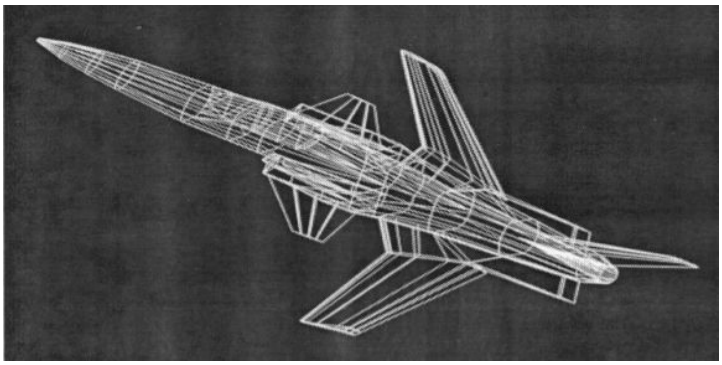
1.0 - What is a GPU?

The GPU (graphics processing unit) is a computer component designed around the concept of parallel processing to help computers process graphics. It is often contrasted with the CPU in that it has many cores, higher throughput, and is structurally optimized for parallel processing running numerous operations at the same time e time [1]. What makes the GPU unique as a component of an embedded system is its powerful parallel processing and image handling capabilities. Computer systems contain a CPU and at least one GPU. The two components function quite differently, due to the different natures of the two devices. The CPU has few cores and can handle just a few software threads at a time. In contrast, the GPU is composed of several hundred cores, making it suited for processing large amounts of information at a time

1.1 - The History of GPUs



3D graphics processing began almost three decades before the first true GPU, with the now-infamous wireframe models of the ‘70s and ‘80s. This early on, pixels weren’t even used for 3-D models; the line drawings created were more similar to the creations of an Etch-a-sketch. Polygons and planes weren’t used either; just lines that formed the outlines of the image being displayed.

In 1987, the first models created with polygons were introduced, and in 1992, texture mapping made those models even more realistic. As time went on and 3D graphics became more complex, changes had to be made to the CPU in order to make rendering them feasible. This eventually led to the release of Nvidia’s GeForce 256, the first processor advertised as a Graphics Processing Unit.

With 3-D processing becoming more and more popular, GPUs continued to improve. As the 21st century was entering its infancy, the first programmable GPUs were released, allowing users to map data and operations to better suit their purposes. At about that same time, the first General Purpose Graphics Processing Units (GPGPUs) were becoming available, allowing for a wider range of applications for the technology. As the years continued on, both the hardware and software were refined. Between 2002 and 2003, Nvidia was able to nearly double the number of transistors available on their GPUs, while increasing the number of operations per second by over 6 times. [2]

In 2007, the Compute Unified Device Architecture (CUDA) platform was born, allowing for users to develop for GPUs in more popular languages, such as C and C++, making development for GPUs more accessible. CUDA support has since become a regular feature on Nvidia's processors.[3]

