석사학위논문

Secu sLLM: 데스크톱 환경용 시큐어 코딩 특화 sLLM

2025년

한 성 대 학 교 대 학 원

컴 퓨 터 공 학 과

컴 퓨 터 공 학 전 공

이 찬 우

석 사 학 위 논 문
지도교수 허준영

Secu sLLM: 데스크톱 환경용 시큐어 코딩 특화 sLLM

Secu sLLM: An sLLM Optimized for Secure Coding in Desktop Computing Environments

2024년 12월 일

한 성 대 학 교 대 학 원

컴 퓨 터 공 학 과

컴 퓨 터 공 학 전 공

이 찬 우

석 사 학 위 논 문
지도교수 허준영

Secu sLLM: 데스크톱 환경용 시큐어 코딩 특화 sLLM

Secu sLLM: An sLLM Optimized for Secure Coding in Desktop Computing Environments

위 논문을 공학 석사학위 논문으로 제출함

2024년 12월 일

한 성 대 학 교 대 학 원

컴 퓨 터 공 학 과

컴 퓨 터 공 학 전 공

이 찬 우

이찬우의 공학 석사학위 논문을 인준함

2024년 12월 일

심사위원장 정 진 만 (인)

심 사 위 원 민 홍 (인)

심 사 위 원 허 준 영 (인)

국 문 초 록

Secu sLLM: 데스크톱 환경용 시큐어 코딩 특화 sLLM

한 성 대 학 교 대 학 원
컴 퓨 터 공 학 과
컴 퓨 터 공 학 전 공
이 찬 우

컴퓨팅 환경과 생성형 AI의 발전으로 인해 모든 산업에서 IT의 중요성이 필수적으로 요구되고 있다. 이에 따라 모든 기업은 IT 관련 부서를 조직하여 운영하고 있으며, 국가에서는 "SW 보안 개발 가이드"를 준수하여 개발할 것을 의무화하고 있다. 그러나 서비스 출시를 서두르는 신생기업이나 IT 개발 인력이 부족한 기업들은 해당 가이드의 시큐어 코딩 지침에 맞춰 개발하지 못하는 사례가 많다. 이러한 문제에 대한 대응책으로 생성형 AI를 활용한 시큐어 코딩 개발 방법이 제시되고 있다. 하지만 이 방법 역시 두 가지 문제점을 가지고 있다. 첫째, 생성형 AI의 특성상 새로운 데이터를 끊임없이 학습하므로 기업의 기밀 코드 및 민감한 정보가 학습 데이터로 사용될 위험성이 존재한다. 둘째, 사용량에 따라 비용이 발생하는 금전적인 문제가 있다. 이에 대한 또 다른 대응책으로 오픈소스 기반의 LLM 모델을 로컬 환경에서 구축하여 활용하는 방법이 있지만, 이 역시 두 가지 문제점이 있다. 첫째, 한국어 성능이 우수한 LLM은 일반적인 사무 환경의 컴퓨팅 사양으로는 원활하게 구

동하기 어려운 고사양을 요구한다. 둘째, 해당 모델이 시큐어 코딩에 특화된 것이 아니라 범용적으로 학습된 모델이므로 개발 보안에서 원하는 결과를 얻지 못할 수 있다. 따라서 본 논문에서는 일반적인 컴퓨팅 환경에서도 원활히 구동될 수 있는 sLLM을 활용하고자 한다. sLLM은 앞서 언급한 상용 LLM 서비스의 학습 데이터 유출 문제와 고사양 컴퓨팅 환경에서의 모델 구축 제약을 해결할 수 있어, 시큐어 코딩에 특화된 모델로 설계하여 활용할 수 있다. 제안하는 모델은 한국인터넷진흥원이 제시한 "언어별 시큐어 코딩 가이드"를 기반으로 검증된 단체들의 시큐어 코딩 자료를 학습하여, 소스 코드의 보안 약점을 검사하는 데 활용하고자 한다.

【주요어】 LLM, sLLM, 시큐어 코딩, 인공지능 보안, 데스크톱 환경

목 차

표 목 차

그 림 목 차

제 1 장 서론

제 1 절 연구의 배경

최근 다양한 산업 분야에서 정보기술(IT)의 중요성이 더욱 강조되고 있으며, 컴퓨팅 인프라와 생성형 인공지능 기술의 발전은 이를 가속화하고 있다(장종원, 2024). 이러한 환경 변화에 대응하기 위해 많은 기업이 전문적인 IT 부서를 운영하고 있으며, 국가 정책 역시 SW 보안 약점 제거·조치 성과에 따라 2012년 SW 개발 보안 제도(한국인터넷진흥원(KISA), 2021)가 도입하여 점진적으로 의무화를 시행하였다. 그러나 신생 기업이나 보안 전문 인력 확보가 어려운 조직에서는 서비스 출시 속도를 우선시하면서 SW 보안 개발 가이드 지침을 충분히 반영하지 못하는 사례가 종종 발생하고 있다. 이러한 경우, 알려진 취약점에 대한 적절한 조치가 이루어지지 않아 웹 애플리케이션을 통한 사용자 정보 유출과 같은 심각한 침해 사고로 이어질 위험이 크다. 더 나아가 소프트웨어 개발 초기 단계에서 보안을 고려하지 않으면 보안 취약점을 수정하는 비용이 기하급수적으로 증가하고, 이는 대규모 보안 사고로 이어질 가능성이 커진다(한국인터넷진흥원(KISA), 2023).

이를 해결하기 위한 대안으로 AI 기반의 시큐어 코딩 지원 방법론(김동연 외, 2023)이 제안되었다. 생성형 AI 기술 중 하나인 거대 언어 모델(Large Language Model, LLM)을 활용할 수 있는데, 현 LLM을 활용한 코딩 보조 서비스는 보안 취약점 가능성을 사전에 식별하고 보완책을 제시함으로써 개발 효율성과 보완성을 동시에 제고할 수 있다(Motlagh et al., 2024). 하지만 이러한 접근은 새로운 데이터가 계속 학습되는 LLM의 특성으로 인해 기업 내부 기밀 코드나 민감한 정보가 모델 학습 자료로 활용될 위험성이 있으며, 사용량에 따른 높은 비용도 고려해야 한다. 이러한 문제를 극복하기 위한 대안으로 오픈소스 LLM을 로컬 환경에 구축하는 방법이 논의 되고 있으나, 이 역시 현실적인 한계를 가지고 있다. 우선 한국어 처리가 우수한 LLM 대부분은 고사양의 하드웨어를 요구하여 일반적인 업무 환경에서는 원활한 구동이

쉽지 않다. 또한 범용적으로 학습된 모델은 보안 특화 지침을 충분히 반영하기 어렵기 때문에, 개발 현장에서 기대하는 시큐어 코딩 지원 능력을 충분히 확보하기 어렵다.

본 논문에서는 이러한 제약을 동시에 해소하기 위해 대형 언어 모델보다 파라미터(Parameter) 수가 적으며, 소형 언어 모델(small Language Model, sLM)보다 파라미터의 수가 큰 3B ~ 70B 사이의 sLLM(small Large Language Model)을 활용하는 전략을 제안한다. sLLM은 거대 언어 모델 대비 경량화된 구조를 통해 일반적인 컴퓨팅 환경에서도 동작할 수 있어, 자체 구축을 통한 데이터 유출 위험을 줄일 수 있다. 나아가 한국인터넷진흥원(KISA)이 제시한 “언어별 시큐어 코딩 가이드” 등 공신력 있는 자료를 학습함으로써, 시큐어 코딩에 특화된 모델을 구축할 수 있다. 이를 통해 본 연구는 로컬 환경에서 운영할 수 있는 sLLM 기반 시큐어 코딩 모델을 제안하고, 이를 통해 실무적인 개발 보안 강화의 새로운 가능성을 모색하고자 한다.

