경량 언어 모델(SLM) 학습을 위한 웹 기반 시각화 플랫폼 설계 및 구현

2025 전기 졸업과제 중간보고서 <SLaM Dunk>

제출일	2025.5.15	전공	정보컴퓨터공학부
팀장	김명석	팀명	SLaM Dunk
팀원	염현석 정지윤	지도교수	조준수

1. 연구목표

1.1. 인공지능과 SLM의 발전

1.1.1.

최근 인공지능(AI)과 자연어 처리(NLP) 분야에서 대규모 언어 모델(Large Language Model, LLM)의 등장이 혁신을 주도하고 있다. 그러나 LLM은 막대한 하드웨어 자원과 높은 운영 비용을 요구하기 때문에, 실무 환경에서 적용하기에는 한계가 존재한다.

1.1.2.

이러한 한계를 극복하기 위해 **SLM(Small Language Model)**이 주목받고 있다. SLM은 상대적으로 작은 규모의 모델로, 특정 도메인이나 작업에 맞춰 최적화되어 적은 자원으로도 우수한 성능을 발휘한다.

1.1.3.

대표적인 예로 Meta의 LLaMA 2-7B, Mistral AI의 Mistral 7B, Google의 Gemma 시리즈 등이 있으며, 이러한 모델들은 경량화된 구조와 최적화된 알고리즘을 통해 높은 효율성을 제공한다.

1.2. SLM의 필요성과 가능성

1.2.1.

SLM은 적은 메모리와 연산 자원으로도 학습과 추론을 수행할 수 있어, 개인 연구자나 중소 규모의 프로젝트에서도 활용이 가능하다. 특히 GPU 자원이 제한된 환경에서도 실험과 최적화를 진행할 수 있다는 장점이 있다.

1.2.2.

또한 SLM은 특정 분야에 맞춰 도메인 특화 모델로 쉽게 커스터마이징할 수 있으며, 파인튜닝(fine-tuning)과 지식 증류(knowledge distillation) 기법을 활용하면 LLM에 근접한 성능을 확보할 수도 있다.

1.2.3.

이러한 특성 덕분에 교육, 의료, 금융, 법률 등 다양한 분야에서 비용 효율적인 언어 모델로 활용되고 있다.

1.3. 연구 필요성과 목표

1.3.1.

대부분의 상용 LLM은 대규모 자원을 필요로 하고, 모델의 내부 동작 원리를 이해하기 어렵다. 반면 SLM은 구현과 학습, 운영의 부담을 줄이면서도 모델 구조와 최적화 과정을 직접 경험할 수 있다는 장점이 있다.

1.3.2.

본 연구는 SLM의 핵심 구성 요소인 토크나이저(Tokenizer), 임베딩 층(Embedding Layer), 트랜스포머 블록(Transformer Block) 등을 직접 구현하고, 단계별로 모델을 구축하면서 성능과 최적화 가능성을 분석하는 것을 목표로 한다.

1.3.3.

이를 통해 SLM의 내부 구조와 학습 과정을 심층적으로 이해하고, 향후 다양한 응용 프로그램과 실무 환경에서 활용 가능한 경량 언어 모델을 구축하는 기반을 마련하고자 한다.

2. 문제상황

2.1 모델 설계 복잡성

최근 인공지능(AI) 및 딥러닝 기술의 발전으로 자연어 처리, 이미지 인식, 음성 인식 등 다양한 분야에서 혁신적인 성과가 나타나고 있다. 특히 딥러닝 모델의 구조가 점점 고도화되면서 트랜스포머(Transformer) 기반 모델이나 복잡한 신경망 구조를 활용하는 사례가 증가하고 있다. 그러나 이러한 모델을 직접 설계하고 학습시켜 보는 과정은 아직도 많은 사람들에게 진입장벽이 높다. 모델 설계에 대한 이론적 이해뿐 아니라, 파이프라인 구성, 라이브러리 사용법, 하드웨어 자원 관리 등 여러 요소를 종합적으로 고려해야 하기 때문이다.