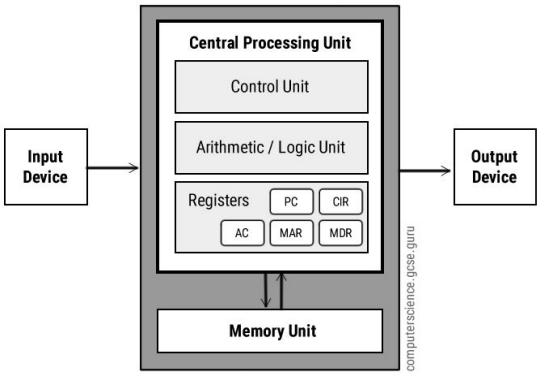
2.0 - GPU Theory

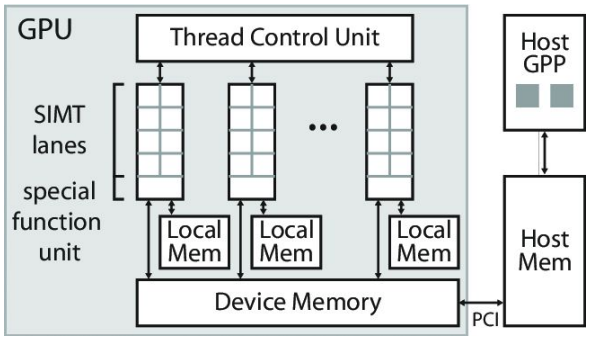
2.1 - GPU Architecture

As can be seen in the images on the right, GPUs and CPUs have different architectures to fit their differing roles. Generally, GPUs have only one I/O port, referred to as the PCI, while the CPU has separate I/O ports. The CPU also outputs from the CPU itself, while the GPU's PCI port is linked directly to the memory registers. As previously stated, GPUs also have more cores than CPUs, to allow for more tasks to be completed at once. A GPU also relies on a host to handle all I/O, as well as to provide its instructions, while a CPU takes in the instructions directly, and will likely handle the GPU through the host.

2.2 - Types of GPUs

Single GPU is the basic configuration GPU for standard users like gamers, video editors, home office users as it has HD resolutions with playable frame rates. Single GPU is a good value and has plenty of power needed for standard users. It has more efficient cooler operations and is quieter than multi-GPU configurations. Multi-GPU configuration is more specific to getting the best out of a specific task. It provides better frame rates and it is a higher powered card compared to single GPU. The benefit of using a multi-GPU is that it " Increased graphic performance and benchmarks especially at high resolutions or 4K. Preferable for multi-monitor gaming.". Multi GPU requires more power and it being more expensive doesn't necessarily mean that the performance will match the cost. Compared with single GPU, multi-GPU has a lot more noise and produces more heat; it's frame rates is a lot faster to identify with the eyes





2.3 - The CPU/GPU Relationship

CPUs and GPUs differ in the way that they are most efficient. A CPU can only focus on a few tasks at once, but gets through those tasks relatively quickly, making it better for running basic systems and operations. On the other hand, a GPU can take a more complex problem, split it up into smaller parts, and can complete hundreds of operations at once. While each individual task takes longer to complete, in general a GPU can complete large numbers of tasks faster than a CPU [4]. However, a CPU is required to run the system, and there are some operations that the GPU simply can't perform, causing it to rely on the CPU. Both processors have their own boons and banes, leading to many people putting both together into a heterogeneous system.

A GPU is best used over a CPU with the inclusion of many small calculations done at once, as they will compute these calculations much faster than a CPU would. They're also well-used when trying to repetitively complete the same task, as well as when you need to free up the CPU for other tasks.

At their most effective, a GPU should take instructions from the CPU and complete as many actions as possible without having to refer back to the CPU, as communication between the two boards leads to a slowdown.

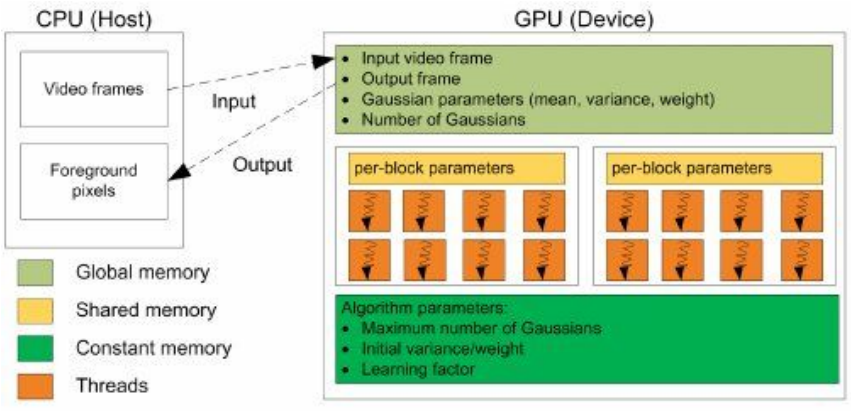
2.4 - When to Use a GPU

When considering what processes can be delegated to the GPU, it is important to consider the GPU's fundamental characteristic: parallelism. An important implication of parallelism is the ability to speed up any processes that can be broken into simultaneously computable parts. This is clearly seen in the prevalent usage of GPUs in machine learning and image-processing. It has several important implications for image processing. GPUs can process images faster, with minimal to no loss of quality [5]. Common image processors, such as OpenCV generally treat images as a matrix of pixels, one-at-time. GPUs allow this process to be broken into smaller, simultaneous procedures which can be executed in parallel on a number of different threads [6].