제 2 절 논문의 구성

본 논문은 총 7장으로 구성되어 있다. 제1장 서론에서는 연구의 배경과 필요성을 제시하며, 기존 문제점을 분석하고 이를 해결하기 위한 sLLM 기반 시큐어 코딩 모델의 개발 방향을 간략히 소개하였다.

제2장 관련 연구에서는 LLM의 기본 개념과 트랜스포머(Transformer) 모델의 구조를 설명하며, sLLM 모델을 분석한다. 또한, LLM이 사이버 보안 분야에서 활용된 사례를 검토하여 본 연구의 차별성과 방향성을 논의한다.

제3장 모델 선정에서는 본 연구에서 사용될 sLLM 모델을 선정하기 위한 기준을 제시하고, 선정된 모델인 LGAI-EXAONE의 아키텍처와 주요 매개변수를 분석하여 적합성을 평가한다.

제4장 데이터세트 선정 및 구축에서는 학습 데이터세트 선정 기준을 설명하고, 한국인터넷진흥원(KISA)이 제시한 “언어별 시큐어 코딩 가이드”와 검증된 단체들의 자료를 바탕으로 수집된 데이터세트와 전처리 과정을 상세히 설명한다.

제5장 제안하는 sLLM 모델에서는 Secu sLLM 모델의 개요와 실험 환경을 기술하며, 학습 방법 및 전략을 구체적으로 제시한다.

제6장 실험 결과 및 성능 평가에서는 실험 결과를 분석하고, 성능 평가 지표를 통해 제안한 모델의 효율성과 정확성을 검토한다.

마지막으로, 제7장 결론에서는 본 연구의 결과와 향후 도전 과제를 제안한다. 이와 같은 논문의 구성을 통해 sLLM 기반 시큐어 코딩 가이드 모델의 설계 및 구현 과정을 체계적으로 제시하고, 실무적 적용 가능성을 논의하고자 한다.

제 2 장 관련 연구

제 1 절 LLM 개요

1) 트랜스포머(Transformor)

트랜스포머는 LLM의 핵심 아키텍처로 자리 잡은 혁신적인 모델이다. 이 아키텍처는 2017년 Vaswani et al.의 논문 “Attention is All You Need”에서 처음 제안되었으며, 기존의 순환 신경망(Recurrent Neural Network, RNN)과 합성곱 신경망(Convolutional Neural Network, CNN)의 한계를 극복하며 자연어 처리(Natural Language Processing, NLP)와 생성형 AI의 새로운 가능성을 열었다(Vaswani et al., 2017). 트랜스포머는 데이터 시퀀스의 병렬 처리를 지원하며, Self-Attention 메커니즘을 통해 입력 데이터 내 모든 토큰 간의 관계를 효율적으로 학습한다. 트랜스포머의 주요 구성 요소는 아래의 [표 2-1]와 같다.

[표 2-1] 트랜스포머의 주요 구성 요소

구성 요소	설명
Self-Attention	쿼리(Query), 키(Key), 값(Value) 벡터의 내적 연산을 통해 데이터 간 관계를 계산하고 문맥 정보를 학습한다.
Multi-Head Attention	여러 어텐션 헤드를 병렬로 구성하여 다양한 문맥 관계를 처리하며, 모델의 표현력을 높인다.
Positional Encoding	사인(Sin)과 코사인(Cosine) 함수를 사용해 순차적 데이터를 모델에 반영한다.

이 아키텍처는 NLP 및 기계 번역 작업에서 뛰어난 성능을 보여주었으며,

특히 WMT 2014 영어-독일어 번역 작업에서 당시 최고 성능을 기록하며 기존 모델을 능가하는 결과를 나타냈다. 병렬 처리를 통해 학습 속도도 대폭 향상되어, 8개의 NVIDIA P100 GPU를 사용하여 단 12시간 만에 최고의 성능을 달성했다.

트랜스포머는 2024년 기준 인기 있는 다양한 LLM 모델의 기초가 되었으며, 대표적인 모델로는 다음과 같다.

- OpenAI의 GPT-4o: OpenAI가 2024년에 발표한 최신 멀티모달 언어 모델로, 텍스트, 이미지, 오디오 입력을 처리하고 생성할 수 있다. GPT-4o는 GPT-4와 유사한 수준의 지능을 제공하면서도 더 빠른 응답 속도와 향상된 멀티모달 처리 능력을 보유하고 있다. 특히, 50개 이상의 언어를 지원하며 음성 인식 및 번역 분야에서도 새로운 기준을 제시하였다.

- Google의 BERT (Bidirectional Encoder Representations from Transformers): 문장 내 단어 간 관계를 양방향으로 학습하며, 문장 분류 및 질의응답 작업에서 높은 성능을 보여준다.

- Google의 T5 (Text-To-Text Transfer Transformer): 모든 NLP 작업을 텍스트 입력과 텍스트 출력의 변환 문제로 정의하며, 번역, 요약, 문장 생성 등 다목적 작업을 처리한다.

- Meta의 LLaMA (Large Language Model Meta AI): 경량화된 트랜스포머 기반 모델로, 학계와 연구 커뮤니티에 공개되어 연구 및 실무적 활용이 활발히 이루어지고 있다.
- BigScience의 BLOOM: 다국어 처리가 가능한 오픈 모델로, 여러 언어를 지원하며 다양한 연구 기관에서 파생형 모델을 제작해 공유하고 있어 언어 작업에서 강력한 성능을 제공한다.

이러한 모델들은 트랜스포머 구조를 기반으로 개발되어 자연어 처리와 생성형 AI 응용에서 중요한 성과를 보인다.

2) sLLM(small Large Language Model)

sLLM은 대규모 언어 모델과 소규모 언어 모델 사이에 위치하는 중간 크기의 언어 모델로 정의할 수 있다. 상용화된 LLM이 수천억 개에서 수조 개의 매개변수를 포함하는 거대한 규모를 자랑하는 반면, sLM은 수백만 개의 매개변수를 가지며 경량화된 환경에서 구동되도록 설계되었다. sLLM은 이러한 두 모델 간의 중간 지점을 차지하며, 수십억 개에서 수백억 개 수준의 매개변수를 통해 양쪽의 장점을 모두 수용하는 모델이다.

트랜스포머 기반의 오픈소스 LLM은 주로 허깅페이스(Hugging Face)[1] 플랫폼을 통해 공개되어 있으며, 연구자와 개발자들에게 폭넓은 선택지를 제공한다. 허깅페이스는 인공지능(AI)과 자연어 처리(NLP) 분야에서 인기 있는 커뮤니티 중심의 오픈소스 플랫폼으로, 다양한 모델, 데이터세트, 그리고 API를 지원한다.

아래의 [그림 2-1]은 허깅페이스에서 제공하는 Meta의 Llama 레포지토리를 캡처한 화면으로, 사용자 다운로드 수를 기준으로 정렬한 결과이다. 이를 통해 공개된 모델 중 많은 사용자가 주로 활용하는 모델이 3B ~ 70B 크기의 sLLM 규모임을 확인할 수 있다. 이는 sLLM이 경량화된 구조로도 높은 활용성을 제공하며, 다양한 환경에서 실질적인 유용성을 입증하고 있음을 보여준다.

1) https://huggingface.co/

[그림 2-1] Meta의 Llama 허깅페이스 레포지토리 (출처: Hugging Face, 2024)

3) 사이버 보안에서의 LLM 활용 사례

LLM은 다양한 사이버 보안 분야에서 주목받고 있으며, 특히 도메인 특화 모델과 자동화된 분석 프레임워크를 통해 보안 위협을 탐지하고 대응하는 데 탁월한 성능을 보여준다. 이러한 기술은 보안 로그 분석, 다크 웹 데이터 처리, 악성코드 분석 등 다양한 분야에서 실질적인 효용성을 입증하고 있다. 아래는 국내 연구를 기반으로, LLM을 활용하여 사이버 보안에 적용한 두 가지 사례를 소개한다.