2.2 LLM 교육 문제 1 - 성능 확인 환경 부족

현재 교육 현장이나 개인 학습 차원에서 딥러닝 모델을 직접 구축하고 실험해 보는 방식은 대개 다음과 같은 제한이 있다. 첫째, 텍스트나 영상 강의 위주의 이론 학습과 실제 코딩 실습의 간극이 크다. 이론을 이해하더라도, 이를 바로 코드로 구현하는 과정에서 난관에 부딪히는 경우가 많아 모델 구조 설계나 하이퍼파라미터 튜닝 등에 대한 실질적인 학습 기회가 제한된다. 둘째, 다양한 딥러닝 아키텍처를 실시간으로 비교하거나 실험 결과를 시각적으로 확인하기 쉽지 않다. 모델의 레이어나 블록을 유연하게 추가·변경하고, 그에 따른 성능 변화를 즉시 확인하며 학습할 수 있는 환경이 부족하다. 일반적으로는 TensorBoard, MLflow 등의 도구를 별도로 설치·활용해야 하며, 이와 같은 도구들에 대한 이해도와 추가 설정 작업이 필요하다. 초보자나 교육 대상자 입장에서는 모델 성능 변화를 직관적으로 모니터링하고 여러 실험 결과를 한눈에 비교·분석하기가 쉽지 않아, 반복 학습 및 피드백 과정이 원활하지 않다.

2.3 LLM 교육 문제 2 - 플랫폼 부족

이로 인해 학습자나 연구 초심자들은 직관적이고 시각화된 방식으로 딥러닝 모델을 조립하고, 성능을 비교·분석해보는 경험을 충분히 누리지 못하고 있다. 이에 따라 'Transformer Block'을 비롯한 여러 딥러닝 레이어를 드래그 앤 드롭(Drag & Drop) 방식으로 쉽게 설계하고, 모델을 직접 학습 및 비교·분석할 수 있는 웹 기반의 학습·실험 플랫폼이 필요하다. 그러나 현재 이러한 플랫폼이 충분히 갖춰져 있지 않아, 원하는 모델을 실제로 구성해 보고 성능을 개선하는 과정을 단일 인터페이스에서 수행하기가 어려운 실정이다. 또한 여러 라이브러리와 환경 설정을 개별적으로 익혀야 하므로, 학습자가 모델 구조의 변화가 성능에 미치는 영향을 직관적으로 파악하기까지 많은 시간과 노력이 소요되고 있다.

2.4 데이터셋 구성

아울러 데이터셋을 다루는 문제도 있다. 딥러닝 실습을 위해서는 일정 규모 이상의 학습 데이터셋을 준비해야 하는데, 이를 직접 수집·전처리하기 위해선 추가적인 기술적·시간적 비용이 발생한다. 기존 오픈소스 데이터셋을 활용한다고 하더라도, 웹 인터페이스 상에서 간편하게 불러와 쓰기가 쉽지 않고, 각기 다른 형식의 데이터셋을 재구성하는 과정이 필요하다. 이러한 어려움은 딥러닝 교육 및 실험의 진입 장벽을 더욱 높이고, 학습자들이 모델 설계나 성능 평가에 집중하기보다는 환경 세팅 및 데이터 준비에 대부분의 시간을 할애하게 만드는 원인이 된다.

2.5 비효율성

결과적으로, 초심자나 학생들이 이론과 실제 코딩을 매끄럽게 연결하여 딥러닝 모델을 설계·학습·평가해보는 통합적인 경험을 얻기 어려운 상황이다. 또한 이미 어느 정도 경험

이 있는 연구자나 개발자도 간단한 시제품 수준의 모델을 빠르게 구성하고 성능을 확인해보려 할 때, 매번 환경 설정과 모델 구현 과정을 반복해야 하는 비효율을 겪게 된다. 이는 딥러닝 학습과 연구 과정에서 발생하는 시간·노력·자원 낭비를 야기하며, 나아가 창의적인 모델 실험과 효율적인 교육 환경 구축을 저해하는 주요한 문제로 작용하고 있다.