2.5 - GPU Output

GPU outputs can look unintelligible to the untrained eye; a mass of words and percentages with no clear meaning. However, it is still very possible for these numbers to be understood by a human being.

The most noteworthy program for this task is the



Nvidia Management Library (NVML)[7], a tool that monitors the state of Nvidia processors. It is installed alongside the GPU Deployment Kit. Possible outputs can include both software information, like error counts and active processes, as well as more hardware-focused information, such as internal temperature and power management. It also allows the developer to edit the mode that the processor is running in.

NVML also allows for the use of other applications alongside it, including nvidia-smi, which takes on a lot of the query duties basic to NVML; the Ganglia plugin, which displays information in graphs rather than tables; and various third party tools. While a more recent development, NVML is built so as to be backwards compatible with most previous Nvidia processors. However, if you are unable to use NVML for any reason, a number of those third party tools, including nvidia-smi, are available separate from NVML.

3.0 - GPU Implementation

The GPU's architecture is composed of several multiprocessors. Each multiprocessor has several scalar processors sharing a single instruction unit. "The processors within a multiprocessor execute in lock-step, all the same instruction each cycle, but on different data. Each multiprocessor can maintain hundreds of threads in execution. These threads are organized in sets, called warps, the size of which is equal to the number of scalar processors that are in a multiprocessor. Every cycle, the hardware scheduler of each multiprocessor chooses the next warp to execute". Different companies has its own types of schematic when it comes to the designs of GPU

At Nvidia, the GPU design is typically made of four categories which is the control circuitry, processing engines, display engines, and misc engines. The control circuitry includes a PMC (master control area), PBUS (bus control and an area where registers operates, PPMI (PCI Memory Interface that handles SYSRAM accesses from other units of the GPU), PTIMER (measures wall time and delivers alarm interrupts), PCLOCK+PCONTROL (clock generation and distribution), PFB (memory controller and arbiter, PROM (VBIOS ROM access), and PSTRAPS (configuration strap access). The processing engines includes PFIFO (gathers processing commands from the command buffers prepared by the host and delivers them to PGRAPH and PVPE engines in an orderly manner), PGRAPH (memory copying 2d and 3d rendering engine), PVPE (a trio of video decoding/encoding engines), and PCOUNTER (performance monitoring counters for the processing engines memory controller). The display engines consist of the PCRTC (generates display control signals and read framebuffer data for display), PVIDEO (reads and preprocesses overlay video data), PRAMDAC (multiplexes PCRTC, PVIDEO and cursor image data, applies palette LUT, converts to output signals) and PTV (an on-chip TV encoder). The misc engines has the PMEDIA (controls video capture input and mediaport, acts as a DMA controller

4.1.0 - GPU Software

Nvidia provides a number of Software Development Kits (SDKs) that simplify any new engineer's interactions with GPUs. It is leaps and bounds above its competitors AMD and INtel in providing resources to hobbyist or individual engineers. (Nevertheless, all three companies actively engage in collaborations with universities and other corporations). SDKs are like coding libraries but designed to specifically suit the capabilities of an Nvidia system.

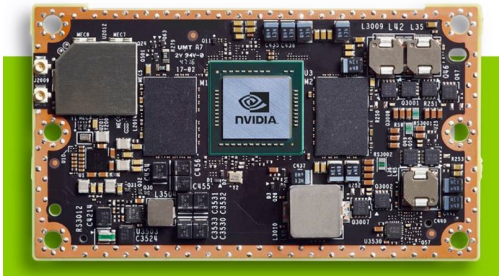


Figure 1: A Jetson TX2
Source: [8]

4.1.1 - Nvidia SDK Manager (Jetpack SDK)

The JetPack SDK is probably the best place to start when approaching a new Nvidia system. It allows new users to set up their embedded device with an operating system and proprietary software [8]. NVIDIA Jetpack is an SDK package equipped with a number of tools to assist in the building of AI applications, providing support to the AGX Xavier, TX2, TX1, and Jetson Nano. It is also referred to as the Nvidia SDK Manager.

