생성형 AI 프로젝트 계획서

1. 팀의 목표 정리

단순히 모델을 만드는 것을 넘어, 한국어 특화 미니 GPT 를 구현하고 실험적 성능 검증까지 하는 것을 목표로 한다.

이후 실사용(챗봇, 번역, 요약 등)으로 확장 가능성을 고려한다.

2. 프로젝트 개요

- 프로젝트명: 한국어 미니 GPT 개발

- 목표:

 - 한국어 특화된 소형 GPT 모델을 개발

 - 대화, 요약, 번역 등 기본적인 자연어 생성(NLG) 기능을 실현

 - 연구/실험 환경에서 활용 가능하며, 추후 확장성을 고려

3. 팀 구조 제안

- 데이터팀: 말뭉치 수집·전처리 (웹 크롤링, 토큰화, 클리닝, 학습 데이터셋 준비).

- 모델팀: Transformer 구조 설계 및 구현 (TensorFlow/PyTorch 기반).

- 학습팀: 학습 환경 구성 (GPU 세팅, 분산 학습, 실험 관리).

- 응용팀: 학습된 모델을 API·웹 서비스로 연결 (예: Flask, FastAPI, React 등).

- 평가팀: 모델 성능 평가 (BLEU, Perplexity, 사용자 평가 등).

3. 리더로서의 역할

- 프로젝트 로드맵 수립.

- 각 팀에 가이드 문서와 교육자료 제공.

- 주간 미팅을 통해 진행 상황 점검 및 병목 해결.

- 연구 방향과 결과를 정리해 보고서 작성.

4. 개발 범위

4-1. 데이터 수집 및 전처리

  - 한국어 말뭉치 확보 (뉴스, 위키, SNS 등 공개 데이터)

  - 클리닝, 불필요한 특수문자 제거

  - 토크나이저 적용 (SentencePiece, BPE)

4-2. 모델 설계 및 학습

  - Decoder-only Transformer 구조 설계

  - 모델 크기: 50M ~ 150M 파라미터 (소형 GPT)

  - 학습 프레임워크: TensorFlow 또는 PyTorch

  - GPU 분산 학습 환경 구성

4-3. 응용 및 배포

  - REST API 구축 (Flask 또는 FastAPI)

  - 웹 UI(React 기반)에서 질의응답/대화 테스트

- 모델 추론 속도 최적화

4-4. 평가 및 고도화

- 평가 지표: Perplexity, BLEU, Rouge

- 사용자 테스트: 간단한 QA, 번역, 요약 성능 확인

- Fine-tuning 으로 특정 도메인 확장 가능성 검토

5, 모델 설계 및 학습 내용

- 모델 구조: Decoder-only Transformer 구조를 처음부터 직접 코드로 구현

  셀프-어텐션 메커니즘, 포지셔널 인코딩, 레이어 정규화, 피드포워드 신경망 등 모든 구성 요소를 수작업으로 작성.

- 데이터 전처리: 대규모 텍스트 데이터를 토큰화하고, 모델이 이해할 수 있는 숫자 형태로 변환하는 전처리 파이프라인을 직접 구축.

  여기에는 BPE(Byte-Pair Encoding)나 SentencePiece와 같은 토크나이저를 사용해야 하는데, 이 또한 직접 구현하거나 외부 라이브러리를 사용

- 훈련 루프: 모델을 훈련시키기 위한 훈련 루프(training loop)를 직접 작성

  옵티마이저 설정, 손실 함수 정의, 그래디언트 계산, 모델 파라미터 업데이트 등 단계가 포함.

- 추론 및 디코딩: 훈련된 모델을 사용하여 새로운 텍스트를 생성하는 추론(inference) 과정도 직접 구현.

특히, 대화형 GPT 모델의 경우 “그리디 서치(greedy search) 또는 빔 서치(beam search)”와 같은 텍스트 디코딩 전략을 구현

6. 팀 구조 보강안

6-1, 데이터팀

•기존 역할: 말뭉치 수집·전처리

•보강 역할: 데이터베이스(DB) 구축 및 관리 (수집 데이터 저장, 버전 관리, 검색 기능 제공)

인프라팀 (신규)

• 역할: GPU 클러스터 및 클라우드 자원 관리, 데이터베이스 서버 운영, 실험 환경 자동화

학습팀과 협력하여 안정적인 학습 및 서비스 환경 제공

6-2. 서비스 단계 고려

•사용자 대화 로그 및 피드백 데이터를 데이터베이스에 저장하여, 향후 모델 파인튜닝 및 성능 개선에 활용

•학습 결과 및 메트릭 기록 (DB 기반 실험 관리: MLflow, W&B 등 활용 가능)

