소스코드 구조 및 내용 분석

ExcuTorch란?

모바일, 임베디드, 엣지 장치 등에서 온디바이스 AI 추론을 하기 위한 PyTorch 솔루션이다.

Executorch의 작동 방식

작동 방식은 크게 3단계의 프로세스로 나누어진다.

1. 모델 내보내기
2. 모델을 ExecuTorch 프로그램으로 컴파일
3. ExecuTorch 프로그램을 실행

ExecuTorch의 장점, 그리고 핵심 기능

ExecuTorch는 다음과 같은 이점이 있다.

- PyTorch와 동일한 export를 사용하여 강력하고 PyTorch와의 호환이 좋다.
- **Core ATen Op set**이라는 작은 연산자 세트로 PyTorch의 연산자를 나타낼 수 있어 모델이 가벼워진다.
- 컴파일러에 작업을 위임하기 위한 표준화 인터페이스를 가지고 있어, 특정 하드웨어나 소프트웨어에 연결이 용이하다.
- 다른 서드파티를 위한 변환이 필요없다.
- 성능 향상을 위한 사용자 정의 최적화가 쉽다.
- 모델을 실행하는 **런타임이 C++로 작성되어 매우 가볍다.** 그리고 실행에 필요한 컴파일, Memory Planning 등을 사전에 완료하기 때문에, 런타임에는 정말 실행만 하면 된다.

모바일, 임베디드, 엣지와 같은 환경에서는 Pytorch 모델을 기반한 추론에 **자원적 제약**이 존재한다.

Executorch는 이러한 제한적 환경에서 추론을 가능케하도록 돕는 **이식성**, 개발자가 항상 동일한 파이프라인을 적용할 수 있는 **생산성**, 가벼운 런타임과 하드웨어를 기반한 **고성능의 환경**을 제공한다.

이를 통해 **자원적 경량화, 범용성, 인터넷 연결에 구애받지 않는 독립적인 온디바이스 추론**을 구현 및 배포할 수 있도록 하는 것이 Executorch가 제공하는 **핵심 기능**이다.

아키텍쳐

모델을 배포하는 과정은 다음과 같이 크게 3가지로 나타낼 수 있다.

1. Program Preparation (Ahead-Of-Time, AOT)

- 더욱 경량화된 런타임과 실행을 위해, Export, Complie 등 실행되기 위한 준비를 가능한 몰아넣은 단계이다.
- 다음과 같은 세부 단계로 이루어져 있다.
- Export
  - PyTorch 모델(`.nn.Module`)에서 시작하여, 백엔드에서 실행할 모델을 컴파일하기 위한 그래프 형태로 변환한다.
  - ATen Dialect
    - `torch.export()`가 이 역할을 하며, 사용하면 ATen과 호환되는 그래프가 생성된다.
    - 이 과정은 Autograd에 대하여 안전하다.
  - 옵션으로 ATen 그래프를 Core ATen으로 변환하기 전에 양자화하여 모델 크기를 줄일 수 있다.
  - Core ATen Dialect
    - ATen에 있는 수천 개의 연산자들을 적은 수의 핵심 연산자들로 분해한다. 덕분에 표준화되고 더 작아진 연산자 집합 (위에서 언급한 Core ATen Op Set)이 만들어져 더 낮은 레벨의 변환이 가능해진다.
- Edge Compilation
  - 위에서 내보낸 그래프를 엣지 기기에 친화적인 형태로 변환하는 과정이다.
  - `to_edge_transform_and_lower()`를 통해 .pte 파일이 생성된다.
  - Edge Dialect
    - Core ATen 그래프에 데이터 타입(dtype)이나 메모리 저장 순서와 같은 임베디드 환경에 필요한 정보를 추가하여 구체화하는 과정.
  - BackEnd Dialect
    - 그래프 전체나 일부를 IOS의 Core ML, 퀄컴의 QNN, ARM의 TOSA와 같은 특정 하드웨어에서의 성능 향상을 위해 해당 하드웨어에게 위임하여 더 Low Level로 변환하는 과정
  - User-defined passes
    - 사용자가 할 수 있는 특정 타겟에 대한 변환. 그 예시로 커널 퓨전, 비동기 동작, 메모리 레이아웃 변환 등이 있다.
- Complie to ExecuTorch Program
  - 그래프를 런타임 환경에 적합하도록 변환한다.
  - 엣지 기기에서 상당한 성능 저하와 전력 낭비를 발생시키는 동적 메모리 할당을 방지하기 위해, 미리 Memory Planning을 세운다. 이는 정적 그래프로 표현된다.
  - 연산 결과를 만들지 않고 인수로 출력을 전달하여 불필요한 메모리 사용을 막는다.
  - 최종적으로 Flatbuffer(`.pte` 파일)로 직렬화되어, 런타임이 인식가능한 형태로 Export 된다. → model.pte

2. Runtime Preparation

직렬화된 프로그램과 연산자를 호출하는 제공된 커널 라이브러리, 혹은 특정 하드웨어에 위임하기 위한 백엔드 라이브러리를 사용하여 런타임을 준비하는 것.