The Nvidia SDK Manager requires two hardware components. First, the user must have a host computer running Ubuntu 16.04 or 18.04, a working Internet connection, and a Jetson device [9]. The host device must have a minimum of 40GB of free disk space whereas the Jetson must have at least 25GB of free disk space. Once a USB 2.0 connection has been established between the two devices, the Jetson can be easily set up with the host computer.

The SDK has a GUI which guides the user on setting up the Jetson as well as any additional packages that need to be installed [10]. This is the only way to put an operating system onto the Jetson. The SDK manager can be used later to update the SDK installation on the system [11].

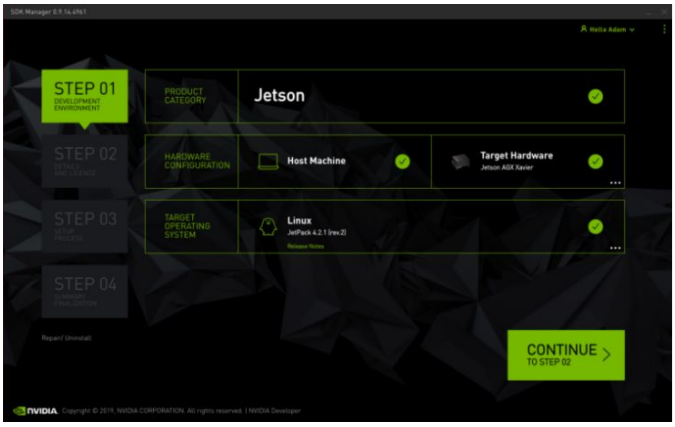


Figure 2: Screenshot of the JetPack Setup Assistant
Source: [12]

The first and most important step is shown above.The targeted device should be specified. In the above image, this device is the Jetson. If the program does not detect the correct model, it will be necessary to set the correct hardware version. The Target Operating System specifies the OS to be installed. The OS will always be Linux, but always double-check the documentation to ensure that the correct version of LInux is being installed (this is a decision made by the engineer). It is important to be clear about the parameters on this page. It is not possible to install the software incorrectly, but it is possible to install a different version than what is desired for the project.

Next, we will explore some other SDKs that Nvidia provides.

4.1.2 Deepstream SDK

The DeepStreak SDK is a computer-vision library that includes AI-based video processing with sensor processing. Deepstream relies on the Gstreamer paradigm [13].

DeepStream is free of charge and downloadable from the Nvidia website. Be sure to consult the Nvidia website for the various compatibility requirements of each version of DeepStream [14].

DeepStream can be installed via Jetpack and comes with several demos to get individuals started on coding for machine learning and computer vision applications [15]. On a device that is set up, DeepStream can be installed via sudo apt-get.

Nvidia DeepStream SDK is an important component of NVIDIA Metropolis, which has one of its applications collection vehicles taxes in Jakarta [15]. In the past, it has also been found on Tesla cars [16]. Many of the sample applications associated with this SDK are designed to run on self-driving cars [17].

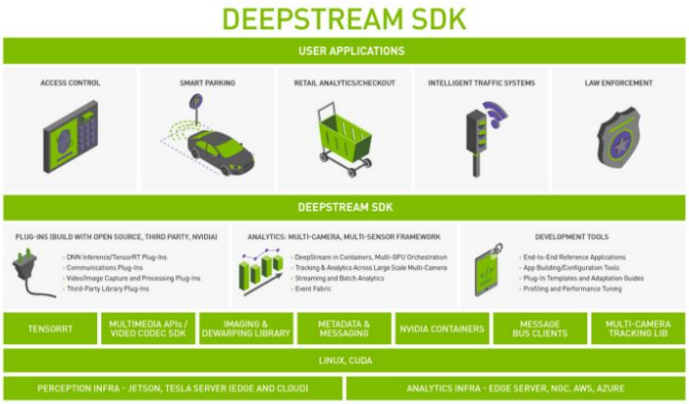


Figure 3: Some of the applications of Deepstream SDK
Source: [18]

4.1.3 Isaac SDK

The Isaac SDK is developed to support the control of mobile robots [14]. It has a structure similar to that of ROS with support for visualization, simulation, and more. The Isaac SDK is fundamentally based on the Isaac Robot Engine with support for a variety of math-planning and visualization algorithms [18]. Simulation is a key stage in robotics design and a critical component to prevent product failures [19]. The SDK package also includes tutorials for several basic packages controlling existing robots, such as the Kaya.