다크 웹에서의 도메인 특화 모델: DarkBERT 사례

다크 웹과 일반 웹(Surface Web)에서 사용되는 언어는 명확한 차이를 보인다. 다크 웹은 범죄 은어, 극단적인 어휘 등 구조적 다양성을 가진다. 이는 일반적인 언어 모델로는 충분히 분석하기 어려운 영역이다. 이러한 특성을 해결하기 위해 도메인 특화 언어 모델인 DarkBERT가 개발되었다(Jin et al., 2023).

DarkBERT는 다크 웹 데이터를 사전 학습(pretraining)하여 해당 영역에서 텍스트를 효과적으로 분석할 수 있도록 설계되었다. 학습 데이터는 다크

웹에서 수집된 텍스트를 필터링 및 정제하여 구축되었으며, 이는 다크 웹의 특유한 어휘와 구조를 모델이 효과적으로 학습할 수 있게 한다. DarkBERT는 기존의 일반 언어 모델 및 BERT 기반 모델과 비교하여 랜섬웨어 유출 사이트 탐지, 주목할 만한 포럼 스레드 식별 등에서 뛰어난 성능을 보여주었다.

DarkBERT의 평가 결과, 이 모델은 다크 웹 도메인 특화 모델로서 기존 언어 모델을 능가하는 성과를 입증하였으며, 다크 웹에 대한 연구와 보안 위협 분석에 있어 중요한 자원이 될 수 있음을 시사한다. 이러한 도메인 특화 모델은 보안 위협의 조기 탐지와 효과적인 대응 방안을 제시하는 데 핵심적인 역할을 할 것으로 기대된다.

악성코드 분석 자동화를 위한 LLM 활용: LangChain 활용 사례

악성코드의 복잡성과 다변화에 따라 이를 신속하고 정확하게 분석하기 위한 자동화 도구의 필요성이 커지고 있다. 이에 LangChain 프레임워크를 활용한 LLM 기반의 악성코드 자동 분석 프로세스가 제안되었다(도예진, 2024).

LangChain은 다양한 데이터 소스와 자연어 처리 모델을 연결하여, 악성코드 데이터를 기반으로 분석 과정을 자동화할 수 있는 프레임워크이다. 연구에서는 LangChain을 활용해 악성코드 데이터로부터 특정 패턴을 학습하고, 이를 분석 결과로 도출하여 보고서를 자동 생성하는 프로세스를 설계하였다. 이 프로세스는 분석가의 작업 부담을 줄이고, 반복적인 작업을 효율적으로 수행하는 데 기여한다.

특히, 연구는 LangChain이 악성코드 탐지에 필요한 데이터를 효과적으로 처리하고, 해당 데이터를 활용한 추가적인 규칙 생성 가능성을 보여주었다. 이는 향후 YARA와 같은 규칙 기반 시스템과의 연계를 통해 악성코드 탐지 및 분류에서 더욱 발전된 자동화 시스템을 구축할 가능성을 시사한다. 이 연구는 LLM이 사이버 보안 분야에서 악성코드 분석과 자동화된 대응 체계를 구축하는 데 필수적인 도구로 자리 잡을 수 있음을 실질적으로 보여주었다.

제 3 장 모델 선정

제 1 절 모델 선정 기준

본 연구에서는 시큐어 코딩 가이드 모델을 구축하기 위해 적합한 언어 모델을 선정하기 위한 기준을 다음과 같이 설정하였다.

첫째, 한국어 성능이 뛰어난 모델이어야 한다. 시큐어 코딩 가이드와 관련된 문서는 주로 한국어로 작성되며, 국내 개발자가 사용하는 소스 코드의 주석 또한 대부분 한국어로 작성되기 때문이다. 한국어 처리 능력이 우수한 모델은 시큐어 코딩 관련 정보를 보다 정확하게 이해하고, 의미 있는 분석 결과를 제공할 수 있다.

둘째, 코딩 능력이 우수한 모델이어야 한다. 이를 통해 소스 코드의 보안 약점을 효과적으로 식별하고, 적절한 수정안을 제시할 수 있어야 한다. 코딩 능력이 부족한 모델은 보안 취약점 검출 및 대응 과정에서 충분히 신뢰할 수 있는 결과를 제공하지 못할 가능성이 있다.

셋째, 모델의 크기가 적절해야 한다. 지나치게 작은 모델은 성능이 부족하여 정확하고 심도 있는 결과를 도출하기 어려우며, 반대로 지나치게 큰 모델은 고사양의 컴퓨팅 환경을 요구하여 일반적인 데스크톱 환경에서 구현하기 어렵다. 따라서, 성능과 효율성을 모두 고려한 중간 규모의 모델이 필요하다.

이러한 기준을 기반으로 2024년 8월 말 기준으로 EXAONE 3.0 7.8B Instruction 모델이 적합하다고 판단하였다. 이 모델은 78억 개의 매개변수를 가진 중간 크기의 언어 모델로, 한국어와 코딩 작업 모두에서 우수한 성능을 보여준다. 또한, 최근 LLM 트렌드인 decoder-only 구조를 채택하여 텍스트 생성 및 분석 작업에서 높은 효율성을 발휘한다(LG AI Research, 2024).

제 2 절 EXAONE 3.0 7.8B Instruction 모델 분석

1) 아키텍처 및 주요 매개변수 분석

EXAONE 3.0 7.8B Instruction 모델은 최근 LLM의 트렌드 방식인 decoder-only 구조를 채택하였다. 이 구조는 트랜스포머 모델의 디코더 부분만을 활용하여 텍스트 생성 및 분석에 최적화된 설계를 제공한다. decoder-only 모델은 이전 토큰의 문맥을 기반으로 다음 토큰을 예측하는 방식으로 작동하며, 텍스트 완성, 코드 생성, 대화형 AI와 같은 작업에서 우수한 성능을 보인다. 아래의 [표 3-1]은 모델의 주요 세부 사항을 정리한 것이다.

[표 3-1] EXAONE 3.0 7.8B Instruction 주요 세부 사항 정리

구성 요소	설명
Number of Parameters	7.8B (78억 개)
Transformer Layers	32
Hidden Size (d_model)	4,096
Feedforward Dimension	14,336
Number of Attention Heads	32
Head Type	Grouped Query Attention(GQA)
Context Length	4,096 tokens
Vocabulary Size	102,400
Non-linearity	SwiGLU
Positional Embeddings	Rotary Position Embeddings(RoPE)
Training Dataset	1.2T tokens, 다중 언어 및 도메인 데이터 포함
Tokenizer Type	Byte-level Byte Pair Encoding (BBPE), 한국어와 영어 이중언어 지원

이 모델은 GQA(Grouped Query Attention)를 활용해 메모리 효율성을 높이고, RoPE(Rotary Position Embeddings)를 통해 긴 문맥에서도 학습과 추론이 가능하다. 또한, 1.2조 개의 토큰으로 사전 학습(pretraining)된 데이터 세트는 한국어와 영어, 코딩 관련 데이터까지 포함하여 다양한 도메인을 포괄한다. 또한, EXAONE 3.0 7.8B는 다음과 같은 평가에서 우수한 성능을 발휘하였다.

- 코딩 능력: HumanEval 벤치마크에서 72.0점을 기록하며, 동급 모델 대비 최고 성과를 보였다.

- 한국어 성능: KoMT-Bench에서 8.92점, LogicKor에서 8.62점을 기록하여 경쟁 모델보다 높은 정확도를 나타냈다. 이러한 이유로 EXAONE 3.0 7.8B 모델은 본 연구를 시작하게 되었던 당시 동종 sLLM 오픈소스 모델 중 시큐어 코딩 가이드 모델 구축을 위한 적합한 모델로 선정되었다.

제 4 장 데이터세트 선정 및 구축

제 1 절 데이터세트 선정 기준

본 연구는 데이터세트를 기반으로 특정 도메인에 특화된 모델을 만드는 것을 목표로 한다. 따라서 데이터세트의 품질과 구성은 연구의 성공 여부를 결정짓는 매우 중요한 요소였다. 그러므로 데이터세트 선정 기준을 아래 세 가지로 정의하였다.

