JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 27, No. 1, pp. 151-164, Jan. 2026

Check for updates

생성형 AI를 이용한 프로그래밍 교육: 코드 생성 정확도 평가와 표절 탐지 중심으로

하 진 영*
강원대학교 컴퓨터공학과 교수

Programming Education Using Generative AI: Focusing on Evaluation of Code Generation Accuracy and Plagiarism Detection

Jin-Young Ha*
Professor, Department of Computer Engineering, Kangwon National University, Gangwon 24341, Korea

[요 약]

본 연구는 생성형 AI를 프로그래밍 교육에 통합하기 위한 구조화된 접근을 제안하고, 주요 AI 모델의 성능 및 표절 탐지 가능성을 분석한다. ChatGPT 4o, ChatGPT 5, Gemini 2.5 Pro, Gemini 3 Pro를 동일한 조건에서 문제해결프로그래밍 과목 실습 문제로 평가하였으며, Sharif Judge 기반 자동 채점과 통계 분석을 통해 성능을 비교하였다. 그 결과 Gemini 3 Pro가 가장 높은 정확도와 안정성을 보였고, ChatGPT 5가 그 뒤를 이었다. 또한 증가하는 AI 활용 표절 문제에 대응하기 위해 ChatGPT로 생성한 다수의 참조 코드를 MOSS 유사도 분석과 결합한 결과, 비 AI 제출물에 대해 0%의 오탐지율, AI 생성 코드에 대해 최대 96.6%의 탐지율을 달성하였다. 본 연구는 생성형 AI가 교육 콘텐츠 개발, 수업 운영, 평가를 지원할 수 있는 가능성을 보여주며, 교육 현장에서 AI 생성 코드를 식별하기 위한 효과적인 방법을 제공한다.

[Abstract]

This study presents a structured approach to integrating generative AI into programming education and investigates the performance and plagiarism-detection capability of leading AI models. ChatGPT 4o, ChatGPT 5, Gemini 2.5 Pro, and Gemini 3 Pro were evaluated on Problem Solving Programming Class exercises under identical settings using Sharif Judge–based automated grading and statistical analyses. Gemini 3 Pro demonstrated the highest accuracy and output stability, followed by ChatGPT 5. To address the growing issue of AI-assisted plagiarism, the study employed multiple ChatGPT-generated reference codes combined with MOSS analysis, achieving a 0% false positive rate for non-AI submissions and up to 96.6% detection accuracy for AI-generated code. The findings highlighted the potential of generative AI to support content development, instruction, and assessment while also offering an effective method for identifying AI-generated code in academic settings.

색인어 : 생성형 AI, 챗지피티, 프로그래밍 교육, 자동 평가, 표절 탐지
Keyword : Generative AI, ChatGPT, Programming Education, Automatic Evaluation, Plagiarism Detection

http://dx.doi.org/10.9728/dcs.2026.27.1.151

Received 04 December 2025; **Revised** 23 December 2025
Accepted 15 January 2026

***Corresponding Author; Jin-Young Ha**
Tel: ※ 개인정보 표시제한
E-mail: jyha@kangwon.ac.kr

Ⅰ. 서 론

최근 ChatGPT를 비롯한 생성형 AI(generative artificial intelligence) 기술의 급격한 발전은 교육, 산업, 콘텐츠 제작 등 다양한 분야에서 새로운 변화를 촉발하고 있다. 특히 대규모 언어 모델(LLM; large language model)을 기반으로 하는 생성형 AI는 자연어 처리뿐만 아니라 프로그래밍 언어 영역에서도 높은 성능을 보이며, 학습자와 교수자 모두에게 새로운 도구로 자리 잡고 있다[1]-[4]. 이들 모델은 문제 설명을 자연어로 입력하면 코드, 알고리즘, 오류 수정 방안을 즉시 생성할 수 있어 프로그래밍 교육의 접근성과 효율성을 크게 높일 잠재력을 갖는다.

이러한 가능성에도 불구하고, 생성형 AI의 활용은 교육 현장에서 여러 새로운 문제를 야기하고 있다. 우선, 학습자가 AI가 작성한 코드를 그대로 제출하는 사례가 증가하면서 기존 표절 탐지 도구가 AI 생성 코드를 효과적으로 구별하지 못하는 한계가 드러나고 있다[5]. 또한 기존 자동 채점 시스템은 주로 출력 기반 평가에 의존하기 때문에 코드 품질, 논리적 구조, 예외 처리 등 고도화된 프로그래밍 역량을 평가하는 데 한계가 있다. 나아가 교수자는 수업 콘텐츠 제작, 실습 문제 생성, 개별 피드백 제공 등 교육 과정 전반에서 상당한 시간과 노력을 요구받고 있으며, 대규모 학급 환경에서는 모든 학습자에게 충분한 피드백을 제공하기 어렵다.

기존 연구들은 ChatGPT 등 생성형 AI를 프로그래밍 교육에 활용한 다양한 사례를 소개하고 있으나, 실제 교육 과정 전반(콘텐츠 제작-수업 운영-평가)에 걸쳐 체계적으로 분석한 연구는 아직 부족하다[3],[6],[7]. 또한 주요 AI 모델 간 성능 차이를 실험적으로 비교하거나, AI 생성 코드에 대한 표절 탐지의 가능성과 한계를 정량적으로 검증한 연구 또한 제한적이다[5]. 이와 같은 공백은 생성형 AI의 교육적 활용 가능성을 객관적으로 평가하는 데 어려움을 초래한다.

본 연구는 생성형 AI를 프로그래밍 실습 교육 과정에 적용 가능한 운영 관점에서 분석하고 있으며, 이는 교육 과정을 설계·개발·수행·평가라는 단계로 구분하여 접근하는 ADDIE 모델의 절차적 관점과 개념적으로 유사하다[8]. 다만 Dick & Carey 모델과 같이 교수 체계를 정밀하게 구조화하여 설계·검증하는 연구라기보다는, 실습 기반 교육 환경에서 각 단계별 활용 가능성과 실증 분석에 초점을 둔 탐색적 연구이다[9]. 이에 본 연구는 생성형 AI를 프로그래밍 교육의 여러 단계에 적용하는 방안을 실증적으로 분석하는 데 초점을 맞추고, 구체적으로는 (1) 수업 콘텐츠 제작, (2) 교육 과정 운영, (3) 학습 결과 평가라는 세 가지 관점에서 생성형 AI의 활용 가능성과 사례를 정리하고, (4) 주요 생성형 AI 모델인 ChatGPT와 Gemini의 프로그래밍 문제 해결 능력을 비교 분석한다. 또한 (5) CodeRunner 및 Sharif Judge 기반의 자동 채점 환경에서 AI 생성 코드의 표절 탐지 가능성을 정량적으로 검토함으로써, 향후 교육 현장에서의 AI 활용 전략을 제시하고자 한다. 본 연구는 새로운 표절 탐지 알고리즘을 제안하는 것이 아니라, 교육 현장에서 즉시 적용 가능한 실용적 운영 방법론을 제시하는데 목적이 있다. 또한 본 연구의 초점이 학습 성취도나 만족도와 같은 교육 효과의 직접 검증이 아니라, 생성형 AI 기반 평가 및 운영 체계의 기술적 타당성 검증에 있음도 밝힌다.