Figure 4: Various functions offered by Isaac
Source: [20]

4.1.4 TensorRT

TensorRT comes with the ONNX (Open Neural network Exchange) parser with support for video-processing and natural language processing [21]. It provides good support for general AI and machine-learning applications. The following diagram indicates how TensorRT functions within a larger system. On the diagram, below it is indicated that Tensorflow takes in one of the common machine-learning libraries and outputs directly to the physical hardware of the embedded system.

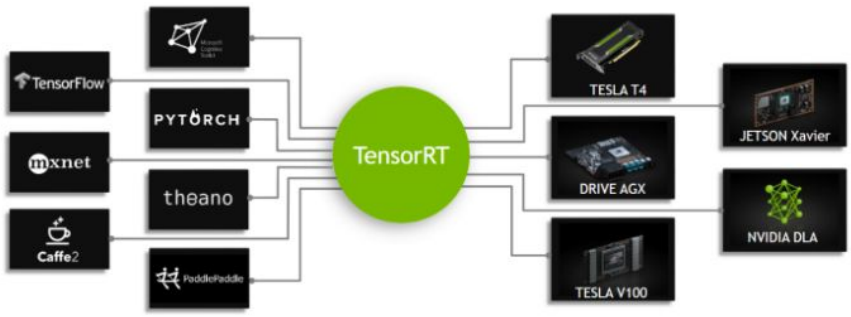


Figure 5: Layout of TensorRT functionality
Source: [22]

4.2.0 - Useful Programming Libraries

The aforementioned SDKs are useful but not necessarily enough to meet all the possible applications of a GPU-accelerated system. Instead, engineers should familiarize themselves with the common software libraries that are implemented using GPUs. While electrical and computer engineers may not interact with these libraries to a very-detailed level, the computer scientists and programmers that they work with will. As ECE bridges the gap between the software and hardware components of the system, an understanding of common GPGPU programming paradigms supports the overall project completion process.

4.2.1 General GPU Access

CUDA (Compute Unified Device Architecture) is an Nvidia-produced GPGPU API designed by Nvidia to facilitate GPGPU programming for nonspecialists [23]. CUDA interacts directly with [the threads and GPU and allows users to specify what threads they want to be executed in parallel as well as if they would like for it to be executed on multiple GPUs. It is a lower-level coding language, specially developed by Nvidia for controlling their GPUs. CUDA

has widespread implementations in deep learning, computational physics, and signal processing. CUDA code can be compiled directly on the GPU.

Nvidia also provides CUDA support in the form of the CUDA Toolkit, providing parallel algorithm support, CUDA signal processing, and CUDA linear algebra libraries [24]. This allows CUDA code to be easily fitted upon the existing C code [25]. There are alternatives to CUDA, but Nvidia support for CUDA in the form of continuous maintenance and Nvidia's dominance of GPU technology, updates make it an ideal option.

Of course, not all programmers are necessarily interested in diving into CUDA programming, and there are other interfaces to take command of the GPU. Some commonly applied libraries are OpenCL, OpenACC, Numba, and CUDA. These libraries allow programmers to have easy access to the GPU, and also allows them to execute traditional GPU applications, granting them powerful control over the specific application of the GPU. Numba is a python library with inbuilt functions that allow quick and easy access to the GPU, with specific lines to specify the parts of code to be executed in parallel. Numba is based around the syntax of NumPy and Scipy, common libraries for handling the representation of code as matrices for mathematical computation. Numba compiles simultaneously for the CPU and the GPU and dramatically increases array processing [26].

OpenACC provides similar support, though in the C language. OpenACC accomplishes parallel programming via a series of simple, easy-to-understand preprocessor directives. Once the environment has been properly set up, the parallelism is fairly straight-forward to achieve. However, OpenACC isn't simply restricted to these low-level applications, and is powerful enough to give users control over the ways they desire to parallelize loops and allocate data [27].