- 안전하지 않은 코드와 안전한 코드의 비교 쌍 제공: 데이터세트에 보안상 위험한 코드와 이를 안전하게 변환한 코드가 포함되어 있어야 한다. 이를 통해 모델이 보안 약점을 식별하고 수정안을 제시하는 학습이 가능하다.

- 공신력 있는 출처와 관리 체계: 데이터세트가 공신력 있는 기관 및 단체에서 발행한 가이드라인을 기반으로 하거나, 오픈소스인 경우 다수가 주기적으로 관리하는 신뢰할 수 있는 자료여야 한다.

- 국내외에서 가장 많이 사용되는 프로그래밍 언어: 데이터세트는 세계적으로 많이 사용되는 언어를 포함하며, 특히 국내에서 가장 많이 사용되는 Python, Java, JavaScript와 같은 언어를 기반으로 작성되어야 한다.

위 기준에 부합하는 기존 데이터세트를 탐색하였으나, 적합한 자료를 찾을 수 없었다. 따라서 본 연구에서는 직접 데이터세트를 구축하였다.

제 2 절 데이터 수집 및 전처리 결과

이 연구에서 활용된 데이터세트는 [표 4-1]에 포함된 가이드라인과 자료를 참고하여 수집한 뒤, 목적에 맞게 가공하여 구성하였다.

[표 4-1] 수집한 데이터세트 상세 정보

자료명 및 출처	언어	주요 내용	형태	비고
Python 시큐어코딩 가이드(2023년 개정본), KISA	Python, 주석 한글	구현단계 보안약점 제거 기준 항목(49개) 중 46개에 대해 소개(한국인터넷진흥원(KISA), 2023, 재인용)	PDF	Python 3.X 버전을 기준으로 작성
JavaScript 시큐어 코딩 가이드(2023년 개정본), KISA	Javascript, 주석 한글	구현단계 보안약점 제거 기준 항목(49개) 중 42개에 대해 소개(한국인터넷진흥원(KISA), 2023, 재인용)	PDF	클라이언트 측 코드 예시는 ReactJS, 서버 측 코드 예시는 NodeJS 기반 ExpressJS(한국인터넷진흥원(KISA), 2023, 재인용)
소프트웨어 개발보안 가이드(2021년 개정본), KISA	C, C#, JAVA, 주석 한글	구현단계 보안약점 제거 기준에 대한 설명 및 보안대책과 JAVA, C 및 C#언어로 작성된 안전하지 않은 코딩 예제 기반의 시큐어코딩 예시 소개	PDF	필요한 항목에 대해서만 수집
SEI CERT Oracle Coding Standard for Java, Software Engineering Institute(Carnegie Mellon University)	JAVA, 주석 영어	Rule을 대주제로 세부적인 항목으로 나눔, 해당 항목에 대해 검사 가능한 도구들도 정리	웹페이지	중단되거나 소스코드가 존재하지 않는 규칙은 수집하지 않음
OpenSSF, Secure Coding One Stop Shop for Python	Python, 주석 영어	보안 약점 리스트(CWE) 기반으로 정리	Git Hub	CPython >= 3.9 이상의 코드 기준

수집한 데이터세트는 아래의 [그림 4-1]처럼 정리하였으며, 데이터는 CSV 파일 형태로 저장되었으며, 총 612개의 데이터로 구성되어 있다.

A	B	C	D	E	F
language	tag	vulnerability_type	description	vulnerable_code	secure_code
Javascript	입력데이터 검증 및 표현	SQL 삽입	SQL 질의문을 생성할 때 검증되지 않은 외부 입력값을 허용하여 악의적인 질의문이 실행가능한 보안약점	const mysql = require("mysql"); // 커넥션 조기화 옵션은 생략함 const connection = mysql.createConnection(...);	const mysql = require("mysql"); ... router.get("/patched/email", (req, res) => { const con =
Javascript	입력데이터 검증 및 표현	SQL 삽입	SQL 질의문을 생성할 때 검증되지 않은 외부 입력값을 허용하여 악의적인 질의문이 실행가능한 보안약점	const mysql = require("mysql"); const Sequelize = require("sequelize"); const { QueryTypes } =	router.get("/vuln/orm/email", (req, res) => { const userInput = req.query.id; // 쿼리 내에서 바인
Javascript	입력데이터 검증 및 표현	코드 삽입	프로세스가 외부 입력값을 코드(명령어)로 해석·실행할 수 있고 프로세스에 검증되지 않은 외부 입력값을 허용한 경우 악의적인 코드가 실행 가능한 보안약점	const express = require('express'); router.post("/vuln/server", (req, res) => { // 사용자로부터 전달받은 값을 그대로 eval 함수의 인자로 전달 const data =	const express = require('express'); function alphanumeric(input_text) { // 정규표현식 기반 문자열 검사 const letterNumber

[그림 4-1] 수집한 데이터세트 CSV 파일 캡처 화면

각 열(column)은 데이터의 특정 속성을 나타내며, 데이터세트의 구조를 명확히 설명하기 위해 다음과 같은 열을 포함하고 있다.

1. language: 수집된 코드의 프로그래밍 언어 타입
2. tag: 가이드라인의 섹션이나 항목
3. vulnerability_type: 코드에서 발생할 수 있는 보안 약점 키워드
4. description: 해당 보안 약점에 대한 설명 또는 CWE 매핑 정보
5. vulnerable_code: 보안상 위험한 코드(비준수 코드)
6. secure_code: 안전한 코드(준수 코드)

수집된 데이터세트는 바로 사용할 수도 있지만, 불필요한 정보들이 포함되어 있거나 특정 문맥에 대한 명확성이 부족할 수 있다. 또한, 해당 데이터셋을 EXAONE 3.0 모델의 학습에 효과적으로 활용하기 위해서는 모델의 요구사항에 맞춘 전처리가 필요하다. 이를 위해 EXAONE 3.0의 Special Tokens

를 활용하여 데이터세트에 의미를 추가하고 구조를 명확히 하였다.

아래의 [표 4-2]는 EXAONE 3.0 Instruct의 Special Tokens가 정의된 JSON 파일이다.

[표 4-2] EXAONE 3.0 Instruct Special Tokens Map


```
{
  "bos_token": {
    "content": "[BOS]",
    "lstrip": false,
    "normalized": false,
    "rstrip": false,
    "single_word": false
  },
  "eos_token": {
    "content": "[|endofturn|]",
    "lstrip": false,
    "normalized": false,
    "rstrip": false,
    "single_word": false
  },
  "pad_token": {
    "content": "[PAD]",
    "lstrip": false,
    "normalized": false,
    "rstrip": false,
    "single_word": false
  },
  "unk_token": {
    "content": "[UNK]",
    "lstrip": false,
    "normalized": false,
    "rstrip": false,
    "single_word": false
  }
}
```


해당 모델은 문맥 이해와 구성을 돕기 위해 다음과 같은 특수 토큰을 사

용한다. 각 특수 토큰은 의미를 가지고 있는데 아래의 [표 4-3]은 특수 토큰에 대한 설명을 나타난 것이다.

[표 4-3] EXAONE 3.0 특수 토큰 설명

특수 토큰	설명
[BOS]	문장의 시작을 나타냄.
[|endofturn|]	문장의 끝을 나타냄.
[PAD]	문장을 고정 길이로 맞추기 위한 패딩.
[UNK]	알 수 없는 단어를 나타냄.