본 논문의 구성은 다음과 같다. 2장에서는 생성형 AI, 프로그래밍 교육, 자동 평가 및 표절 탐지와 관련된 기존 연구를 검토한다. 3장에서는 교육 콘텐츠 제작, 수업 운영, 평가 과정에서 생성형 AI를 활용하는 구체적 방안을 제시한다. 4장에서는 주요 생성형 AI 모델의 코드 생성 정확도를 비교 실험하고, 5장에서는 Sharif Judge 및 MOSS(Measure of Software Similarity) 기반 표절 탐지 실험 결과를 보고한다. 마지막으로 6장에서는 연구 결과를 요약하고, 생성형 AI의 교육적 활용에 대한 시사점과 향후 연구 방향을 제시한다.

Ⅱ. 관련 연구

본 장에서는 생성형 AI 기술의 발전과 프로그래밍 교육에서의 활용 연구, 자동 채점 시스템, 그리고 코드 표절 탐지 기법에 관한 기존 연구를 검토한다. 이를 통해 본 연구가 다루는 문제의 위치를 명확히 하고, 선행 연구의 한계와 본 연구의 차별점을 제시하고자 한다.

2-1 생성형 AI와 코드 생성 연구

대규모 언어 모델의 발전은 코드 생성, 코드 요약, 디버깅 등 다양한 프로그래밍 지원 기능을 가능하게 하였다. OpenAI의 ChatGPT와 구글의 Gemini 등 주요 생성형 AI 모델들은 자연어 기반의 문제 설명으로부터 직접 실행 가능한 코드를 생성하거나, 코드의 문제를 분석하여 해결 방법을 제시하는 능력을 보여 왔다.

이들 모델의 프로그래밍 능력은 새로운 버전이 나오면서 빠르게 향상되고 있으며, Python, Java, C++ 등 주요 언어에서 인간 초급 학습자 수준에 근접하거나 이를 능가하는 결과도 보고되고 있다. 그러나 생성형 AI의 응답 품질은 프롬프트 구조, 문제 난이도, 모델 설정에 따라 크게 달라지는 특성이 있어 일관된 성능 평가가 어렵다는 문제가 지적된다[5],[7].

1) ChatGPT

먼저, OpenAI의 ChatGPT는 생성형 AI의 대중화를 촉발한 가장 중요한 전환점으로 평가된다. GPT-3.5 및 GPT-4 계열 모델을 기반으로 한 ChatGPT는 자연어 대화, 코드 생성, 문제 해결, 요약·분석 등 다양한 영역에서 뛰어난 성능을 보이며 교육 현장에서 실질적인 도움을 제공하고 있고 최근 버전 5.1까지 출시되었다. 그 결과, ChatGPT는 학습 보조, 코딩 튜터링, 자동 피드백 생성 등 프로그래밍 교육 전반에 빠르게 도입되어 새로운 교수·학습 패러다임을 형성하고 있

다. 특히 대부분의 학술 연구는 ChatGPT 기반으로 진행되어 왔다[1],[3]-[7].

2) Gemini

구글의 Gemini는 기존 언어 중심 LLM에서 더 나아가 텍스트, 코드, 이미지, 오디오, 비디오 등 여러 모드를 통합적으로 처리할 수 있는 멀티모달 통합 모델로 개발되었다. Gemini는 특히 복합 추론, 코드 분석, 시각 이해 및 생성, 대규모 데이터 기반 추론 능력 등을 강화하여 교육 콘텐츠 제작, 자동 채점, 멀티모달 숙제 지원 등 새로운 형태의 교육 서비스 가능성을 확장하고 있고 최근 3 Pro 버전까지 출시되었다[10]. Gemini를 이용한 학술 연구는 ChatGPT에 비해 많지 않으나 여러 연구가 진행되어 왔다[11],[12].

2-2 프로그래밍 교육에서의 AI 활용 연구

프로그래밍 교육에서 생성형 AI를 활용하는 연구는 크게 세 가지 방향으로 진행되고 있다. 첫째, 수업 콘텐츠·예제 코드 생성 연구로, 자연어 기반 요청을 통해 교수자가 즉시 코드 예제, 문제 설명, 퀴즈 등을 생성하도록 지원하는 연구가 보고되고 있다. 둘째, 학습자 피드백 제공 연구에서는 생성형 AI가 문법 오류, 예외 메시지, 논리적 오류를 분석해 실시간 피드백을 제공함으로써 초급 학습자의 이해를 돕는 효과가 확인되었다. 셋째, 튜터링 시스템 연구에서는 AI가 반복 설명, 예제 제시, 힌트 제공 등을 수행하여 개별 맞춤형 학습을 지원할 수 있음을 강조한다. 그러나 기존 연구 대부분은 단편적 활용 사례 또는 소규모 실험에 한정되어 있으며, 교육 전체 흐름(콘텐츠 제작–수업 운영–평가)에 걸친 체계적 분석 연구는 부족하다[3],[6],[13].

2-3 자동 채점 시스템 연구

프로그래밍 과제의 자동 채점은 오래전부터 연구되어 왔으며, 대표적으로 CodeRunner, AutoLab, Gradescope, 그리고 Sharif Judge 등이 널리 사용되고 있다. Sharif Judge는 특히 C, C++, Java, Python 등 여러 언어의 자동 채점을 지원하며, 제출 코드 실행, 출력 비교 기반 채점, 테스트 케이스 기반 평가 등 기능을 제공한다. 또한 역할 기반 사용자 관리, 제출 기록 다운로드, 재채점(rejudge) 기능 등이 제공되어 대학 프로그래밍 수업에서 활용도가 높다.