4.2.2 Machine Learning Libraries

It must be emphasized, even with the application of GPGPU programming to other fields, the major application of GPU programming remains rooted in the area of machine learning and image processing. This means that an engineer working with a GPU should be familiar, if not fluent, with the common machine libraries used in industry. Tensorflow and PyTorch are among the most-popular and widely-used.

Facebook's PyTorch is an open-source, machine-learning library derived from the Torch, a repository of machine-learning libraries written in Lua. In simple terms, its two top-level functions are access to machine-learning algorithms, and a numpy based interface that supports the GPU-acclerated image processing. Broadly speaking, its API is divided into the following components: torchtext for natural langauge processing, torchaudio for spoken language processing, torchvision for image processing, and torch for general access to machine-learning algorithms.

Google Brain's Tensorflow is written C++ with Python bindings. In Summer 2019, Google released Tensorflow2, a major overhaul of the previous version of Tensorflow. Though it has

support for Java and Go, the most stable version of Tensorflow2 is the python version. Just as PyTorch implements the Torch interface, Tensorflow relies on the Keras machine-learning interface. Its API is more detailed, but also harder to use as a beginner, with separate libraries for audio processing, data handling, debugging, and error checking, as well as a Keras API (tf.keras) for training models.

Both libraries can accomplish the same end-goal, but there are minor differences between the two that can dramatically influence project deadlines, depending on the needs of the individual engineer. PyTorch's syntax is rooted in NumPy and is widely regarded as having better API documentation and support, but lacking a native data visualizer that Tensorflow provides, meaning that users will have to export data to an alternate visualizer to track the process of their programs. In addition, Tensorflow 2 is relatively new. Unlike other software updates, Tensorflow 2 dramatically overhauled the previous version of Tensorflow [28, 29]. While Tensorflow 2 promises advantages over the previous version, it is young and untested, meaning that programmers that run into issues have relatively fewer sources to consult compared to users of PyTorch.

5.0 - Annotated Bibliography

[1] B. Caulfield, “Difference Between a CPU and a GPU?,” *The Official NVIDIA Blog*, 16-Dec-2009. [Online]. Available: https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/. [Accessed: 06-Feb-2020].

[2] I. Buck, *The Evolution of GPUs for General Purpose Computing*: GPU Technology Conference, September 20-23, 2010, San Jose, CA, US.

[3] P. Zunitch, “CUDA vs. OpenCL vs. OpenGL,” July 2018, [Online]. Available: Videomaker, http://videomaker.com Videomaker. [Accessed January 29, 2020].

[4] B. Caulfield, “What’s the Difference Between a CPU and a GPU?” December 16, 2009. [Online]. Available: Nvidia, http://blogs.nvidia.com. [Accessed January 22, 2020].

[5] “Introduction to Image Processing on the GPU,” p. 6.

[6] H. Scheidl, “GPU Image Processing using OpenCL,” *Medium*, 05-Jan-2019. [Online]. Available: https://towardsdatascience.com/get-started-with-gpu-image-processing-15e34b787480. [Accessed: 06-Feb-2020].

[7] S. Perkins, *Computer Science 306: Computer Architecture*, Chapter 10. [E-book] Available: Study.com

[8] Github, “Features - The right tools for the job.” Available: Github, http://github.com

[9] Nvidia, “Nvidia Management Library (NVML).” Available: Nvidia, http://developer.nvidia.com

[10] Pham, Vu & Vo, Phong & Vu, Hung & Le, Bac. (2010). GPU Implementation of Extended Gaussian Mixture Model for Background Subtraction. 1 - 4. 10.1109/RIVF.2010.5634007.

[11] Nvidia, “Introduction to Jetpack,” 2019. Available: http:/docs.nvidia.com

[12] S. Kaul, “Explained Output of nvidia-smi Utility,” December 15, 2019, Available: Medium, http://medium.com

[13] “The new NVIDIA Jetson TX2 module may be just what edge devices need for AI computing.” [Online]. Available:

https://www.hardwarezone.com.sg/tech-news-new-nvidia-jetson-tx2-module-may-be-just-what-edge-devices-need-ai-computing. [Accessed: 06-Feb-2020].

[14] M. Zensius, “Jetson TX2 Developer Kit,” p. 24.

