE Real-Time Systems Symposium (RTSS), Hong Kong, China, 2019, pp. 392-405, doi: 10.1109/RTSS46320.2019.00042.

[4] H. Zhou, S. Bateni and C. Liu, "S3DNN: Supervised Streaming and Scheduling for GPU-Accelerated Real-Time DNN Workloads," 2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Porto, Portugal, 2018, pp. 190-201, doi: 10.1109/RTAS.2018.00028.

[5] Z. Houssam-Eddine, N. Capodieci, R. Cavicchioli, G. Lipari and M. Bertogna, "The HPC-DAG Task Model for Heterogeneous Real-Time Systems," in IEEE Transactions on Computers, vol. 70, no. 10, pp. 1747-1761, 1 Oct. 2021, doi: 10.1109/TC.2020.3023169.

[6] Liu, Liangkai and Wang, Yanzhi and Shi, Weisong, "Understanding Time Variations of DNN Inference in Autonomous Driving", 2022 arXiv, doi: 10.48550/ARXIV.2209.05487

[7] Eunjin Jeong, Jangryul Kim, and Soonhoi Ha. 2022. TensorRT-Based Framework and Optimization Methodology for Deep Learning Inference on Jetson Boards. ACM Trans. Embed. Comput. Syst. 21, 5, Article 51 (September 2022), 26 pages. https://doi.org/10.1145/3508391

[8] Mourad Dridi, Frank Singhoff, Stéphane Rubini, Jean-Philippe Diguet, ECTM: A network-on-chip communication model to combine task and message schedulability analysis, Journal of Systems Architecture, Volume 114, 2021, 101931, ISSN 1383-7621.

[9] M. Yang, S. Wang, J. Bakita, T. Vu, F. D. Smith, J. Anderson, and J.-M. Frahm, "Re-thinking CNN frameworks for time-sensitive autonomousdriving applications: Addressing an industrial challenge," in Proceedings of the 25th Real-Time and Embedded Technology and Applications Symposium, 2019, pp. 305–317

[10] Ai meets real-time: Addressing real world complexities in graph response-time analysis, Sergey Voronov, Stephen Tang, Tanya Amert, and James H. Anderson, RTSS 2021