본 연구에서 사용한 Sharif Judge는 C, C++, Java, Python을 지원하는 오픈소스 온라인 채점 시스템으로, 프로그래밍 교육 환경에서 자동 채점과 학습 관리 기능을 제공하기 위해 개발되었다. 웹 인터페이스는 PHP(CodeIgniter 프레임워크)로 작성되었으며, 백엔드 채점 엔진은 BASH 스크립트로 구현되어 있어 경량성·확장성이 뛰어나다. Sharif Judge는 github에서 다운로드받을 수 있다[14]. 이 시스템은 관리자, 강의자, 조교, 학생 등 여러 사용자 역할을 지원하며, 제출물 대기열, 재채점(rejudge), 성적표(scoreboard), 마감 지연 감점 규칙 등의 교육 중심 기능을 갖추고 있다. Sharif Judge는 기본적으로 프로그램의 출력 정확성 평가를 중심으로 설계되었으며, “Output Comparison”과 “Tester Code” 방식 두 가지 채점 방식을 제공한다. 또한 제출된 코드 전체를 한 번에 ZIP으로 다운로드하거나 결과를 엑셀 파일로 내보낼 수 있어 대규모 수업 운영에도 적합하다. Python의 경우 기본적인 보안 수준만 제공되며, 완전한 샌드박스 기능이 제공되지 않기 때문에 안전성이 요구되는 환경에서는 별도의 보호 장치가 필요하다는 한계도 존재한다. 특이하게, Sharif Judge는 자체 표절 탐지 알고리즘을 제공하지 않지만, Stanford MOSS와의 연동을 통해 유사 코드 탐지 기능을 제공한다. 다만 이는 기존 학생 제출물 간의 문자열·구조 유사도를 분석하는 도구로, ChatGPT와 같은 생성형 AI 모델이 생성한 코드의 비결정적·비유사적 패턴을 탐지하기에는 충분하지 않다는 기존 연구의 한계가 존재한다. 이러한 배경으로 Sharif Judge는 자동 채점 시스템으로는 효과가 검증되었지만, AI 생성 코드 탐지에는 별도의 머신러닝 모델 또는 AST 기반 분석 도구가 필요함이 제기되고 있다[5].

이와 같이 기존 자동 채점 시스템의 대부분은 정답 출력 여부 중심으로 평가가 이루어지며 코드 품질(가독성, 구조, 효율성), 논리적 오류(semantic error) 등은 평가하기 어렵다는 한계가 있다. 또한 AI 생성 코드를 포함한 다양한 형태의 제출물을 구별하거나 평가하는 기능은 제한적이다.

2-4 코드 표절 탐지 연구

프로그래밍 교육에서는 학습자의 코드 표절 여부를 판별하기 위해 다양한 도구가 활용되어 왔다. 특히 스탠포드 대학에서 개발한 MOSS는 소스 코드의 토큰 구조와 문자열 유사성을 기반으로 코드 간 유사도를 계산하는 대표적 표절 탐지 도구이다[15],[16]. 또한 AST 기반 유사도 분석, 코드 스타일 특징 추출 기반 탐지, 신경망 기반 유사도 모델 등 보다 고도화된 연구도 진행되어 왔다[17]-[19].

그러나 최근 학생들이 ChatGPT 등 생성형 AI를 활용하여 과제를 작성하는 현상이 증가하면서 기존 표절 탐지 기법의 한계가 드러나고 있다. AI가 생성한 코드는 사람이 작성한 코드와 구조적·문법적 패턴이 다르며, 동일한 문제라도 다양한 형태로 작성되기 때문에 전통적인 코드 유사도 기반 탐지 도구가 AI 생성 코드를 효과적으로 탐지하지 못한다는 연구 결과가 보고되었다. 특히 MOSS는 동일한 문제를 서로 다른 프롬프트로 입력하여 생성된 코드들을 “서로 유사하지 않은 코드”로 판단하는 경우가 많아, AI 생성 코드 탐지에는 적합하지 않은 것으로 알려져 있다.

생성형 AI에 관련된 코드 표절 탐지 연구 중 중요한 연구는 Muntasir Hoq 등의 연구이다[5]. CS1(초급 프로그래밍) 수업에서 ChatGPT가 생성한 코드 제출물을 어떻게 탐지할 수 있는지를 분석한 연구로, 학생이 작성한 실제 코드와 ChatGPT가 생성한 코드 데이터를 구축한 뒤, 전통적 머신러

닝 모델(SVM, XGBoost)과 딥러닝 기반 AST 모델(code2vec, ASTNN, SANN)을 비교 평가하였다. CodeWorkout 플랫폼의 실제 학생 코드(3162개), 동일한 문제에 대해 ChatGPT가 생성한 3000개 코드 등 모두 정답 코드만 사용하여 '학생 코드 vs ChatGPT 코드'의 차이를 정밀 분석한 결과, AST 기반 모델이 ChatGPT 생성 코드를 가장 정확하게 구별했으며, 특히 SANN은 약 97%의 최고 탐지 정확도를 보였다고 보고했다. ChatGPT 코드에는 짧은 길이, 최적화된 제어 구조, 불필요한 변수 제거 등 일관된 전문가적 패턴이 나타났으며, 이는 모델이 AI 생성 코드를 식별하는 데 중요한 단서가 되었다고 한다. 반면 기존 표절 탐지 도구(MOSS)는 ChatGPT 코드 간 유사성이 매우 낮아 탐지에 효과적이지 않았다고 밝혔지만[5], 필자의 연구에서는 특정 문제에 대한 ChatGPT 생성 코드를 복수 개(3~9개)의 참조 코드 세트(reference code set)로 만든 후 ChatGPT가 생성한 다른 3개의 코드에 대해 표절을 검사하면 ChatGPT가 서로 다른 코드를 생성하게 했음에도 유사성을 발견할 수 있었다.

2-5 기존 연구의 한계 및 본 연구의 차별성

지금까지의 연구는 생성형 AI의 프로그래밍 능력, 교육 활용 가능성, 자동 채점 도구의 기능, 코드 표절 탐지 기법 등 개별 분야에서 중요한 성과를 이뤄왔다. 그러나 다음과 같은 한계가 존재한다.