특히, 데이터셋의 전처리 과정에서 중요한 역할을 하는 [BOS] 토큰과 [|endofturn|] 토큰은 문장의 시작과 끝을 명확히 나타내는 데 사용된다. 이를 통해 모델은 각 데이터 항목의 경계를 명확히 인식하고 문맥을 보다 정확히 이해할 수 있다. 아래의 [그림 4-2]는 토큰을 추가한 결과물을 캡처한 것이다.

	prompt	response
2	[BOS] 소스 코드: const mysql = require("mysql"); // 커넥션 초기화 옵션은 생략함 const connection = mysql.createConnection(...); router.get("/vuln/email", (req, res) => { const con = connection; const userInput = req.query.id; // 사용자로부터 입력받은 값을 검증 없이 그대로 쿼리에 사용 const query = `SELECT email FROM user WHERE user_id =	[BOS] 보안 취약점 설명: SQL 질의문을 생성할 때 검증되지 않은 외부 입력값을 허용하여 악의적인 질의문이 실행가능한 보안약점 취약점 유형: SQL 삽입 안전한 코드: const mysql = require("mysql"); ... router.get("/patched/email", (req, res) => { const con = connection;
3	[BOS] 소스 코드: const mysql = require("mysql"); const Sequelize = require("sequelize"); const { QueryTypes } = require("sequelize"); // 커넥션 및 ORM 초기화 옵션은 생략함 const connection = mysql.createConnection(...); const sequelize = new Sequelize(...); router.get("/vuln/orm/email", (req, res) => {	[BOS] 보안 취약점 설명: SQL 질의문을 생성할 때 검증되지 않은 외부 입력값을 허용하여 악의적인 질의문이 실행가능한 보안약점 취약점 유형: SQL 삽입 안전한 코드: router.get("/vuln/orm/email", (req, res) => { const userInput = req.query.id;
4	[BOS] 소스 코드: const express = require('express'); router.post("/vuln/server", (req, res) => { // 사용자로부터 전달 받은 값을 그대로 eval 함수의 인자로 전달 const data = eval(req.body.data); return res.send({ data }); }); [|endofturn|]	[BOS] 보안 취약점 설명: 프로세스가 외부 입력값을 코드(명령어)로 해석·실행할 수 있고 프로세스에 검증되지 않은 외부 입력값을 허용한 경우 악의적인 코드가 실행 가능한 보안약점 취약점 유형: 코드 삽입 안전한 코드: const express = require('express'); function alphanumeric(input_text) { // 정규표현식 기반 문자열 검사 const letterNumber = /^[0-9a-zA-Z]+$/;

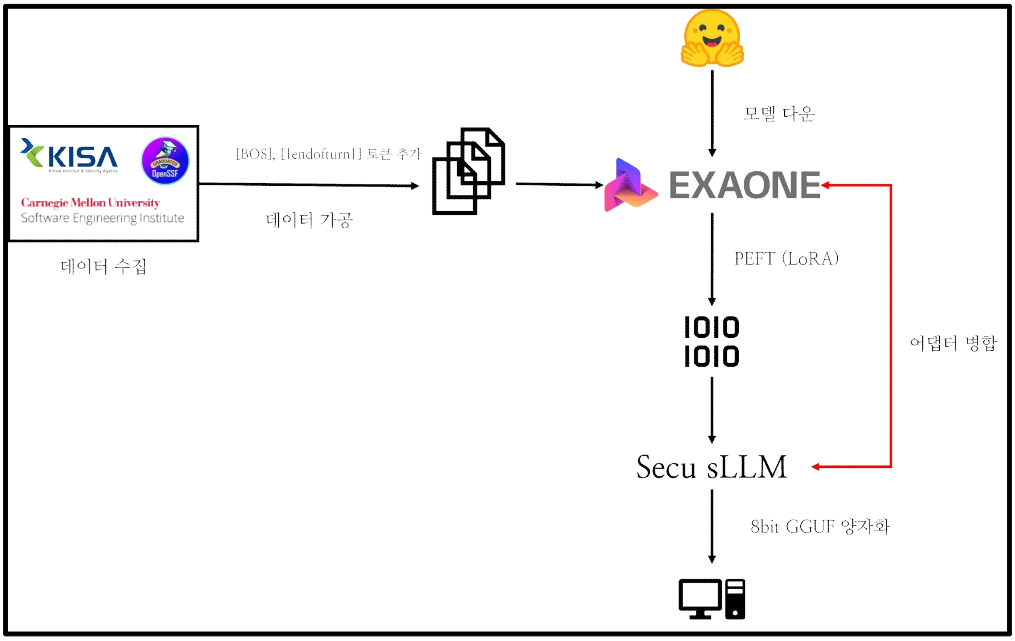
[그림 4-2] 특수 토큰을 활용한 데이터세트 가공 후 캡처 화면

제 5 장 제안하는 sLLM 모델

제 1 절 Secu sLLM 모델 개요

연구에서 제안하는 Secu sLLM은 일반적인 데스크톱 환경에서도 구동 가능한 경량화된 언어 모델로, 시큐어 코딩 지원을 목적으로 설계되었다. 기존 LLM의 한계로 지적된 높은 하드웨어 요구사항과 데이터 유출 문제를 극복하기 위해, Secu sLLM은 오픈소스 모델을 기반으로 보안 데이터세트를 활용한 도메인 특화 학습을 진행하였다.

Secu sLLM의 생성 과정을 도식화한 [그림 5-1]은 데이터 전처리, 모델 학습, 그리고 모델 병합의 주요 단계를 포함한다. 이를 통해 Secu sLLM은 시큐어 코딩 가이드라인에 기반한 보안 약점 검출 및 개선 코드를 생성하는 데 특화된 모델로 구현되었다.



[그림 5-1] Secu sLLM 모델 생성 과정

[그림 5-1]에서 보이듯, Secu sLLM의 주요 단계는 다음과 같다.

- 데이터 수집 및 전처리: Secu sLLM은 공신력 있는 시큐어 코딩 가이드 데이터를 기반으로 학습한다. 데이터 수집 후, 비준수 코드와 준수 코드를 매핑하여 학습 가능하도록 정리하고, 필요 없는 정보를 제거하는 전처리 작업을 수행한다.

- 모델 학습: 7.8B 규모 모델이라도 모든 파라미터를 학습시키는 Full Fine-Tuning 기법은 많은 하드웨어 자원과 긴 시간을 요구한다. 이를 해결하기 위해, 일부 파라미터만 학습하는 PEFT(Parameter-Efficient Fine-Tuning) 기법(허정준, 2024)을 사용하여 단일 GPU 환경에서도 효율적인 학습이 가능하게 하였다. 특히, 다양한 PEFT 기법 중 가장 널리 활용되는 LoRA(Low-Rank Adaptation)를 적용(Hu et al., 2022)하여, 효율적인 학습을 진행했습니다. 이 과정에서 특수 토큰을 활용하여 모델의 문맥 이해 능력을 강화하였다.

- 모델 병합 및 양자화: 학습 과정에서 특정 레이어의 가중치 일부만 조정된 결과로 어댑터 파일이 생성된다. 이 파일은 기존 EXAONE 3.0 모델과 병합하여 새로운 기능을 통합한다. 마지막으로, 일반적인 컴퓨팅 환경에서도 원활히 구동할 수 있도록 8bit GGUF 양자화(Hugging Face, n.d.)를 수행하여 모델을 최적화하였다.

제 2 절 실험 환경

LoRA 기법을 사용하더라도 모델의 실행과 학습에는 여전히 많은 자원이 요구된다. 이를 해결하기 위해 본 연구는 클라우드 환경에서 학습과 추론을 지원하는 Google Colab의 유료 서비스인 Colab Pro+를 활용하여 진행되었다. 아래의 [표 5-1], [표 5-2], [표 5-3]에는 본 연구에서 사용된 컴퓨팅 환경과 주요 라이브러리가 정리되어 있다.