3. 목표

본 프로젝트의 주된 목표는 웹 환경에서 직관적이고 체계적으로 딥러닝 모델을 설계하고 학습·비교할 수 있는 플랫폼을 구축하는 것이다. 이를 통해 모델 최적화를 실시간으로 실험하고 다양한 성능 지표를 시각적으로 확인함으로써, 사용자(교육 대상자 및 연구 초심자)가 딥러닝 개념을 빠르고 정확하게 이해할 수 있도록 돕는 것을 최종 목적으로 삼는다. 구체적으로 다음과 같은 세부 목표를 설정한다.

3.1 드래그 & 드롭을 활용한 직관적인 모델 설계

마우스 드래그 & 드롭으로 트랜스포머 블록이나 선형 레이어 등 다양한 레이어를 쉽게 배치하고 연결할 수 있는 그래픽 기반 인터페이스를 제공한다. 이를 통해 초심자도 복잡한 딥러닝 구조를 시각적으로 이해하며 설계 과정을 체험할 수 있다.

생성된 모델 그래프에서 레이어 간 연결 관계, 하이퍼파라미터 설정 등을 자유롭게 수정·확장할 수 있도록 하여, 다양한 실험 시나리오에 대응할 수 있게 한다.

3.2 빠른 학습 및 비동기 처리를 통한 사용자 경험 강화

웹 서버와 모델 학습 서버를 분리하거나 병렬 연산을 적극적으로 활용해 학습 속도를 향상시킨다. 이를 통해 사용자는 빠른 피드백 루프를 확보하고, 반복 실험을 손쉽게 진행할 수 있다.

모델 훈련이 백그라운드에서 이루어지도록 설계해, 웹 인터페이스가 멈추거나 지연되지 않도록 한다. 사용자는 다른 설정을 변경하거나, 이미 학습된 다른 모델과 성능을 비교하는 등 병행 작업이 가능해진다.

3.3 시각화된 학습 결과 및 모델 비교 제공

학습 과정에서의 손실 곡선, 정확도, F1 점수 등 주요 성능 지표를 그래프 형태로 시각화

하여, 모델이 어떻게 학습되고 있는지를 직관적으로 파악할 수 있도록 한다.

서로 다른 모델 구조(예: 레이어 개수 차이, 트랜스포머 블록 포함 여부 등)를 한 화면에서 비교 분석할 수 있게 하여, 어떤 요소가 성능 개선에 영향을 미치는지 빠르게 확인할 수 있도록 한다.

3.4 교육 목적 최적화 및 접근성 강화

데이터셋 업로드, 모델 구성, 학습, 평가 및 시각화 결과 확인까지 모든 과정을 한 웹사이트에서 통합적으로 처리할 수 있도록 하여, 복잡한 환경 설정이나 라이브러리 설치 부담을 최소화한다.

초심자도 다양한 모델 구조를 시도해보며 학습 과정에서 얻은 피드백을 토대로 모델을 개선해나갈 수 있게 한다. 이를 통해 딥러닝 개념과 최적화 과정을 자연스럽게 체득할 수 있는 학습 환경을 조성한다.

3.5 효율적 자원 활용 및 시스템 확장성

GPU, CPU 등 컴퓨팅 자원을 동적으로 할당·관리하여 사용자가 여러 명이 동시에 실습하더라도 웹사이트가 무리 없이 동작하도록 설계한다.

향후 추가되는 딥러닝 레이어(예: RNN, CNN 등)나 새로운 시각화·분석 기능을 손쉽게 통합할 수 있도록 시스템 구조를 모듈화하고, 확장성을 고려하여 개발한다.

4. 요구사항 및 제약 사항 분석에 대한 수정사항

4.1. REST API 통신이 아닌 사용자 각각 도커 파일을 로컬에서 실행하는 구조로 변경

학습한 모델을 추론하는 것이 아닌 처음부터 학습해야 되므로 GPU 자원이 많이 소요된다는 점, 사용자 간의 상호작용이 없고 개개인으로 모델을 설계하고 학습하고 평가해본다는 점에서 로컬로 실행할 수 있는 구조가 적합하다고 판단하였다.

