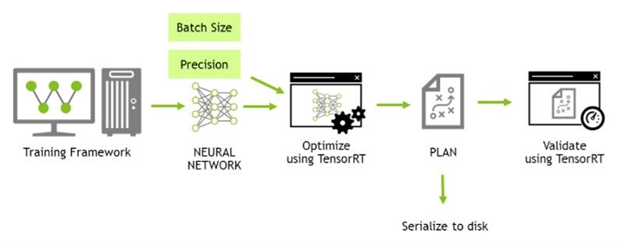
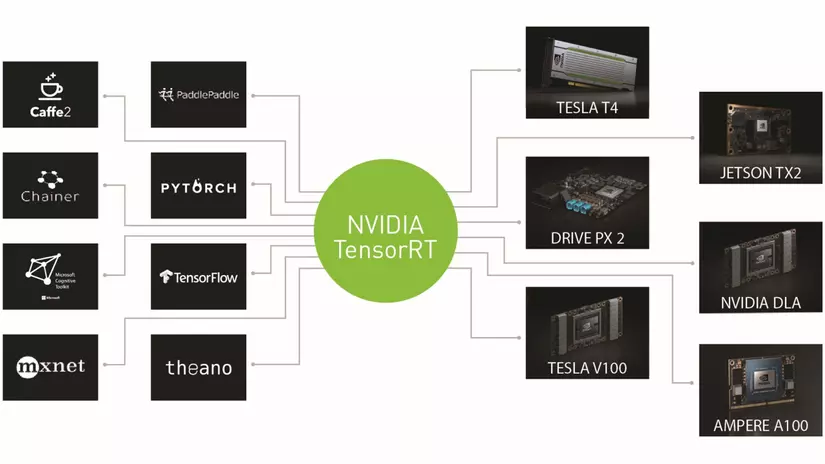
TensorRT

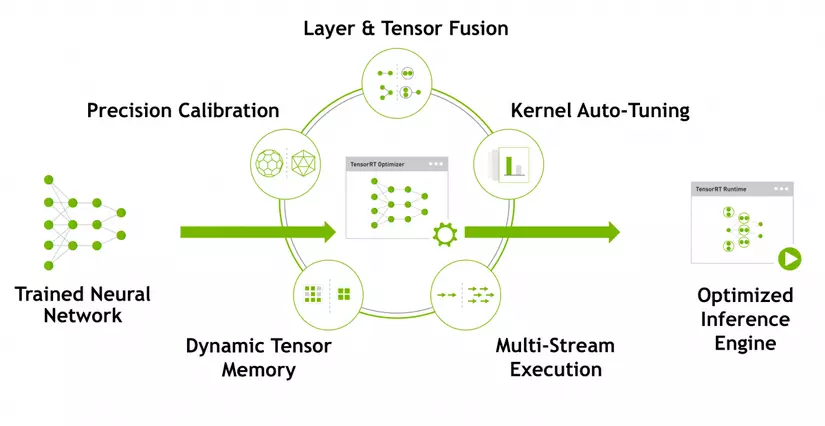
# Introduction:

* TensorRT là một thư viện được phát triển bởi NVIDIA nhằm cải thiện tốc độ inference, giảm độ trì truệ trên các thiết bị đồ họa NVIDIA(GPU). Nó có thể cải thiện tốc độ suy luận lên đến 2-4 lần so với các dịch vụ real-time và nhanh hơn gấp 30 lần so với hiệu suất của CPU.





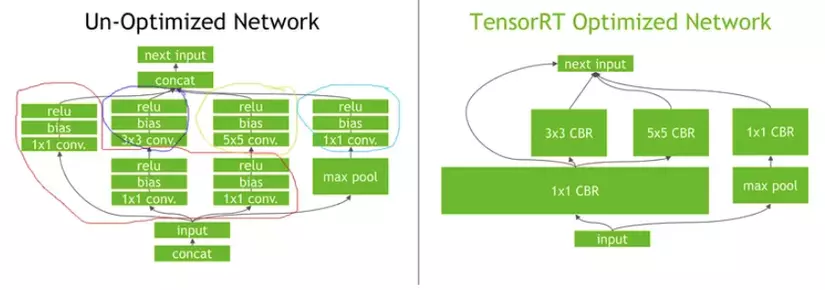
* TensorRT cải thiện việc tối ưu như thế nào (5 laoị tối ưu để tăng hiệu suất inference):



* + Precision Calibration:
    - Các tham số và hàm kích hoạt (activations) có **độ chính xác FP32 (Float Point 32) được convert về độ chính xác FP16 hoặc INT8**
    - Giúp giảm độ trì trệ và tăng tốc độ infer nhưng ngược lại độ chính xác bị giảm (không đáng kể). Phù hợp với real-time