[15 “System Requirements.” [Online]. Available: http://docs.nvidia.com/sdk-manager/system-requirements/index.html. [Accessed: 06-Feb-2020].

[16] “kdb374: Installing Jetpack SDK Components alongside the CTI-L4T BSP,” *Connect Tech Inc.* [Online]. Available: http://connecttech.com/resource-center/kdb374/. [Accessed: 06-Feb-2020].

[17] “NVIDIA DeepStream SDK,” *NVIDIA Developer*, 09-Sep-2016. [Online]. Available: https://developer.nvidia.com/deepstream-sdk. [Accessed: 06-Feb-2020].

[18] “Accelerated GStreamer User Guide,” p. 40.

[19] “DeepStream Plugin Manual.” [Online]. Available: https://docs.nvidia.com/metropolis/deepstream/plugin-manual/index.html#page/DeepStream_Plugin_Manual%2Fdeepstream_plugin_faq.html%23wwpID0EGHA. [Accessed: 06-Feb-2020].

[20] M. Tu, “Jakarta Uses NVIDIA Metropolis to Retrieve Unpaid Vehicle Tax | NVIDIA Blog,” *The Official NVIDIA Blog*, 04-Dec-2019. [Online]. Available: https://blogs.nvidia.com/blog/2019/12/03/nodeflux-jakarta-smart-city/. [Accessed: 06-Feb-2020].

[21] J. Jeun, “DEEPSTREAM SDK 2.0 WEBINAR,” p. 47.

[22] K. Purandare, “AN INTRODUCTION TO DEEPSTREAM SDK,” p. 19.

[23] “New NVIDIA DeepStream SDK 3.0 Removes Boundaries of Video Analytics – NVIDIA Developer News Center.” [Online]. Available: https://news.developer.nvidia.com/new-nvidia-deepstream-sdk-3-0-removes-boundaries-of-video-analytics/. [Accessed: 06-Feb-2020].

[24] “NVIDIA Isaac SDK | NVIDIA Developer.” [Online]. Available: https://developer.nvidia.com/isaac-sdk. [Accessed: 06-Feb-2020].

[25] “Isaac SDK Developer Guide — Isaac 2019.1 documentation.” [Online]. Available: https://docs.nvidia.com/isaac/archive/2019.1/doc/index.html. [Accessed: 06-Feb-2020].

[26] “NVIDIA Isaac SDK Brings Modern AI to Autonomous Machines | NVIDIA Blog.” [Online]. Available:

https://blogs.nvidia.com/blog/2019/03/18/isaac-sdk-general-availability/. [Accessed: 06-Feb-2020].

[27] “NVIDIA TensorRT,” *NVIDIA Developer*, 05-Apr-2016. [Online]. Available: https://developer.nvidia.com/tensorrt. [Accessed: 06-Feb-2020].

[28] “whatistrt2.png（PNG 圖片，1373x530 像素）.” [Online]. Available: https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/graphics/whatistrt2.png. [Accessed: 06-Feb-2020].

[29] P. Zunitch, “CUDA vs. OpenCL vs. OpenGL,” *Videomaker*, 24-Jan-2018. [Online]. Available: https://www.videomaker.com/article/c15/19313-cuda-vs-opencl-vs-opengl. [Accessed: 06-Feb-2020].

[30] M. Heller, “What is CUDA? Parallel programming for GPUs,” *InfoWorld*, 30-Aug-2018. [Online]. Available: https://www.infoworld.com/article/3299703/what-is-cuda-parallel-programming-for-gpus.html. [Accessed: 06-Feb-2020].

[31] “Seven Things You Might Not Know about Numba,” *NVIDIA Developer Blog*, 03-Oct-2017. [Online]. Available: https://devblogs.nvidia.com/seven-things-numba/. [Accessed: 06-Feb-2020].

[32] “OpenACC: Directives for GPUs,” *NVIDIA Developer Blog*, 12-Mar-2012. [Online]. Available: https://devblogs.nvidia.com/openacc-directives-gpus/. [Accessed: 06-Feb-2020].

[33] “GPU support,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/install/gpu?hl=zh-tw. [Accessed: 06-Feb-2020].