- 생성형 AI를 교육의 여러 단계(교육 콘텐츠 제작 – 수업 – 평가) 관점에서 종합적으로 분석한 연구가 부족함
- ChatGPT와 Gemini 등 주요 모델 간 프로그래밍 문제 해결 능력을 비교한 정량적 연구가 부족함
- AI 생성 코드가 교육 현장에 미치는 효과(표절, 평가, 학습 방식 변화)에 대한 실험적 분석이 제한됨
- 기존 표절 탐지 도구가 AI 생성 코드를 어떻게 처리하는지에 대한 정량적·정성적 검증 연구가 부족함

따라서 본 연구는 생성형 AI를 활용한 프로그래밍 교육의 실제 적용 가능성을 다각도로 분석하고, 자동 채점 시스템과 표절 탐지 시스템이 AI 생성 코드에 대해 가지는 한계와 가능성을 실험적으로 검토함으로써 기존 연구들이 다루지 못한 교육적·기술적 공백을 보완하고자 한다. 기존 연구들은 생성형 AI의 단편적 활용 사례 또는 단일 모델 평가에 국한되었으며, 교육 콘텐츠–수업 운영–평가 여러 단계를 포괄적으로 분석하고 최신 모델(Gemini 3 Pro 등 포함) 간 성능 차이를 정량적으로 비교한 연구는 매우 제한적이었다[3],[5]-[7]. 본 연구는 생성형 AI를 프로그래밍 실습 교육 과정에 적용하는 방안을 탐색함에 있어, 교육 활동을 단계적으로 구분하여 논의한다는 점에서 ADDIE(Analysis–Design–Development–Implementation–Evaluation) 모델이 제시하는 절차적 관점과 개념적 정합성을 갖는다[8]. 본 연구에서 제안한 활용 범위는 특히 콘텐츠 설계(design) 및 개발(development), 수업 운영을 통한 적용(implementation), 그리고 학습 결과에 대한 평가(evaluation)에 해당하는 교육 활동을 중심으로 구성되어 있으며, 각 단계에서 생성형 AI가 수행할 수 있는 기능과 교육적 역할을 체계적으로 분석한다는 점에서 ADDIE 모델과 구조적 유사성을 보인다.

다만, 본 연구는 전면적인 교수설계 체계를 재구성하거나 통합적으로 구현하는 연구라기보다, 실습 중심 프로그래밍 교육 환경에서 생성형 AI의 단계별 운영 관점에서의 실천적 활용 가능성을 탐색하는 연구에 초점을 둔다. 이에 따라 요구분석(analysis) 단계와 프로그램 성과를 다음 주기로 환류하는 종합적 평가(feedback/evaluation Loop)는 본 연구의 실증 범위에서 직접적으로 다루고 있지 않으며, 이러한 부분은 후속 연구를 통해 보완되어야 할 한계로 명시한다.

Ⅲ. 프로그래밍 교육에서의 생성형 AI 활용 방안

프로그래밍 교육에서 즉각적 피드백과 예제 중심 학습의 중요성은 여러 교육 연구에서 반복적으로 강조되어 왔다[20],[21]. 생성형 AI는 이러한 교육학적 요구를 충족할 수 있는 기술적 기반을 제공한다. 본 장에서는 생성형 AI를 프로그래밍 교육에 활용하는 방법을 수업 콘텐츠 제작, 교육 과정 운영, 학습 결과 평가의 세 단계에서 체계적으로 정리하고, 실제 적용 가능성을 분석한다.

본 연구에서는 컴퓨터공학과 1학년 과목인 자바프로그래밍과 2학년 과목인 문제해결프로그래밍에 생성형 AI를 적용 가능성을 분석한다. 본 장의 논의는 생성형 AI 활용의 가능성과 운영적 특성을 개념적으로 분석하는 데 목적이 있다.

3-1 교육 콘텐츠 제작에서의 활용

교육 콘텐츠 제작 단계에서 생성형 AI는 예제 코드 생성, 실습 문제 변형, 학습 목표에 따른 문제 난이도 조절, 설명 자료 자동 생성 등 다양한 작업을 자동화할 수 있다. 이는 교수자의 초기 개발 부담을 줄이고, 학생에게 더 풍부한 학습 자료를 제공하는 데 기여한다.

1) 예제 코드 생성 및 개념 설명 보조

생성형 AI는 특정 프로그래밍 개념(예: 반복문, 조건문, 배열, 객체지향 개념)에 대한 예제 코드를 즉시 생성할 수 있으며, 개념 설명과 코드 구조를 동시에 제공함으로써 교수자의 자료 개발 시간을 절감한다. 예를 들어, "for 문을 이용한 1~10 합 계산"이라는 요청에 대해 다음과 같은 예시 코드를 자동으로 제시할 수 있다.

```
int sum = 0;
for (int i = 1; i <= 10; i++) {
    sum += i;
}
System.out.println(sum);
```

이와 같은 자동 생성 예제는 기본 문법 학습뿐 아니라 설명·시각화 자료 제작에도 활용될 수 있다.

2) 수준별 실습 문제 자동 생성

생성형 AI는 학습 목표에 따라 난이도별 실습 문제를 자동 생성할 수 있다. 이는 교수자가 다양한 수준의 학습자를 동시에 지원할 수 있도록 한다. 예를 들어 if 조건문 연습을 위한 난이도별 문제 생성 요청 시 AI는 다음과 같이 초급, 중급, 응용 단계로 구분된 문제를 구성할 수 있다.

- 초급: "두 정수를 입력받아 합을 구하는 프로그램"
- 중급: "if-else를 사용해 양수/음수 판별하기"
- 응용: "배열을 활용해 학생들의 평균 점수 계산"

3) 복습 및 평가용 퀴즈 자동 생성

생성형 AI는 특정 주제에 대한 OX 문제, 객관식 문제, 단답형 문제 등을 자동으로 생성해 학습자가 복습하고 개념 정리를 수행할 수 있도록 돕는다.

본 절에서 제시한 사례들은 동일한 학습 목표에 대해 다양한 예제 코드와 문제 변형을 생성할 수 있다는 점과, 교수자가 학습자의 수준 차이를 고려한 차별화된 콘텐츠를 설계하는 데 실질적인 지원을 제공하는 가능성을 보여준다. 다만 생성형 AI가 생성한 예제는 문법적 정확성은 확보할 수 있으나, 교육적 의도나 난이도 조절 측면에서는 교수자의 검토와 보완이 필수적이다.

3-2 교육 과정에 활용

수업 운영 단계에서 생성형 AI는 학습자에게 실시간 피드백을 제공하고, 코드 오류 분석, 코드 구조 개선, 개념 질의응답 등 다양한 기능을 수행함으로써 교수자 중심의 교육 구조를 보완한다.