4.2. D3.js 및 Chart.js 미사용

D3.js는 고도로 커스터마이징 가능한 강력한 시각화 도구지만, 낮은 수준의 DOM/SVG

제어를 요구하므로 학습 곡선이 가파르고 개발 시간이 상대적으로 많이 소요된다. Chart.js는 상대적으로 간단하지만, 프로젝트 목표에 맞는 동적 비교 및 세부 커스터마이징이 필요한 경우 확장성이 제한된다.

본 프로젝트는 정해진 기간 내에 연구 및 실습 기능 전반을 통합하는 것을 목표로 하므로, 시각화 구현에 과도한 시간을 투자하는 것은 전체 일정에 부담이 되었다.

MLflow는 이미 손실, 정확도, 파라미터, 메트릭 비교 등 기본적인 학습 과정 시각화를 지원하고 있으며, 실험 간 비교 기능도 내장되어 있어 본 프로젝트의 모델 비교 및 시각화에 부합하여 대체 가능하다.

4.3. WebSocket 미사용

WebSocket은 서버와 클라이언트 간 양방향 통신을 지속적으로 유지하기 위한 기술로, 비동기 서버 구성, 학습 상태와 로그 메시지를 이벤트 기반으로 브로드캐스트하는 큐 설계, 클라이언트 측에서는 React에서 WebSocket 핸들링 및 상태 동기화 구현 필요하다. 이러한 구성은 시간상 구현 부담이 크고, 로컬에서 실행하는 구조로 변경했기 때문에 WebSocket 사용성이 감소했다고 판단하였다.

5. 설계 상세화 및 변경 내역

5.1 상세 설계

5.1.1. 사용자 인터페이스 (Front-end)

- Vite + React + Tailwind CSS를 사용하여 사용자 친화적인 웹 기반 모델 학습 플랫폼을 구축
- React의 상태 관리 및 Virtual DOM을 활용하여 성능을 최적화하고, 컴포넌트 기반 아키텍처로 재사용성을 극대화
- Reactflow를 활용한 그래프 기반 모델 설계 인터페이스를 구현

5.1.2. 서버 측 애플리케이션 (Back-end)