* + Layer & Tensor Fusion:
    - Là các kỹ thuật tối ưu hóa mô hình AI để cải thiện hiệu suất thời gian chạy bằng cách **kết hợp nhiều lớp (layers) và tensors lại với nhau** => giảm thiệu số lượng phép tính và tối ưu hóa tính toán song song GPU.
      * Layer Fusion: kết hợp nhiều lớp trong mô hình thành một lớp duy nhất có chức năng tương đương.
        + Ví dụ: với convolutional layer có bias và activation function là relu => TensorRT sẽ phát hiện cấu trúc tương tự và kết hợp lại thành 1 lớp duy nhất là CBR (ConvolutionalBiasRelu) -> giúp làm giảm số lượng phép tính
      * Tensor Fusion: Là quá trình kết hợp các phép tính trên tensor lại với nhau để giảm số lượng phép tính cần thiết.
        + Ví dụ: với convolutional layer và batch normalization. TensorRT có thể kết hợp thành ConvulutionalBatchNorm layer -> giảm số lượng tensor trung gian.



* + Kernel Auto-Tuning:
    - Là quá trình tự động tìm kiếm và tối ưu các tham số và cấu trình của các kernel tính toán trên GPU để đạt được hiệu suất tối đa cho mô hình khi triển khai trên TensorRT
    - Tối ưu hóa bằng cách thử nghiệm 1 loạt các giá trị khác nhau để tìm ra các giá trị tối ưu cho mô hình và GPU cụ thể
    - Chọn các kernel tối ưu dựa trên các tham số của mô hình:
      * Batch\_size
      * Filter\_size
      * Input data size
  + Dyanmic Tensor Memory:
    - Cho phép mô hình sử dụng quản lý bộ nhớ động cho các tensor, giúp tiết kiệm tài nguyên và tăng tính linh hoạt.
  + Multiple-Stream Execution:
    - Là một tính năng cho phép mô hình triển khai trên TensorRT chạy động thời trên nhiều luồng của GPU -> giúp cải thiện hiệu suất và tăng tốc độ thời gian chạy của mô hình.
* Cơ chế hoạt động của TensorRT:
  + Lượng tử hóa (Quantization): giảm kích thước của các tham số của mô hình bằng cách giảm số lượng bit sử dụng để biểu diễn các tham số.
  + Tối ưu (Pruning): Loại bỏ các trọng số không cần thiết bằng cách đặt chúng bằng 0. Được phát hiện bằng các thuật toán như (LK1 regularization, Taylor approximation)
  + Kết hợp (Fusion): kết hợp các phép tính nhỏ thành 1 phép tính lớn. Thường áp dụng với các phép tính như: Element-wise, Convulotional, Batch Normalization, Element-wise activation, Multi-Head attention.
  + Tối ưu kernel (Kernel auto-tuning): tìm kiếm kernel tốt nhất để tính toán các phép tính trên GPU bằng cách thử nghiệm nhiều kernel với các thông số khác nhau.
  + Tinh chỉnh độ chính xác (Precision calibration): Tìm kiếm giá trị có

# Hệ sinh thái của TensorRT:

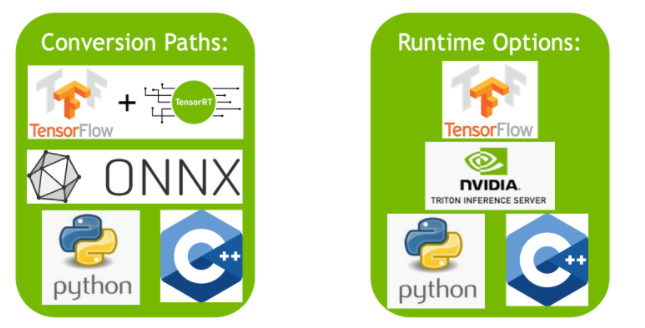
## Basic TensorRT Workflow:

A diagram of a model

Description automatically generated

## Các lựa chọn để chuyển đổi và phát triển:

* Hệ sinh thái của TensorRT có 2 phần chính:
  + Chuyển đổi models sang TensorRT engines được tối ưu hóa.
  + Sử dụng các runtime mục tiêu là TensorRT khi deploy TensorRT engines tối ưu.