1) 실시간 코드 오류 분석 및 디버깅 지원

초급 학습자는 NullPointerException, ArrayIndexOutOfBoundsException 등 기본적인 오류에서 어려움을 겪는 경우가 많다. 생성형 AI는 오류 메시지뿐 아니라 오류 발생 원인을 분석하고 구체적인 해결 방안을 제시해 학습자의 문제 해결 능력을 강화한다. 또한 논리 오류나 누락된 조건과 같은 실행 전 오류도 설명할 수 있어 디버깅 과정을 단축시킨다.

2) 개념 기반 실시간 질의응답

생성형 AI는 배열과 ArrayList의 차이, 객체지향의 상속 관계, 제네릭 개념 등 학습자가 어려움을 겪는 개념을 즉각 설명할 수 있다.

3) 코드 스타일 및 구조 개선

생성형 AI는 학생이 작성한 코드의 가독성, 중복 구조, 변수 네이밍, 제어 구조의 효율성 등 다양한 측면을 분석하여 코드 품질 향상 방안을 제시할 수 있다. 예를 들어 다음과 같은 중복 출력 구조를 포함한 코드를,

학생 코드 예시:

```
if (a > b) {
    max = a;
    System.out.println("Max value is " + max);
} else {
    max = b;
    System.out.println("Max value is " + max);
}
```

다음과 같이 간결하고 효율적인 구조로 리팩토링하도록 안내할 수 있다. 이러한 개선은 초급 학습자에게 코드 품질 개념(가독성, 중복 제거, 간결성)을 자연스럽게 학습하도록 돕는다.

AI 리팩토링 예시:

```
int max = (a > b) ? a : b;
System.out.println("Max value is " + max);
```

생성형 AI를 수업 운영 단계에 적용한 사례들은 학습 과정 지원 기능으로서 의미를 갖는다. 학생이 코드 작성 중 발생하는 오류에 대해 생성형 AI가 실시간으로 원인과 수정 방향을 제시 가능해진다. 그러나 생성형 AI의 설명은 항상 학습자의 이해 수준을 고려한 단계적 설명을 보장하지는 않으며, 경우에 따라 정답 중심의 설명에 치우칠 가능성도 존재한다. 따라서 수업 운영 단계에서의 생성형 AI 활용도 교수자의 지도와 병행될 필요가 있다.

3-3 학습 결과 평가에 활용

생성형 AI는 학습 과정뿐 아니라 학습 결과를 평가하는 과정에서도 중요한 역할을 수행할 수 있다. 전통적인 자동 채점 시스템이 주로 기능적 정확성에만 초점을 맞추고 있다. 소스코드의 내용을 자동 분석하지 못하기 때문에 각 테스트 케이스의 입력에 대해 정답 출력과 동일한 출력을 내는 것으로만 판단하는 것이 일반적이다. 반면, 생성형 AI는 코드의 구조, 가독성, 효율성, 논리적 오류 등 다양한 측면을 포괄적으로 분석할 수 있어, 보다 정교한 다면적 평가를 가능하게 한다. 아래에서는 이러한 활용 가능성을 구체적 예시와 함께 제시한다.

1) 기능적 정확성 기반의 자동 코드 평가

생성형 AI는 학생 제출물의 출력이 주어진 사양과 일치하는지를 자동으로 검사해 기능적 정확성을 평가할 수 있다. 이는 반복적이고 기계적인 채점 업무를 대체하는 데 효과적이다.

예제 프롬프트 : "다음 Java 코드가 입력값 5에 대해 25를 출력하는지 평가하라."

AI의 분석 예시 :

- 코드 실행 경로를 추론하거나,
- 예상 입력을 대입해 논리적 계산을 시뮬레이션하며,
- 출력이 요구 조건을 충족하는지 서술한다.

AI 출력 예시 : "입력값 5가 주어졌을 때 함수는 return x * x;를 수행하므로 출력은 25이며, 요구된 사양을 만족한다."

2) 코드 리뷰 및 정성적 피드백 제공

생성형 AI는 코드의 품질을 정성적으로 평가할 수 있다는 점에서 기존 자동 채점 도구와 구별된다. 가독성, 효율성, 변수명 사용의 적절성, 중복 코드 여부 등을 종합적으로 평가해 피드백을 제공할 수 있다.

예제 프롬프트 : "이 학생의 코드를 분석하고 개선할 부분을 제시하라."

학생 코드 예시 :

```
int s = 0;
for(int i=1;i<101;i++){
    s=s+i;
}
System.out.println(s);
```

AI의 피드백 예시 :

- 변수명 s는 의미가 불명확하며 sum이 더 적절함
- 1~100 합 계산은 i <= 100이 명시적임

3) 평가용 퀴즈 및 테스트 자동 생성

AI는 수업 수준에 적합한 평가 문제(객관식·단답형·프로그래밍 문제)를 자동 생성한다.

예제 프롬프트 : "초급 Java 문법 테스트 문제를 5개 생성하라."

AI 출력 예시 :

- 변수 선언에서 유효하지 않은 형태는?
- while문과 for문의 차이점은 무엇인가?
- 문자열 비교에 == 대신 equals()를 사용해야 하는 이유는?
- 아래 코드의 출력값을 구하라.

4) 점수 산출 및 루브릭 기반 자동 평가

AI는 기능적 정확성, 코드 품질, 예외 처리 여부 등 다양한 기준을 반영하여 점수를 산출할 수 있다. 특히 루브릭 기반 평가를 자동화할 수 있어 프로젝트나 응용형 과제 채점에도 적용 가능하다.

예제 프롬프트 : "아래 코드에 대해 (1) 기능적 정확성 40점, (2) 코드 품질 30점, (3) 가독성 30점을 기준으로 점수를 산출하라."

AI는 각 항목별 강점·약점을 분석하여 점수를 제시한다.

5) 오답 분석 및 학습약점 진단

AI는 학생이 반복적으로 범하는 오류 패턴을 분석하고, 학습자가 잘못 이해한 개념을 추출하여 보완 학습을 제안할 수 있다.

예제 프롬프트 : "이 학생이 반복문 문제를 지속적으로 틀리는 이유를 분석하고 추가 학습 방향을 제시하라."

AI의 분석 예시 :

- 조건식 설정 오류 빈번
- 반복 변수 초기화 실수
- 종료 조건을 이해하지 못함

→ 이를 바탕으로 반복문 개념 강화를 위한 연습 문제 3~5개 자동 생성 가능

6) 수준별 맞춤형 피드백 제공

AI는 학생의 제출물, 퀴즈 결과, 오류 패턴 등을 분석하여 개별 학습 수준에 맞는 추가 학습 자료를 제공할 수 있다.

예제 프롬프트 : "조건문 사용이 미숙한 학생에게 적절한 레벨의 연습 문제 3개를 제시하라."