- **FastAPI**를 활용하여 REST API를 구현 및 용자의 모델 설계 및 학습 요청을 처리

- **Celery + Redis**를 활용하여 모델 학습을 비동기적으로 실행하고, 작업 큐를 효율적으로 관리

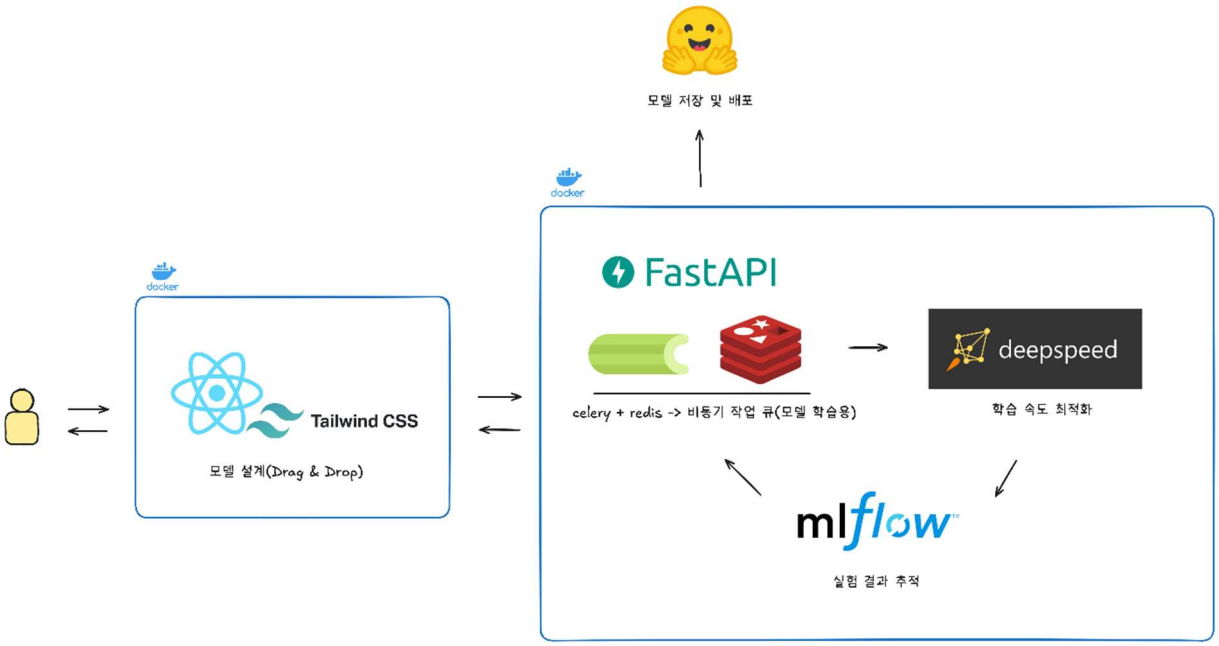
5.1.3. 모델 학습 및 실험 시스템

- **PyTorch** 및 **Hugging Face Accelerate**를 활용하여 모델을 효과적으로 학습
- **DeepSpeed**를 적용하여 학습 속도 최적화 및 메모리 효율성 개선
- **MLFlow**를 도입하여 학습 진행 과정 및 성능 변화를 추적하고, 실험 결과를 비교할 수 있도록 지원
- **Hugging Face Model Hub**와 연동하여 모델을 공유하고 배포할 수 있도록 지원

5.1.4. 배포

- **Docker**를 활용하여 대시보드와 서버를 컨테이너화하여 로컬에서 사용자 본인의 GPU로 사용할 수 있게 함

5.2 전체구조도



5.3 시스템 흐름

- **사용자는 웹 브라우저를 통해 React 기반의 웹 페이지에 접속**

    - 웹 프론트엔드는 Vite로 번들링되어 사용자에게 직관적인 드래그 & 드롭 인터페이스를 제공
    - 모델 구성을 드래그 & 드롭 방식으로 설정하고, 하이퍼파라미터 지정
- **사용자의 요청은 NGINX를 통해 로드 밸런싱되어 Fastapi 백엔드 서버로 전달**
    - 백엔드는 사용자가 설정한 모델 아키텍처 정보(트랜스포머 블록, 레이어 수, 하이퍼파라미터 등)를 수신
- **백엔드 서버는 로컬 환경에서 모델 학습 로직을 직접 실행**
    - 사용자가 설정한 모델 아키텍처 정보(트랜스포머 블록, 레이어 수, 하이퍼파라미터 등)를 기반으로 PyTorch 모델 구성
    - 구성된 모델은 로컬 GPU 자원을 사용하여 학습을 수행
- **학습 진행 정보 및 성능 지표는 MLflow를 통해 기록 및 시각화**
    - 학습 과정에서의 손실값, 정확도, 파라미터, 메트릭 등은 MLflow Tracking API를 통해 자동 기록되며, 사용자는 웹 UI를 통해 실험별 결과 비교 및 시각화를 확인할 수 있음
    - 학습 시각화는 MLflow에 통합
- **학습 완료 후, 사용자 로컬 브라우저에서 학습 결과를 확인**
    - 결과 확인은 FastAPI에서 직접 제공하는 간단한 상태 조회 API 및 MLflow 웹 대시보드를 통해 수행
    - 모델 성능 비교, 반복 실험, 구성 수정은 사용자 주도로 계속 진행 가능하며, 별도의 추론 결과 전송 API는 제공하지 않음

6. 개발환경 및 사용기술

6.1. 개발 언어

- **Frontend:** HTML, CSS, TypeScript

- **Backend:** Python (FastAPI)
- **AI Server:** Python (PyTorch, Transformers)

6.2. 개발 도구

- **Frontend:** Vite, React, Tailwind CSS, TypeScript
- **Backend:** FastAPI, PostgreSQL
- **Infra:** Docker, Redis
- **AI Server:** FastAPI
- **LLM/SLM:** Hugging Face Transformers, Sentence Transformers
- **시각화:** React Flow (Drag & Drop 기반 시각적 워크플로우)

6.3. 사용 기술

6.3.1. DeepSpeed

DeepSpeed는 Microsoft에서 개발한 PyTorch 기반의 오픈소스 라이브러리로, 대규모 모델 학습과 추론을 최적화하는 기술이다. 특히 GPU 메모리 사용량을 줄이고, 학습 속도를 향상시키는 기능을 제공한다.