7. 기대 효과

- 한국어 특화 소형 GPT 모델 확보

- 오픈소스 수준의 실험 환경 구축

- 대화/요약/번역 등 다양한 NLG 응용 가능

- 추후 대규모 모델 확장 및 서비스화 가능

8. 향후 발전 방향

- 대규모 GPU 클러스터 기반 학습 확장

- 도메인 특화 Fine-tuning (법률, 의료, 쇼핑 등)

- 멀티모달(텍스트+이미지) 모델 확장

구분	**EXAONE (LG AI 연구원)**	**LLaMA (Meta, 오픈소스)**
개발 주체	LG AI Research (한국)	Meta (페이스북 모회사)
주요 언어	**다국어(특히 한국어 강화)**	영어 중심, 다국어 일부 지원
모델 크기	수십억~수천억 파라미터 (1.0 → 2.0 → 3.0 진화)	LLaMA 1: 7B~65B LLaMA 2: 7B/13B/70B LLaMA 3: 더 확장
공개 여부	**상용 모델** (API 제공 중심, 전체 가중치 공개 X)	**오픈소스** (가중치 공개, 연구/개발자 활용 자유로움)
아키텍처	Transformer 기반, GPT 계열 (Decoder-only)	Transformer 기반, GPT 계열 (Decoder-only)
학습 데이터	**국내외 뉴스, 논문, 특허, 한국어 중심 대규모 데이터**	**주로 영어 중심 웹 크롤링 + 책, 논문, 코드 등**
활용	기업 특화 (R&D, 문서 요약, 한국어 대화)	범용 (연구, 오픈소스 커뮤니티, 파인튜닝)
강점	한국어/다국어 이해도 매우 높음, 산업 맞춤	오픈소스 자유도, 글로벌 연구 생태계
약점	모델/가중치 비공개, 연구자 제약	한국어 성능은 EXAONE 대비 부족할 수 있음