AI의 출력 예시 :

- 단순 조건문
- 중첩 조건문
- 조건문 기반의 간단한 분기형 프로그램 문제

7) 자동 반복 학습 자료 생성

AI는 특정 개념에 취약한 학생에게 맞춤형 반복 학습 자료를 제공할 수 있다. 이는 학습 결손 회복에 효과적이다.

예제 프롬프트 : "배열 개념이 약한 학생을 위해 난이도별 배열 실습 문제를 3개 생성하라."

AI의 출력 예시 :

- 배열 합 구하기
- 최대값·최소값 찾기
- 배열 회전 또는 역순 출력

학습 평가 단계에서의 생성형 AI 활용은 자동 채점, 코드 분석, 피드백 제공 등 평가 과정의 보조적 기능을 수행한다. 그러나 생성형 AI 기반 평가는 코드의 기능적 정답 여부에 초점을 두는 경향이 있으며, 알고리즘적 사고 과정이나 문제 해결 전략과 같은 질적 요소를 완전하게 반영하기에는 한계가 있다.

Ⅳ. 생성형 AI의 코드 생성 정확도 분석

생성형 AI를 프로그래밍 교육에 활용할 때 고려해야 할 점 중 하나인 생성형 AI의 소스 코드 생성의 정확도 분석을 하기 위해 ChatGPT와 Gemini를 대상으로 하였고, 문제해결프로그래밍 과목의 문제들에 대해 ChatGPT 4o, ChatGPT 5, Gemini 2.5 Pro, Gemini 3 Pro를 사용하여 평가했다.

생성형 AI는 프로그램 소스 코드의 작성에도 많이 활용되고 있어, 초급 프로그래머를 빠르게 대체할 정도로 성능이 향상되고 있다. 본 연구에서는 주요 생성형 AI인 ChatGPT 4o, ChatGPT 5, Gemini 2.5 Pro, Gemini 3 Pro 버전을 대상으로 비교 평가하였다. 1학년 자바프로그래밍 과목은 문제 생성의 난이도가 크지 않기 때문에 제외했고, 2학년 문제해결프로그래밍 과목의 각 문제를 대상으로 실험했는데, 자료구조 과목을 수강한 학생들이 심화 문제 해결 능력을 키우기 위한 과목이다. 문제해결프로그래밍 과목의 주차별 주제와 문제 제목, 그리고 일부 문제는 부록에 수록되어 있는데 백트래킹, 그래프 알고리즘, 동적프로그래밍 등 어려운 주제가 포함되어

있고, 매주 2~3 문제에 대한 소스 코드 답안을 자동 평가 시스템에 제출하면 즉시 점수 피드백을 받을 수 있고, 제한된 시간 내에 여러 번 제출할 수 있다. 공개 벤치마크 데이터 세트를 사용하지 않고 문제해결프로그래밍 과목의 문제를 실험에 사용한 이유는 각 생성형 AI에 대한 코드 생성 정확도 평가와 표절 탐지가 실제 운영되고 있는 수업에 얼마나 효과적인지를 검증하기 위함이다.

4-1 생성형 AI의 버전별 정확도 평가

각 생성형 AI의 버전에 대해 1차 시도에서 모든 테스트 케이스를 통과하지 못한 경우 2차 시도까지 실험하였다. 총 문제 수는 26개이고 각 문제마다 테스트 케이스가 최대 5개까지 포함되어 있다. 문제 26개를 난이도와 유형/알고리즘에 따라 분류하면 표 1과 같다. 문제별로 테스트 케이스를 모두 통과하면 만점이고, 일부만 통과하면 부분 점수를 부여하는 방식으로 총점을 구한 후, 모두 만점을 받았을 때의 점수에 대한 백분율로 정확도를 평가하였다. 문제별로 다른 가중치가 적용될 수 있는데, 1주에 3문제가 있는 경우 가중치를 30:30:40으로 부여했고, 1주에 2문제가 있는 경우 가중치를 50:50으로 부여했다. 총 41명의 수강생이 매주 2~3 문제에 대한 코드를 제출해야 하는데, 문제 당 최대 제출 수는 41개, 평균 제출 수는 32개, 최저 제출 수는 18개였다.

표 1. 문제에 대한 분류
Table 1. Classification of the problems

Criteria		# of problems
Difficulty level	High	6
	Medium	9
	Low	11
Type / Algorithm	String Processing	7
	Abstract Data Structure	5
	Sorting	2
	Combinatorics	2
	Backtracking	4
	Graph Traversal / Algorithm	4
	Dynamic Programming	2

본 실험에 사용한 프롬프트는 다음과 같다.

프롬프트(1차) : “다음 문제에 대해 자바 소스코드를 만들어줘. (문제 설명과 샘플 입출력 삽입)”

프롬프트(2차) : “테스트 케이스 n개 중 m개만 통과했어. 코드를 다시 만들어줘.” (n은 총 테스트 케이스 수로 1~5개이고, m은 1차 시도 때 통과한 테스트 케이스 수)

위와 같이 1차 시도 프롬프트는 문제 설명과 그곳에 포함된 샘플 입출력만 제시했고, 사용할 알고리즘이라든가 방법을 추가하지 않았다. 단, 문제 설명이나 힌트에 나와 있는 것은 모두 포함하여 제시했다. 1차 시도에서 통과하지 못한 테스트 케이스가 있는 경우 총 테스트 케이스 몇 개 중 몇 개만 통과했는지를 프롬프트에 추가해서 제시했다.

표 2는 그 결과를 보여준다. 1차 시도만 평가하면 Gemini 3 Pro가 81.67%의 가장 높은 정확도를 보여주었고, 2차 시도까지 포함해도 Gemini 3 Pro가 90.00%의 가장 높은 정확도를 보였다. 만점은 1,200점이고, ChatGPT 5는 1,040점을 Gemini 3 Pro는 1,080점을 획득했다. 총 수강생 41명 중 실습 점수 1위는 1,095점, 2위는 1,038점으로 생성형 AI의 점수가 학생 1위 점수보다는 낮지만, 학생 2위 점수보다 높은 결과임을 알 수 있었다. 단, 학생들은 만점이 나올 때까지 횟수에 제한 없이 소스코드를 제출할 수 있었다. 참고로, 수강생 전체 평균은 748점, 상위 10명의 평균은 1,030점이었다.