또한, **ZeRO(Zero Redundancy Optimizer)** 기술을 통해 모델을 여러 GPU에 분산하고, 중복된 메모리 사용을 제거하여 메모리 효율성을 극대화한다. 이 과정에서 FP16(반정밀도 훈련) 과 Offloading(메모리 전환) 기법을 활용해 학습 성능을 최적화한다.

이에 더하여, **Gradient Accumulation** 기능을 통해 작은 배치 크기에서도 모델 성능을 유지할 수 있으며, 학습 속도를 높이면서도 비용을 절감할 수 있다.

본 프로젝트에서는 소형 언어 모델(SLM) 학습 시 GPU 메모리 최적화와 성능 향상을 위해 DeepSpeed를 적용하며, 이를 통해 하드웨어 자원 제한 환경에서도 원활한 모델 학습과 추론이 가능하도록 한다.

6.3.2. MLFlow

MLFlow는 모델 학습과 실험 과정을 추적하고 관리하는 오픈소스 플랫폼이다. 모델, 하이퍼파라미터, 성능 지표, 로그 등을 기록하고 시각화하는 기능을 제공한다.

이를 통해 Experiment Tracking, Model Registry, Artifacts 저장, Web UI 제공 등 주요 기

능을 통해 모델 학습의 효율성을 높이고, 버전을 체계적으로 관리할 수 있다.

본 프로젝트에서는 SLM 학습 과정과 성능 분석을 기록하고, 실험 간 성능을 비교하는 데 MLFlow를 활용한다. 이를 통해 가장 최적화된 모델을 선택하고 배포할 수 있도록 지원한다.

6.3.3. Celery

Celery는 비동기 작업을 처리하는 분산 태스크 큐 시스템이다. FastAPI와 함께 사용하여 시간이 오래 걸리는 작업을 백그라운드에서 처리하고, 작업 완료 상태를 실시간으로 추적할 수 있다.

이를 통해 **Redis**와 함께 메시지 브로커 역할을 하며, 학습, 추론, 데이터 처리와 같은 무거운 작업을 비동기적으로 실행하고 성능을 최적화한다.

본 프로젝트에서는 SLM 학습과 추론을 실시간으로 처리하고, 작업 상태를 실시간으로 모니터링하기 위해 Celery를 도입한다. 이를 통해 사용자가 플랫폼을 원활하게 이용하면서 모델 성능을 분석하고 최적화할 수 있도록 지원한다.

7. 일정 및 역할 분담

7.1 개발일정

7.2 역할 분담

김명석	-프론트엔드 개발 (React)
정지윤	-ML -AI 학습을 위한 데이터 수집
염현석	-백엔드 개발(FastAPI) -데이터 베이스 구축

8. 보고 시점까지의 과제 내용 및 중간 결과

8.1 완성된 기능

8.1.1 프론트엔드

- **웹페이지 전반적인 UI**
    - Canvas 메인 화면
    - Dataset 선택 및 모델 전송 화면
- **GPT-2 구현을 위한 Node 및 Cofiguration**
    - Embedding Layer 및 Trasnformer Block, Attention Layer 등
    - Epochs, Batch Size 등 하이퍼파라미터 및 Context Size, Number of Heads 등 모델 구성 파라미터

8.1.1 백엔드

- **모델 학습 요청 및 상태 조회 API**
- **비동기 학습 처리 구조**
    - Celery + Redis를 기반으로 모델 학습 요청을 백그라운드에서 실행
    - 작업 ID를 통해 학습 상태 추적 및 결과 확인 가능
- **입력 데이터 유효성 검사 (Validation)**

- Pydantic을 이용한 TrainRequest 모델 정의를 통해 config 및 layer_stack의 형식 검증
- 필수 필드 누락, 잘못된 타입 등 클라이언트 측 입력 오류에 대한 사전 차단 처리