전체 코드

```
import tensorflow as tf
from tensorflow import keras
from keras import layers
import numpy as np


# 1. 하이퍼파라미터 정의
# 아래 값들은 필요에 따라 조정하세요.
VOCAB_SIZE = 16000
MAX_LEN = 1024
N_LAYERS = 8
D_MODEL = 512
N_HEADS = 8
D_FF = 2048
DROPOUT_RATE = 0.1
BATCH_SIZE = 32
EPOCHS = 3
LEARNING_RATE = 3e-4


# 2. 데이터 파이프라인
# 이 부분은 사용자의 데이터와 파일 경로에 맞게 직접 작성해야 합니다.
# 예시로 더미 데이터를 생성합니다.
def create_dummy_dataset(num_samples=1000):
    # 실제로는 대규모 텍스트 데이터를 읽어와서 토큰화해야 합니다.
```

```
    # 예시: (입력 시퀀스, 타겟 시퀀스) 쌍 생성
    dummy_input_ids = np.random.randint(0, VOCAB_SIZE, (num_samples, MAX_LEN), dtype=np.int32)
    dummy_target_ids = np.random.randint(0, VOCAB_SIZE, (num_samples, MAX_LEN), dtype=np.int32)

    # tf.data.Dataset 으로 변환
    dataset = tf.data.Dataset.from_tensor_slices((dummy_input_ids, dummy_target_ids))
    dataset = dataset.shuffle(buffer_size=1000).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
    return dataset

# 3. 모델 아키텍처 (제공된 문서의 핵심 구조)
class CausalSelfAttention(layers.Layer):
    def __init__(self, d_model, n_heads, dropout=0.0):
        super().__init__()
        assert d_model % n_heads == 0
        self.nh = n_heads
        self.hs = d_model // n_heads
        self.qkv = layers.Dense(3 * d_model, use_bias=False)
        self.proj = layers.Dense(d_model, use_bias=False)
        self.dropout = layers.Dropout(dropout)

    def call(self, x, training=False):
        B, T, C = tf.unstack(tf.shape(x))
        qkv = self.qkv(x)
```

```
        q, k, v = tf.split(qkv, 3, axis=-1)
        q = tf.reshape(q, [B, T, self.nh, self.hs])
        q = tf.transpose(q, [0, 2, 1, 3])
        k = tf.reshape(k, [B, T, self.nh, self.hs])
        k = tf.transpose(k, [0, 2, 1, 3])
        v = tf.reshape(v, [B, T, self.nh, self.hs])
        v = tf.transpose(v, [0, 2, 1, 3])

        att = tf.matmul(q, k, transpose_b=True) / tf.sqrt(tf.cast(self.hs, tf.float32))
        mask = tf.linalg.band_part(tf.ones((T, T)), -1, 0)
        mask = tf.reshape(mask, [1, 1, T, T])
        att = att + (1.0 - mask) * -1e9

        att = tf.nn.softmax(att, axis=-1)
        att = self.dropout(att, training=training)
        y = tf.matmul(att, v)
        y = tf.transpose(y, [0, 2, 1, 3])
        y = tf.reshape(y, [B, T, self.nh * self.hs])
        return self.proj(y)

class TransformerBlock(layers.Layer):
    def __init__(self, d_model, n_heads, d_ff, dropout=0.0):
        super().__init__()
        self.ln1 = layers.LayerNormalization(epsilon=1e-5)
        self.att = CausalSelfAttention(d_model, n_heads, dropout)
        self.dropout = layers.Dropout(dropout)
```

```
        self.ln2 = layers.LayerNormalization(epsilon=1e-5)
        self.ffn = keras.Sequential([
            layers.Dense(d_ff, activation="gelu"),
            layers.Dense(d_model),
            layers.Dropout(dropout),
        ])

    def call(self, x, training=False):
        x = x + self.dropout(self.att(self.ln1(x), training=training), training=training)
        x = x + self.ffn(self.ln2(x), training=training)
        return x

def build_decoder_only(vocab_size, max_len, n_layers, d_model, n_heads, d_ff, dropout):
    inp = layers.Input(shape=(max_len,), dtype=tf.int32)
    tok_emb = layers.Embedding(vocab_size, d_model)
    pos_emb = layers.Embedding(max_len, d_model)
    positions = tf.range(start=0, limit=max_len, delta=1)
    x = tok_emb(inp) + pos_emb(positions)

    for _ in range(n_layers):
        x = TransformerBlock(d_model, n_heads, d_ff, dropout)(x)

    x = layers.LayerNormalization(epsilon=1e-5)(x)
    logits = layers.Dense(vocab_size, use_bias=False)(x)
    return keras.Model(inp, logits)
```

```
# 4. 모델 컴파일 및 학습

# tf.keras.mixed_precision.set_global_policy("mixed_float16") # GPU 사용 시 혼합 정밀도 활성화

model = build_decoder_only(VOCAB_SIZE, MAX_LEN, N_LAYERS, D_MODEL, N_HEADS, D_FF, DROPOUT_RATE)


loss_fn = keras.losses.SparseCategoricalCrossentropy(from_logits=True)

optimizer = keras.optimizers.AdamW(learning_rate=LEARNING_RATE, weight_decay=0.01)


model.compile(optimizer=optimizer, loss=loss_fn)


# 데이터셋 로드

train_ds = create_dummy_dataset()

# validation_ds = create_dummy_dataset() # 실제로는 검증 데이터셋 사용


print("모델 학습을 시작합니다...")

model.fit(train_ds, epochs=EPOCHS)


# 5. 생성(Inference) 함수 (문서 내용 + 실제 사용 가능하도록 보강)

def generate(model, tokenizer, prompt_text, max_new_tokens=100, top_k=50, temperature=0.8):

    # 텍스트 프롬프트를 ID 로 변환 (SentencePiece 토크나이저 필요)

    # 실제로는 tokenizer.encode(prompt_text)와 같은 함수 사용

    # 예시: 더미 프롬프트 ID
```

```
prompt_ids = np.random.randint(0, VOCAB_SIZE, (1, 10))

x = tf.convert_to_tensor(prompt_ids, dtype=tf.int32)

for _ in range(max_new_tokens):
    # 텐서플로우 모델은 고정 길이 입력을 받으므로, max_len 보다 길어지면 슬라이싱
    # x_slice = x[:, -MAX_LEN:]
    logits = model(x)[:, -1, :] / temperature

    # Top-k 필터링 (간략화)
    values, _ = tf.math.top_k(logits, k=top_k)
    min_val = values[:, -1, None]
    logits = tf.where(logits < min_val, tf.fill(tf.shape(logits), -1e9), logits)

    # 샘플링 및 다음 토큰 예측
    next_id = tf.random.categorical(logits, num_samples=1)
    x = tf.concat([x, next_id], axis=1)

# ID 를 텍스트로 변환 (tokenizer.decode 필요)
# 실제로는 tokenizer.decode(x.numpy()[0]) 사용
generated_text = "생성된 텍스트입니다."

return generated_text
```

```
# 6. 모델 생성 예시

# SentencePiece 토크나이저를 학습/로드 후 사용

# sp_tokenizer = ...

# print(generate(model, sp_tokenizer, "미니 GPT 를 만들려면", max_new_tokens=50))
```