3. Program Execution

ExecuTorch 런타임은 C++로 작성되어 이식성과 효율성이 높다. 앞서 언급한 단계로 인해 런타임 구성 요소를 최소화했으며, 다음과 같다.

- Platform abstraction layer → 시스템 호출, 메모리 접근, 파일 I/O 같은 기능을 추상화하는 단계.
- Logging and optionally profiling → 런타임 로그를 생성하고, 선택적으로 프로파일링
- Execution data types → 런타임에서 사용하는 경량 구조체 EtTensor 메타데이터.
- Kernel and backend registry → 커널을 함수 포인터에, backend Delegator를 실행기 객체에 연결하는 Mapper. 런타임이 .pte 파일을 읽으면 그래프 노드 별 연산자를 찾고 이를 커널 레지스트리에서 조회해 실행하는 구조. 백엔드 역시 동일하다.
- Memory management → Memory Plan에 따라 메모리를 할당하고 제어. MemoryAllocator의 allocate 메서드를 오버라이딩 하여 플랫폼 별 할당을 진행.

소스 코드 구조

ExecuTorch 소스코드 구조를 위에서 정리한 프로세스와 연결지어 보았다.

- executorch
  - backends →애플의 CoreML, XNNPACK 등 특정 하드웨어에 대한 delegate 구현. Edge Compliation의 Backend Dialect를 담당한다.
    또한 하위 디렉토리 명은 고유한 Delegate 이름을 갖는다.
  - codegen → kernel과 runtime 간의 바인딩을 생성하는 도구
  - configurations
  - devtools → 개발자용 도구
  - examples → export, delegate, runtime 실행 예제 코드
  - exir → AOT. 모델 캡쳐 및 저장, 그래프 저수준 변환 등 라이브러리. Program Preparation에 해당된다.
  - export → PyTorch 모델을 export 하기 위한 도구와 유틸. Program Preparation에서 export 부분을 담당한다.
  - extension → 안드로이드/IOS 래퍼, C++ 모듈, 파이썬 바인더
  - kernels → ATen 기본 연산자, 특수 연산, 양자 연산
  - profiler
  - runtime → 경량 C++ 런타임. 백엔드 위임, 모델 로드 및 실행, 커널 등록 및 관리, 플랫폼 추상화 담당
    - executor → 모델 로드 및 실행
    - core → 텐서 타입, value 객체, shape, 메모리 할당 및 plan. Excution data types에 해당.
    - backend → 런타임에서 backend를 호출할 수 있는 인터페이스 제공. Kernel and backend registry의 백엔드 부분에 해당.
    - kernel → 연산자. Kernel and backend registry의 커널 부분에 해당.
    - platform → Platform abstraction layer에 해당.
  - schema → .pte 를 위한 Flatbuffer 스키마
  - scripts
  - test → end-to-end 테스트 코드
  - third-party → 외부 라이브러리 Dependency

지원 범위 및 한계점

ExecuTorch는 다음과 같은 Support를 제공한다.

Model Types

- Large Language Models (LLMs)
- Computer Vision (CV)
- Speech Recognition (ASR)
- Text-to-Speech (TTS)

Platforms

- Android & iOS
- Linux, macOS, Windows
- Embedded & MCUs

Rich Acceleration

- CPU
- GPU
- NPU
- DSP

필자가 생각한 한계점은 다음과 같다.

- Python 3.10-3.12, PyTorch 2.9 이상에서만 동작한다. 이러한 의존성 때문에, PyTorch 모델에 따른 구현이 필요하게 된다.
- 실행되는 기기 환경의 제약을 받기 때문에 더 낮은 성능, 작은 모델을 쓰게 되고, 그로 인해 상대적으로 정확도가 떨어질 수 있다.
- 사용자 정의 작업을 구현하려면 개발자가 YAML을 작성하고 관리해야 하는데, 이런 프로세스는 개발 속도가 더뎌지게 만들 것으로 예상한다.
- Core ATen Op Set는 완전 무결하지 않다. 해당 이슈 작성자의 디버깅 과정에서 이전에 놓쳤던 연산자 프로파일링을 통해 런타임을 16초에서 3.5초로 단축시킨 사례가 있다. FAQ 에서도 이 부분에 대해 언급하고 있다.