- **레이어 구성 기반 GPT 모델 동적 생성**
  - token_embedding, positional_embedding, transformer_block 등 주요 레이어 구성 정의
  - JSON 형식의 layer_stack을 기반으로 PyTorch로 모델을 동적으로 생성
  - dummy input에 대한 forward pass로 모델 구성 유효성 확인

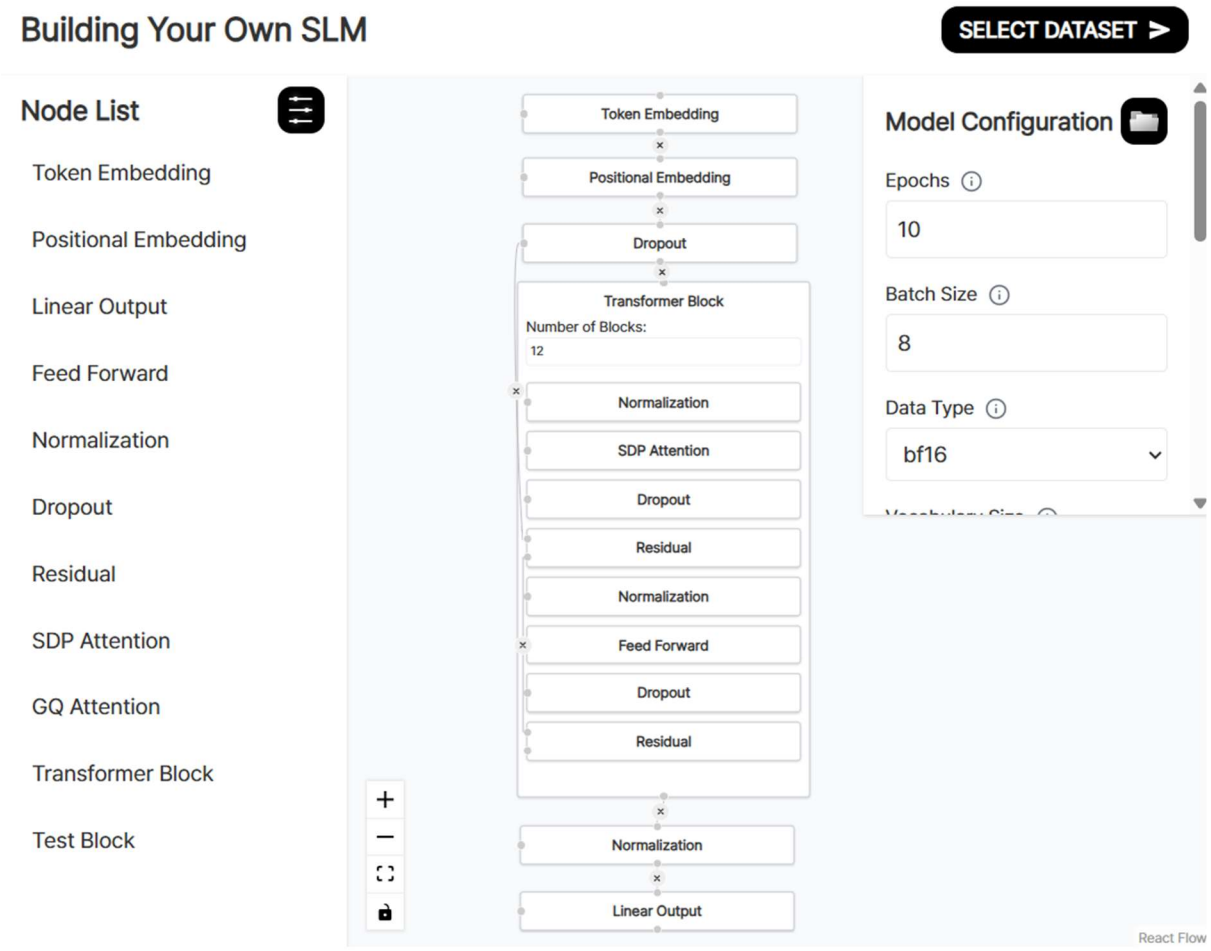
8.1.1 AI

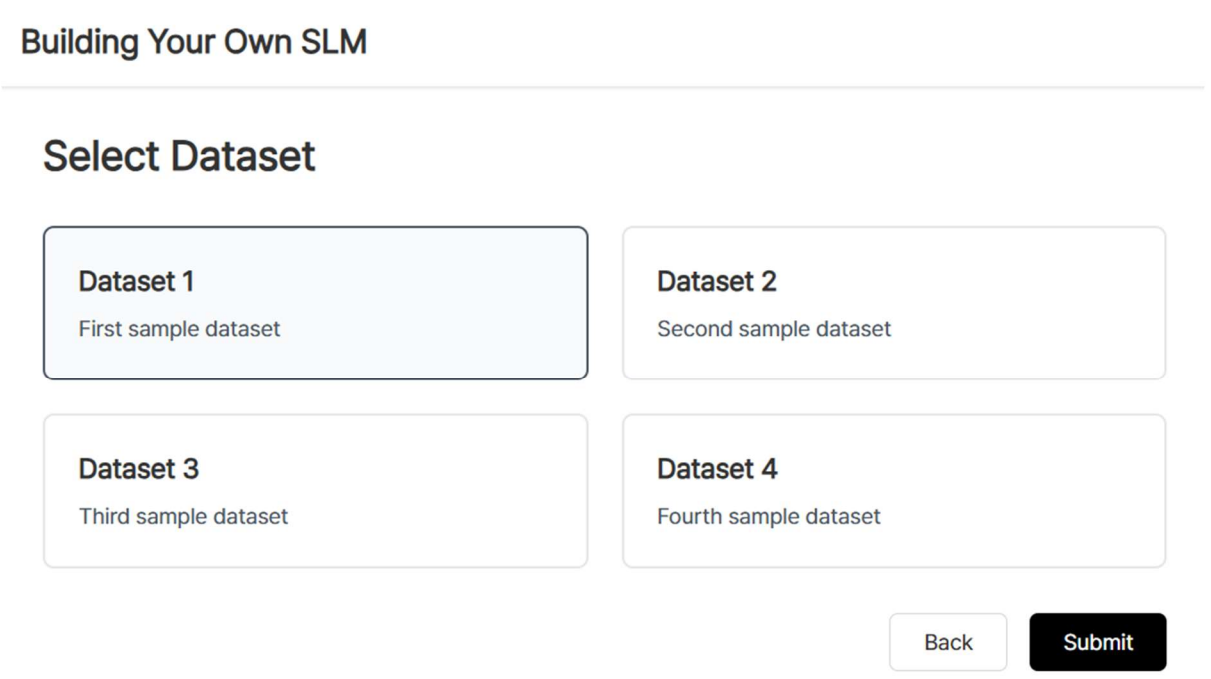
- **모델 학습 파이프라인 구현**
- **DeepSpeed를 활용하여 학습 속도 개선 기능 추가**
- **MLFlow를 적용하여 실험 추적 기능 구현**
- **Small Language model에 쓰이는 데이터셋 리서치**

8.2 추후 개발 기능

- Llama 2와 Llama 3 구현을 위한 노드 및 백엔드 PyTorch 구현
- MLFlow 고도화 및 시각화
- 학습 후 실습 기능 구현
- 도커 패키징

8.3 중간 보고 시점까지의 결과물 프리뷰


Building Your Own SLM
SELECT DATASET
Node List
Token Embedding
Positional Embedding
Linear Output
Feed Forward
Normalization
Dropout
Residual
SDP Attention
GQ Attention
Transformer Block
Test Block
Token Embedding
Positional Embedding
Dropout
Transformer Block
Number of Blocks:
12
Normalization
SDP Attention
Dropout
Residual
Normalization
Feed Forward
Dropout
Residual
Normalization
Linear Output
React Flow
Model Configuration
Epochs
10
Batch Size
8
Data Type
bf16


Building Your Own SLM
Select Dataset
Dataset 1
First sample dataset
Dataset 2
Second sample dataset
Dataset 3
Third sample dataset
Dataset 4
Fourth sample dataset
Back
Submit