[표 5-1] Colab Pro+ 환경

시스템 RAM	GPU RAM	디스크
83.5GB	NVIDIA A100 (40GB)	235.7GB

[표 5-2] 모델 학습 및 파인튜닝 관련

라이브러리	버전
transformers	4.46.3
peft	0.13.2
torch	https://download.pytorch.org/whl/cu121_full/torch-2.5.1%2Bcu121-cp310-cp310-linux_x86_64.whl
bitsandbytes	0.45.0
diffusers	0.31.0

[표 5-3] 데이터 처리 및 전처리

라이브러리	버전
datasets	3.1.0
numpy	1.26.4
pandas	2.2.2
huggingface-hub	0.26.3
sentencepiece	0.2.0
scikit-learn	1.5.2
scipy	1.13.1

제 3 절 모델 학습 방법 및 전략

클라우드 기반 컴퓨팅 환경을 구성한 후, 제작한 데이터세트를 활용하여 모델 학습을 진행하였다. 학습이 완료된 어댑터 파일은 기본 모델과 병합한 뒤 허깅페이스 Hub에 공유하였다. 아래 [그림 5-2], [그림 5-3], [그림 5-4], [그림 5-5]에는 학습에 사용된 양자화 기법과 하이퍼파라미터 설정이 주석과 함께 포함되어 있다.

```
# LoRA 설정

peft_params = LoraConfig(
    r=32,  # LoRA의 저랭크 차원(rank)
    lora_alpha=16,  # LoRA scaling factor
    lora_dropout=0.1,  # 드롭아웃 확률
    bias="none",  # bias는 학습하지 않음.
    task_type="CAUSAL_LM",  # 작업 유형.
    target_modules=[  # LoRA가 적용될 모델의 대상 모듈.
        "q_proj",
        "k_proj",
        "v_proj",
        "o_proj",
        "gate_proj",
        "up_proj",
        "down_proj",
    ],
)
```

[그림 5-2] LoRA 설정 코드

```
# BitsAndBytes 양자화 설정
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,  # 4비트 양자화 모델 로드 활성화.
    bnb_4bit_use_double_quant=True,  # 더블 양자화 활성화.
    bnb_4bit_quant_type="nf4",  # 양자화 유형. "nf4"는 Non-Fixed 4-bit 양자화 방식을 사용.
    bnb_4bit_compute_dtype=torch.bfloat16,  # 계산 시 사용하는 데이터 타입을 bfloat16으로 설정.
)
```

[그림 5-3] bitsandbytes 양자화 설정 코드

```python
# 학습 데이터의 Prompt와 Response를 결합하여 하나의 텍스트로 구성
def formatting_prompts_func(examples):
    prompts = examples["prompt"]  # 데이터셋의 "prompt" 열
    responses = examples["response"]  # 데이터셋의 "response" 열
    texts = []
    for prompt, response in zip(prompts, responses):
        # prompt와 response를 이어붙이되, 필요에 따라 문자열 양쪽 공백 제거
        # BOS (시작 토큰) 및 EOS (끝 토큰)는 모델 학습 규칙에 맞게 포함
        text = f"{prompt.strip()} {response.strip()}"
        texts.append(text)
    return {
        "text": texts,  # 결합된 텍스트를 새로운 열로 반환
    }

# 학습 및 검증 데이터에 Prompt와 Response를 결합하는 포맷팅 함수 매핑
formatted_train_dataset = train_dataset.map(
    formatting_prompts_func,
    batched=True,  # 데이터를 배치 단위로 처리
)
formatted_val_dataset = val_dataset.map(
    formatting_prompts_func,
    batched=True,
)

# 데이터셋의 텍스트를 모델 입력에 맞게 토큰화
def tokenize_function(examples):
    texts = examples["text"]  # 포맷팅된 텍스트
    if isinstance(texts, list) and all(isinstance(t, str) for t in texts):
        tokenized = tokenizer(
            texts,
            padding="max_length",  # 입력 길이를 고정 길이로 패딩
            truncation=True,  # 길이가 긴 입력을 잘라내기
            max_length=1024,  # 최대 길이 설정
            return_tensors="pt",  # PyTorch 텐서 형식으로 반환
        )
        return {
            "input_ids": tokenized["input_ids"],  # 토큰화된 입력 ID
            "attention_mask": tokenized["attention_mask"],  # 어텐션 마스크
        }
```

[그림 5-4] 데이터세트를 모델 입력에 맞게 토큰화 과정 코드

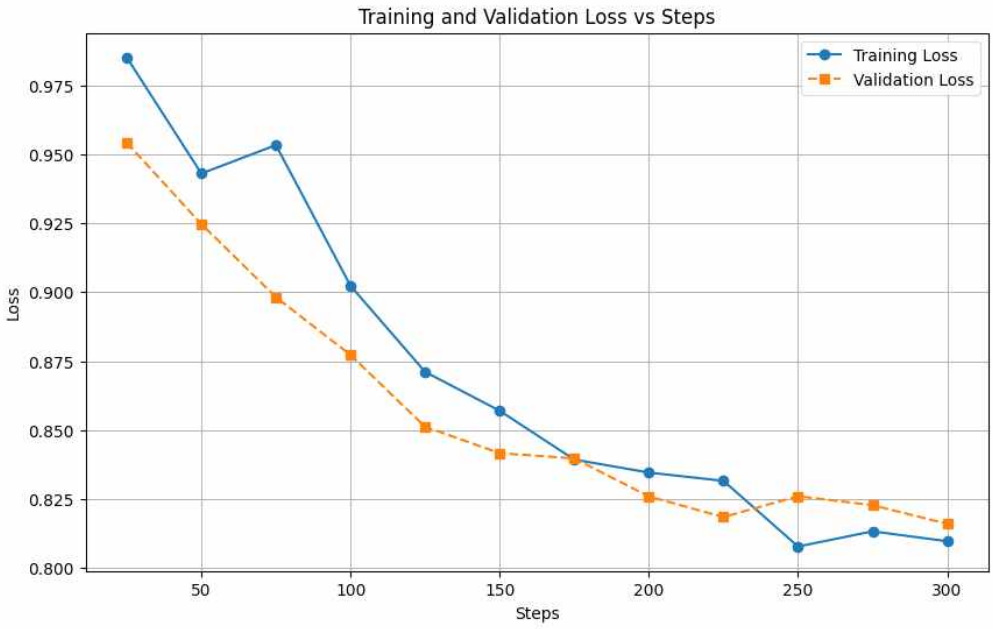
```python
# Supervised Fine-Tuning(SFT)을 위한 설정 객체로, 학습 환경 및 하이퍼파라미터를 정의
sft_config = SFTConfig(
    output_dir="./results",
    dataset_text_field="input_ids",  # 데이터셋의 텍스트 필드 이름을 'input_ids'로 지정
    dataset_num_proc=2,              # 데이터셋 처리에 사용할 병렬 프로세스 수
    max_seq_length=1024,             # 시퀀스의 최대 길이, 토크나이저와 동일하게 설정하여 일관성 유지
    num_train_epochs=10,             # 전체 학습 반복 횟수
    per_device_train_batch_size=2,   # 장치당 학습 배치 크기
    gradient_accumulation_steps=8,   # 그래디언트를 누적할 스텝 수 (가상 배치 크기 증가 효과)
    optim="paged_adamw_32bit",       # 옵티마이저 설정, 메모리 효율성을 고려한 AdamW 사용
    save_steps=25,
    logging_steps=20,
    learning_rate=2e-5,              # 학습률 설정, 모델의 학습 속도 조절
    weight_decay=0.01,               # 과적합 방지를 위한 가중치 감쇠
    fp16=False,
    bf16=False,
    max_grad_norm=0.3,               # 그래디언트 클리핑을 위한 최대 노름 설정
    max_steps=-1,
    warmup_ratio=0.03,               # 학습 초기에 학습률을 점진적으로 증가시키는 비율
    lr_scheduler_type="cosine",      # 학습률 스케줄러: 코사인 감소 스케줄
    report_to="none",
    group_by_length=True,
    gradient_checkpointing=True,
    remove_unused_columns=False,
    eval_strategy="steps",
    eval_steps=25,
    save_total_limit=2,
)
```

[그림 5-5] 지도학습 파인튜닝 하이퍼파라미터 정의 코드

제 6 장 실험 결과 및 성능 평가

제 1 절 실험 결과

학습에 필요한 모든 조건을 설정한 뒤, Trainer API를 사용하여 학습을 시작하였다. 초기 10회 에포크 학습 결과, Training Loss가 1.3x로 나타나 추가 학습이 필요함을 확인하였다. 이에 따라, 10회 학습된 어댑터 파일을 기본 모델에 병합한 후, 미세 조정된 모델을 바탕으로 다시 10회 에포크 학습을 진행하였다. 학습 과정에서는 모델이 잘 학습되고 있는지, 과적합이 발생하지 않는지를 확인하기 위해 데이터세트를 학습용과 검증용으로 8:2 비율로 나누어 실험을 수행하였다. 아래[그림 6-1]는 Training Loss와 Validation Loss의 감소 추세를 시각화한 것이다.