특이한 점은 ChatGPT 4o, Gemini 2.5 Pro, Gemini 3 Pro에서는 컴파일 오류가 하나도 발생하지 않았지만, ChatGPT 5의 경우 중괄호({)를 2개씩 생성한다든가, 문장의 끝에 세미콜론(;)을 누락 하는 등 사소한 오류도 발생했다.

단, JDK 버전 문제로 인한 오류는 무시했다.

표 2. 생성형 AI의 코드 생성 정확도 평가
Table 2. Evaluation of code generation accuracy of generative AIs

Version	1st attempt only	Including 1st and 2nd attempt
ChatGPT 4o	61.67%	73.33%
ChatGPT 5	70.00%	86.67%
Gemini 2.5 Pro	75.00%	84.17%
Gemini 3 Pro	81.67%	90.00%

표 3. 난이도에 따른 획득 점수 비교
Table 3. Comparison of score obtained according to difficulty level

Criteria		# of pro-blems	Score obtained		
			Max score	ChatGPT 5	Gemini 3 Pro
Difficulty level	High	6	290	170	**220**
	Medium	9	430	310	**320**
	Low	11	480	360	**440**
	Sum	26	1,200	840	**980**

표 4. 유형/알고리즘에 따른 획득 점수 비교
Table 4. Comparison of score obtained according to type / algorithm

Criteria		# of problems	Score obtained		
			Max score	ChatGPT 5	Gemini 3 Pro
Type / Algo-rithm	String processing	7	300	150	**270**
	Abstract data structure	5	200	**110**	100
	Sorting	2	100	**70**	50
	Combinatorics	2	100	90	90
	Backtracking	4	200	200	200
	Graph traversal / Algorithm	4	200	150	**200**
	Dynamic programming	2	100	70	70
	Sum	26	1,200	840	980

Gemini 3 Pro가 테스트 케이스를 하나도 통과하지 못한 문제는 3 주차 버섯 농장2(mushroom farm2)이고, ChatGPT 5가 테스트 케이스를 하나도 통과하지 못한 문제는 11 주차 마피아 게임(Mafia game)이었다. 마피아 게임은 Gemini 2.5 Pro도 전혀 해결하지 못한 문제였는데, Gemini 3 Pro에서는 모든 테스트 케이스를 통과하는 정답 소스 코드를 생성해 내었다. 마피아 게임에 대한 설명은 부록에 수록되어 있다.

실험에 사용한 생성형 AI 4가지 버전에 대해 전체 테스트 케이스 중 일부만 통과한 문제는 3 주차 버섯 농장2(mushroom farm2)와 7 주차 아나그램(anagram), 그리고 13 주차 모래성(sand castle)이었다. 다시 말하면 4가지 생성형 AI를 1, 2차 시도까지 하더라도 모든 테스트 케이스를 통과하는 정답 소스 코드를 생성할 수 없었다. 지금까지의 실험 결과를 정리하면, Gemini 3 Pro가 4가지 버전 중 가장 우수한 성능을 보였다. 컴파일 오류도 없었고, 1차 시도만으로도 가장 높은 정확도를 보였다.

표 3은 난이도에 따른 ChatGPT 5와 Gemini 3 Pro 모델의 1차 시도 결과를 비교한 결과이다. 난이도와 관계없이 Gemini 3 Pro가 ChatGPT 5보다 높은 점수를 기록했다. 표 4는 유형/알고리즘에 따라 획득 점수를 비교한 결과를 보여주는데, 추상자료구조와 정렬 유형에서는 ChatGPT 5가 더 높은 점수를 기록했고, 문자열 처리와 그래프 순회/그래프 알고리즘 유형에서는 Gemini 3 Pro가 더 높은 점수를 획득했다. 그 밖의 유형과 알고리즘에서는 동일한 결과를 보였다.

네 가지 생성형 AI 버전(ChatGPT 4o, ChatGPT 5, Gemini 2.5 Pro, Gemini 3 Pro)의 프로그래밍 문제 해결 점수(1, 2차 시도 중 높은 점수)를 사용하여 Friedman Test를 수행한 결과 버전 간 통계적으로 유의미한 성능 차이가 확인되었다. Friedman Test 결과 통계량(χ^2 = 12.19, 자유도(df) = 3, 유의확률(p) = 0.0067이 나왔는데, 설정한 유의수준(α = 0.05 또는 α = 0.01)보다 작으므로, 생성형 AI 버전 간 프로그래밍 해결 성능에는 통계적으로 매우 유의미한 차이가 있음을 확인했다($p < .01$). 모델 간 쌍별 차이를 검증하기 위해 Nemenyi 사후 분석을 수행하였는데, 평균 순위가 가장 높았던 Gemini 3 Pro는 ChatGPT 4o와의 비교에서도 통계적으로 유의미한 차이를 보이지 않았으며(p = 0.641), ChatGPT 5(p = 0.999) 및 Gemini 2.5 Pro(p = 0.990)와의 비교에서도 통계적 유의성은 확인되지 않았다. 또한 ChatGPT 5와 ChatGPT 4o 간의 성능 차이 역시 통계적으로 유의미하지 않은 것으로 나타났다(p = 0.665). 이러한 결과는 네 모델이 다수의 문제에서 동일한 점수를 획득함에 따라 평균 순위 차이가 크지 않았고, 26개 문제라는 제한된 표본 크기로 인해 통계적으로 유의미한 결과를 내지 못했다고 해석할 수 있다. 그럼에도 불구하고 Gemini 3 Pro는 전체 문제에서 높은 점수(50점)를 획득한 비율이 가장 높고, 0점이나 10점과 같은 극단적 실패 사례가 상대적으로 적어, 다른 모델에 비해 보다 안정적인 성능 경향을 보였다. 이는 쌍별 비교에서 통계적 유의성은 확보하지 못했으나, 실제 수업 운영 및 평가 환경에서 생성형 AI의 활용 가능성을 논의하는 데 있어 실질적인 성능 차이를 시사하는 결과로 해석할 수 있다.

4-2 생성형 AI의 버전별 통과 문제와 테스트 케이스

1차 시도만을 고려했을 때의 버전에 따른 만점 통과 문제 수, 일부 통과 문제 수, 통과 테스트 케이스 수 등에 대해 살펴보면 표 5와 같다. 총 문제 수는 26개이고, 총 테스트 케이스 수는 111개이다. 괄호 안에 있는 비율은 총 문제 수와 총 테스트 케이스 수 각각에 대한 비율을 소수 둘째 자리에서 반올림한 것이다. 표 5의 결과에서도 Gemini 3 Pro가 가장 우수한 결과를 보였다.