* Chuyển đổi mô hình:
  + Có 3 lựa chọn chính để chuyển đổi mô hinh với TensorRT:
    - Sử dụng TF-TRT:
      * Cho phép cả chuyển đổi model và high-level runtime API, đồng thời có khả năng quay lại TensorFlow implementations khi TensorRT không hỗ trợ 1 số phép toán cụ thể.
    - Chuyển đổi tự động từ .onnx files:
      * Là cách đơn giản nhất, tự động chuyển đổi mô hình và deploy sử dụng ONNX.
      * ONNX là framework tương thích với TensorFlow, PyTorch, …
      * TensorRT hỗ trợ chuyển đổi tự động từ ONNX files sử dụng TensorRT API hoặc trtexec.
      * Chuyển đổi ONNX (1 là thành công hoặc không) nghĩa là cần phải sử dụng các operations mà được hỗ trợ bởi TensorRT hoặc phải cung cấp thêm custom plug-in cho những operations không được hỗ trợ. Xem các operations hỗ trợ tại link: https://docs.nvidia.com/deeplearning/tensorrt/support-matrix/index.html#supported-ops
      * Kết quả của chuyển đổi ONNX là 1 TensorRT egine duy nhất với chi phí thấp hơn sử dụng TF-TRT
    - Tự xây dựng mạng sử dụng TensorRT API (có thể sử dụng C++ hoặc Python):
      * Để cho hiệu suất và tính linh hoạt cao nhất, có thể tự xây dựng TensorRT egines sử dụng TensorRT network definition API.
      * Về cơ bản, điều này liên quan đến việc xây dựng một mạng giống hệt với mô hình mục tiêu của bạn trong hoạt động TensorRT theo từng hoạt động, chỉ sử dụng các hoạt động TensorRT.
      * Sau khi mạng TensorRT được tạo ra, sẽ export weights của mô hình từ framework muốn chuyển đổi và load nó vào mạng TensorRT.
      * Các thông tin thêm có thể xem dưới link sau:
        + <https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#create_network_c>
        + <https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/#create_network_python>
* Phát triển (deploy):
  + Có 3 lựa chọn cho deploy model với TensorRT:
    - Sử dụng TensorFlow:
      * Sử dụng TF-TRT
    - Sử dụng TensorFLow runtime API:
      * Cho kết quả có chi phí và finest-grained control thấp nhất, tuy nhiên các operators mà TensorRT không hỗ trợ cần phải được implemented như plug-ins (thư viện hoặc plug-ins viết trước trong link này: <https://github.com/NVIDIA/TensorRT/tree/main/plugin>)
      * Hướng thường được sử dụng là chuyển đổi framework sang ONNX
    - Sử dụng NVIDIA Triton Inference Server
      * Là một mã nguồn mở inference-serving software cho phép deploy các mô hình AI đã được huấn luyện từ bất kì framework nào từ local storage hoặc Google Cloud Platform AWS S3 on any GPU- or CPU-based infrastructure (cloud, data center, or edge).
      * Có thể tham khảo ở đây:
        + <https://developer.nvidia.com/nvidia-triton-inference-server>
        + <https://github.com/triton-inference-server/server/blob/r22.01/README.md#documentation>

## Chọn Workflow phù hợp:

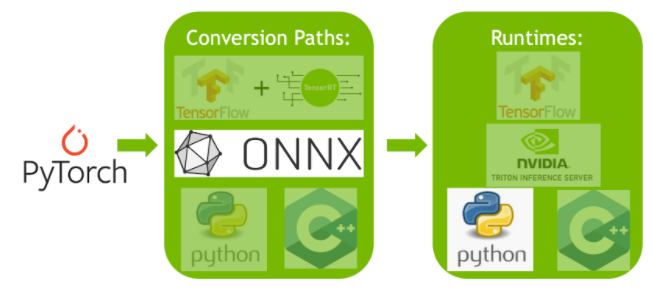
* Hai yếu tố chính để xác định cách để convert và deploy model là:
  + Framework sử dụng
  + TensorRT mục tiêu

A diagram of a program

Description automatically generated

# Hướng dẫn deploy sử dụng ONNX:

* Bước 1: Export the model



* + Ví dụ:

A screenshot of a computer program

Description automatically generated

* Bước 2: lựa chọn batch size:
  + Chọn small batch size nếu muốn ưu tiên độ trễ và larger batch size khi muốn ưu tiên thông lượng. Larger bathes tốn thời gian để xử lý nhưng giảm thời gian trung bình trên từng mẫu.
  + Có thể sử dụng batch size dynamic theo link dưới đây: <https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#work_dynamic_shapes>
* Bước 3: lựa chọn độ chính xác
  + Các độ chính xác hỗ trợ: TF32, FP32, FP16, INT8
  + Default là FP32:



* Bước 4: Convert the model
  + Sử dụng lệnh trtexec



* Bước 5: Deploy model
  + Sau khi tạo thành công TensorRT engine, chúng ta cần xác định chạy nó như thế nào bằng TensorRT:
    - Có 2 TensorRT runtimes:
      * A standalone runtime that has C++ and Python bindings
      * A native intergration into TensorFlow
    - Sử dụng wrappper được đơn giản hóa ONNXClassifierWrapper, nó sẽ gọi standalone runtime

A screenshot of a computer

Description automatically generated