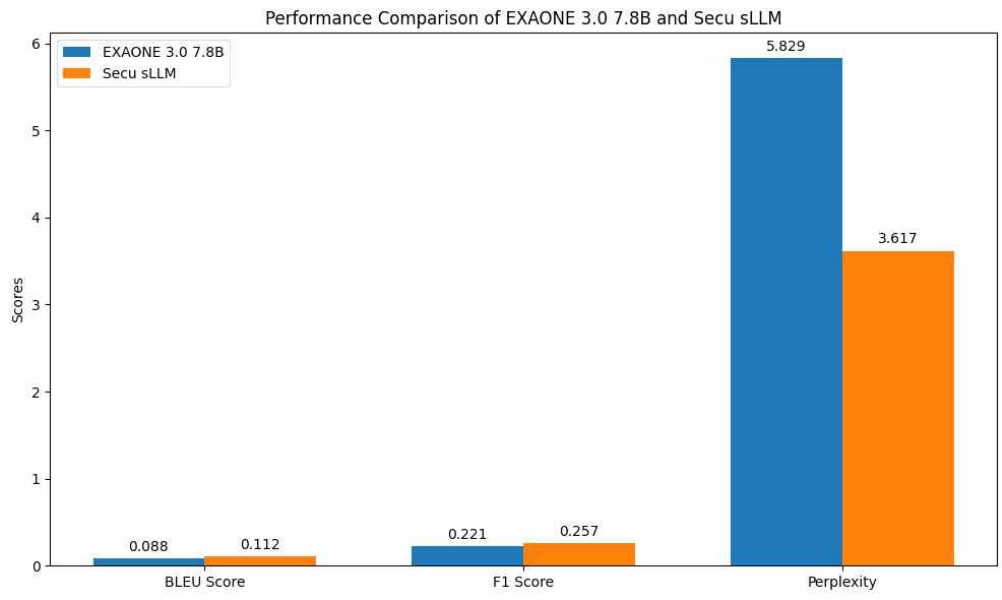
[그림 6-1] Training Loss와 Validation Loss 통계 그래프

학습 결과, Training Loss와 Validation Loss 모두 학습이 진행될수록 안정적으로 감소하는 경향을 보였다. 특히, 두 손실 값 간의 차이가 점진적으로 줄어들며, 과적합 없이 모델이 효과적으로 학습되었음을 확인할 수 있다.

제 2 절 성능 평가 지표 및 추론 결과

완성된 모델을 연구 목적에 맞는 추론이 가능한지 평가를 진행하였다. 본 연구는 비 준수 코드를 입력하면 해당 코드의 보안 위협을 알려주며 준수 코드를 알려주는 모델이므로 관련 평가를 진행할 수 있는 평가 데이터세트를 찾아보았지만, 특정 언어만 검사하거나, 모델이 평가 데이터세트에 지원을 안 하는 등의 제약조건이 있었다. 그래서 기존에 만든 데이터세트를 활용해 무작위로 30개를 추출해 평가 데이터세트로 제작하여 LLM을 평가할 때 주로 사용하는 평가 방법인(유원준, 안상준, 2024) F1 Score, BELU(Bilingual Evaluation Understudy Score), Perplexity를 사용했다.

[그림 6-2]는 세 가지 평가지표를 활용하여 비교 분석한 결과를 시각화한 것이다. 그래프에서 확인할 수 있듯이, 제작된 모델의 F1 Score와 BLEU 점수가 기본 모델보다 높게 나타났으며, 혼란도를 측정하는 Perplexity 역시 낮은 값을 기록하였다. 이는 완벽한 응답에 더 가까워졌음을 의미한다. 특히, FP32(32비트 부동소수점) 타입보다 BF16(16비트 부동소수점)으로 최적화된 Secu sLLM이 더 나은 성능을 보였음을 알 수 있다.



[그림 6-2] 기본 모델과 제작한 모델 비교 그래프

마지막으로 추론 능력을 평가하기 위해 30개의 시큐어 코딩 평가 데이터 세트를 활용하였다. 아래의 [표 6-1]은 기본 모델과 Secu sLLM의 추론 비교 결과를 정리한 것이다.

[표 6-1] 추론 비교 결과

Input Text	EXAONE 3.0 Output	Secu sLLM Output
소스 코드: const userInput = "return 2 + 2"; const func = new Function(userInput); console.log(func());	1. It defines a string `userInput` containing the expression `return 2 + 2`. 2. It creates a new function using the `Function` constructor with `userInput` as its argument. 3. It calls this newly created function and logs the result to the console. Let's break it down step by step: ```javascript const userInput = "return 2 + 2"; ``` This line stores the string `"return 2 + 2"` in the variable `userInput`. ```javascript const func = new Function(userInput); ``` This line creates a new function using the `Function` constructor. The `Function` constructor takes the `userInput` string and executes it as JavaScript code, effectively creating a function that returns `4`. ```javascript console.log(func()); ``` This line calls the newly created function `func()` and logs its result to the console. Since our function returns `2 + 2`, which is `4`, the output will be: ``` 4 ``` So, when you run this code, you should see `4` printed in the console.	이 코드는 사용자 입력을 함수로 실행하는 방식으로, 보안 취약점이 있습니다. `new Function()`을 사용하면 임의의 코드를 실행할 수 있기 때문에, 악의적인 사용자가 악의적인 코드를 입력할 경우 심각한 보안 문제가 발생할 수 있습니다. 보안을 강화하기 위해 다음과 같은 방법을 고려해볼 수 있습니다: 1. 사용자 입력을 검증하고 허용된 범위 내에서만 실행하도록 제한합니다. 2. 안전한 라이브러리나 프레임워크를 사용하여 코드 실행을 제한합니다. 3. 코드 실행 대신 미리 정의된 함수를 호출하는 방식으로 변경합니다. 예시: ```javascript const safeFunction = (a, b) => a + b; console.log(safeFunction(2, 2)); ``` 이와 같이 안전한 방식으로 코드를 작성하면 보안 취약점을 줄일 수 있습니다.

소스 코드: public int doOperation(int exp) { // Compute 2^exp int temp = 1 << exp; // Do other processing return temp; }	코드 설명: 1. `int temp = 1 << exp;`: 비트 시프트 연산자를 사용하여 1을 exp만큼 왼쪽으로 이동시킵니다. 이는 2의 exp 제곱을 계산하는 효율적인 방법입니다. 2. 이후에는 다른 처리가 필요할 수 있지만, 현재 코드에서는 추가 처리가 없습니다. 3. 마지막으로 `return temp;`를 통해 계산된 값을 반환합니다. 이 함수는 다음과 같은 방식으로 작동합니다: - exp가 0일 때: 2^0 = 1 - exp가 1일 때: 2^1 = 2 - exp가 2일 때: 2^2 = 4 - exp가 3일 때: 2^3 = 8 ... 예를 들어, `doOperation(3)`을 호출하면 결과는 8이 됩니다. 이 함수는 주로 지수 계산이 필요한 상황에서 사용될 수 있습니다. 예를 들어, 알고리즘에서 거듭제곱 계산이 필요할 때 유용하게 사용할 수 있습니다.	```java public int doOperation(int exp) { // Compute 2^exp int temp = 1 << exp; // Do other processing return temp; }``` 이 코드의 주요 부분은 다음과 같습니다: 1. `int temp = 1 << exp;`: 비트 시프트 연산자를 사용하여 2의 exp 제곱을 계산합니다. 2. `return temp;`: 계산된 값을 반환합니다. 이 코드는 다음과 같은 문제점을 가지고 있습니다: 1. 함수 이름이 `doOperation`인데, 실제로 하는 일은 단순히 2의 exp 제곱을 계산하는 것입니다. 2. 반환 타입이 `int`인데, 실제로 반환되는 값은 항상 2의 exp 제곱이므로 반환 타입을 `long`으로 변경하는 것이 좋습니다. 3. 주석에 "Do other processing"이라고 되어 있지만 실제로는 아무 작업도 하지 않습니다. 수정된 코드는 다음과 같습니다: ```java public long computePowerOfTwo(int exp) { long result = 1L << exp; return result; }``` 이렇게 수정하면 함수의 목적과 반환 타입이 더 명확해집니다.

소스 코드: int x = 50; int y = x << 2; x += y + 1;	1. `int x = 50;` - 변수 `x`에 50을 할당합니다. 2. `int y = x << 2;` - `x`의 값을 왼쪽으로 2비트 시프트합니다. 50의 이진 표현은 `110010`, 이를 왼쪽으로 2비트 시프트하면 `11001000`이 됩니다. 즉, 100이 됩니다. - 따라서 `y`는 100이 됩니다. 3. `x += y + 1;` - `y`의 값에 1을 더한 후 `x`에 더합니다. - `y + 1`은 101입니다. - `x`는 원래 50이었고, 여기에 101을 더하면 151이 됩니다. 따라서 최종적으로 `x`의 값은 151이 됩니다. 이 코드의 결과는 다음과 같습니다: ```cpp x = 151 ```	1. `int x = 50;` - 변수 x에 50을 할당합니다. 2. `int y = x << 2;` - x의 값을 2비트 왼쪽으로 시프트하여 y에 저장합니다. - 50을 2비트 왼쪽으로 시프트하면 200이 됩니다. - 따라서 y = 200 3. `x += y + 1;` - x에 y와 1을 더한 값을 더합니다. - x = x + y + 1 - x = 50 + 200 + 1 = 251 최종적으로 x의 값은 251이 됩니다.

[표 6-1]을 통해 알 수 있는 사실은 제안된 Secu sLLM 사용자가 원하는 결과에 더욱 가깝게 대답하며, 영문 주석인 데이터를 넣어도 한글 답변으로 추론해 준다는 사실이다.

제 7 장 결론 및 향후 연구 방향

본 연구는 보안 중심의 소프트웨어 개발을 지원하기 위해 시큐어 코딩에 특화된 Secu sLLM 모델을 설계하고 구현하였다. 기존 LLM 모델의 한계를 극복하고자 LoRA 기법과 BF16 최적화를 적용하여, 경량화와 효율성을 동시에 추구하였다. 이를 통해 Secu sLLM은 소프트웨어 보안 약점 탐지와 보안 코드 생성을 효과적으로 수행할 수 있는 모델임을 실험적으로 입증하였다.

BLEU 점수와 F1 Score에서 기존 EXAONE 3.0 7.8B 모델 대비 우수한 성과를 보였으며, Perplexity에서도 낮은 값을 기록하여 신뢰할 수 있는 결과를 도출하였다. 이러한 결과는 Secu sLLM이 시큐어 코딩 가이드 LLM으로 충분히 활용 가능성이 있음을 보여준다.

또한, 시큐어 코딩 특화 모델의 지속적인 발전과 실제 개발 환경에서의 성능 검증을 할 수 있도록 허깅페이스 Hub에 학습된 모델[2]과 사용한 데이터세트[3]를 공개하였다. 이를 통해 연구자와 개발자들이 Secu sLLM을 활용하고 개선할 수 있는 기틀을 제공하며, 시큐어 코딩 특화 모델 분야의 발전을 도모하고자 한다.

2) https://huggingface.co/LINKER98/secusLLM-Fine-tuning

3) https://huggingface.co/datasets/LINKER98/secusLLM-Dataset

참 고 문 헌

1. 국내문헌

한국인터넷진흥원(KISA). (2021).『소프트웨어 개발보안 가이드』. 서울: 한국인터넷진흥원.

한국인터넷진흥원(KISA). (2023).『Python 시큐어코딩 가이드(개정판)』. 서울: 한국인터넷진흥원.

김동연, 김세진, 이도경, 이채윤, 임승연, 서혁준. (2023).『AI 기반 시큐어 코딩 점검 도구 개발에 관한 연구』. 한국정보처리학회 추계학술발표대회 논문집.

도예진, 김형식. (2024).『LangChain을 활용한 악성코드 자동 분석 프로세스 설계』. ASK 2024 학술발표대회 논문집, 31(1), 364－366.

한국인터넷진흥원(KISA). (2023).『Javascript 시큐어코딩 가이드(개정판)』. 서울: 한국인터넷진흥원.

허정준. (2024).『LLM을 활용한 실전 AI 애플리케이션 개발』 경기: 책만

유원준, 안상준. (2024).『딥러닝을 이용한 자연어 처리 입문』 위키독스: https://wikidocs.net/book/2155

2. 국외문헌

Motlagh, F. N., Hajizadeh, M., Majd, M., Najafi, P., Cheng, F., & Meinel, C. (2024). Large language models in cybersecurity: State-of-the-art. arXiv:2402.00891.

Vaswani et al. (2017). Attention is all you need. Advances in Neural Information Processing Systems (NIPS), 30, 5998-6008.

Jin, Y., Jang, E., Cui, J., Chung, J.-W., Lee, Y., & Shin, S. (2023). DarkBERT: A Language Model for the Dark Side of the Internet. Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL), 7515-7533.

LG AI Research. (2024). EXAONE 3.0 7.8B Instruction Tuned Language Model. LG AI Research.

Hu, E., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2022). LoRA: Low-Rank Adaptation of Large Language Models. Proceedings of the 10th International Conference on Learning Representations (ICLR).

3. 기타문헌

장종원. (2024).『기업들의 생성형 AI 개발, 현재는?』. 서울: 삼성SDS 인사이트 리포트.

SEI CERT. (2024).『SEI CERT Oracle Coding Standard for Java』. SEI CERT Wiki.

OpenSSF. (2024).『Secure Coding One Stop Shop for Python』. GitHub Repository.

Hugging Face. (2024). GGUF Model Format. Retrieved from https://huggingface.co/docs/hub/gguf

ABSTRACT

Secu sLLM: An sLLM Optimized for Secure Coding in Desktop Computing Environments

Lee, Chan-Woo

Major in Computer Engineering

Dept. of Computer Engineering

The Graduate School

Hansung University

As computing environments and generative AI continue to evolve, IT has become an indispensable factor in all industries. As a result, companies have established and are operating IT departments, and governments are mandating compliance with the SW Security Development Guide during the development process. However, many startups rushing to release their services, as well as companies with insufficient IT personnel, fail to adhere to these secure coding guidelines. To address this, a secure coding development approach utilizing generative AI has been proposed. However, this approach also faces two major challenges. First, due to the continuous learning characteristic of generative AI, there is a risk that confidential corporate code and sensitive information may be incorporated into the training data. Second, costs arise according to usage. Another proposed solution is to build and

utilize open-source based LLM models in local environments, but this too presents two issues. First, LLMs with strong Korean language capabilities typically require high-performance computing resources that are difficult to secure in a standard office setting. Second, since these models are trained on general-purpose data rather than being specialized in secure coding, they may fail to produce the desired outcomes in terms of development security. In response, this paper aims to leverage a small Large Language Model (sLLM) capable of running smoothly even on low-spec computing environments. By employing sLLM, the risks associated with data leakage in commercial LLM services and the challenges posed by high-performance computing requirements can be overcome. Furthermore, this model can be designed and utilized specifically for secure coding. The proposed model is trained on secure coding materials verified by authoritative organizations, in accordance with the Secure Coding Guide by Programming Language issued by the Korea Internet and Security Agency, allowing it to effectively detect security vulnerabilities in source code.

【KEYWORD】LLM, sLLM, Secure Coding, AI Security, Desktop Environment