표 5. 생성형 AI의 통과 문제와 테스트 케이스
Table 5. Problem and test case pass of generative AIs

Version	# of perfectly passed problems	# of partly passed problems	# of passed test cases
ChatGPT 4o	13 (50.0%)	21 (80.8%)	68 (61.3%)
ChatGPT 5	16 (61.5%)	20 (76.9%)	78 (70.3%)
Gemini 2.5 Pro	17 (65.4%)	23 (88.5%)	84 (75.7%)
Gemini 3 Pro	19 (73.1%)	24 (92.3%)	92 (82.9%)

V. 생성형 AI의 소스 코드 표절 탐지 방법

생성형 AI의 사용이 늘어나면서 제기된 주요 문제점 중 하나는 학생이 생성형 AI를 활용해 프로그래밍 과제에 대한 소스 코드 답안을 만들어 제출할 때의 표절 탐지 방안이다. 본 연구에서는 2025년 11월 기준 ChatGPT 최신 버전인 5.1을 사용하여 정량적으로 분석하였다. 4장의 소스 코드 생성 정확도 실험 시에는 버전 5.1이 출시 전이었고, 표절 탐지 실험 시에는 버전 5.1이 출시된 후였기 때문에 두 실험에서 사용한 ChatGPT 버전이 달랐음을 밝힌다. 소스 코드 생성 정확도 실험도 버전 5.1을 사용했다면 버전 5보다 다소 높은 정확도를 얻을 것이라고 예상하지만, 소스 코드 정확도 실험과 표절 탐지 실험은 서로에게 영향을 주지 않는 독립적 실험이어서 버전 통일을 위한 재실험을 하지 않았다.

2장의 관련 연구에서 언급한 것과 같이 Muntasir Hoq 등은 학생들과 ChatGPT가 각각 생성한 3천여 개의 코드로 딥러닝 기반 모델 훈련을 한 후 ChatGPT가 생성한 코드를 구별해낼 수 있었지만, 이에는 많은 노력이 들어가고, 버전이 업데이트되면 이전 결과를 그대로 신뢰하기에도 문제가 있다. 본 연구에서는 MOSS 등 기존 표절 탐지 시스템을 그대로 사용하면서도 정확하게 생성형 AI로 만들어 제출하는 소스 코드를 탐지할 수 있는 방법을 제안한다. 본 연구에서 제안하는 방법은 생성형 AI의 새로운 버전이나 다른 종류의 생성형 AI가 출시되더라도 짧은 시간 내에 적용할 수 있다.

5-1 제안 방법

생성형 AI는 동일 문제에 대해서도 소스 코드를 생성할 때마다 다른 결과를 내는 경우가 많지만, 다수의 생성 결과를 집합적으로 비교할 경우 구조적 유사성이 누적될 수 있음을 검증하기 위한 실험을 수행하였다. 표절 탐지 절차는 표 6에 정리했다.

본 연구에서는 가장 많이 사용하는 ChatGPT를 실험에 사용했고, 최신 버전인 5.1 버전을 사용했다. 이것을 이용하여 다음과 같은 프롬프트를 주어 각 문제에 대한 소스코드를 생성하도록 했다.

표 6. 표절 탐지 절차
Table 6. Plagiarism detection procedure

Step	Description
Step1	Build AI-generate reference code set and test code set for each problems
Step2	Build student code set
Step3	Plagiarism detection using MOSS for 3, 6, 9 AI generated reference code set, respectively
Step4	Calculate detection rates and false positive rates

프롬프트 : “다음 문제에 대해 서로 다른 자바 소스코드 12개를 만들어주는데, 클래스 이름을 w1p1_1,...,w1p1_12로 해서 zip으로 묶어줘.”

생성된 소스 코드 중 9개를 참조 코드 세트로 사용하고, 나머지 3개를 테스트용 데이터로 사용했다. 또한 문제해결프로그래밍 과목을 수강한 학생 중 실습 점수 상위 10명이 제출한 소스 코드로 검증 데이터로 사용했다. 참조 코드 세트의 수를 3개, 6개, 9개와 같이 증가시키면서 표절 탐지의 정확도를 평가했는데, 사전 실험에서 참조 코드 세트의 수가 2개 이하일 때는 표절 탐지율이 크지 않았고, 9개 이상의 경우도 탐지 성공률이 증가하지 않아 3~9개로 실험하였다. 표절 탐지의 임계값을 낮추면 많은 테스트 데이터가 표절로 탐지되지만, 생성형 AI를 전혀 사용하지 않은 학생들의 제출 소스 코드로 표절도 탐지될 수 있어 임계값에 따른 탐지 성공률과 오탐지율(false positive rate)에 대해서도 실험 결과를 얻었다.

5-2 실험 데이터

ChatGPT 5.1이 생성한 총 348개(29 문제 * 12개)의 소스 코드와 총 40명의 수강생(2학년 27명, 3학년 9명, 4학년 4명) 중 상위 10명이 제출한 288개(29 문제 * 10명 - 2)의 소스 코드를 실험 데이터로 사용하였는데, 학생 중 2명이 각각 1문제씩 제출하지 않은 주차가 있었다. 상위 10명만을 대상으로 한 이유는 문제의 난이도가 대체적으로 높아 전체 수강생을 대상으로 할 경우 학생에 따라 소스코드를 제출하지 않는 주차와 문제들이 많았고, 제대로 동작하지 않는 코드 제출도 포함되어 일정한 테스트 데이터 세트를 구축하는데 어려움이 있었기 때문이다.

총 만점은 1,200점으로, 1등은 1,100점을 획득했고, 10등은 946점을 획득했다. 모든 수강생의 평균은 574점, 중간값은 812점이었다. 평균과 중간값의 차이는 총 34명의 수강생 중 하위 8명은 과제물을 전혀 제출하지 않았거나 제출했어도 0점을 얻은 경우로 인해 발생했다. 1등은 전혀 풀지 못한 문제는 없었지만, 일부 테스트 케이스를 통과하지 못한 경우가 있어 91.67%의 점수를 획득한 것이다. 가장 잘하는 학생도 100%를 획득하기 어려울 정도로 문제의 난이도가 높은 것들이 포함되어 있다고 볼 수 있다. 본 실험에 사용한 문제는 부록의 문제해결프로그래밍 과목의 주차별 문제와 유사하나 동일한 문제는 아닌 점도 밝힌다. 이렇게 한 이유는 학생들이 제출한 소스 코드가 남아 있는 2016년 2학기의 문제를 실험 대상으로 삼았기 때문이다. 이때는 ChatGPT 등 생성형 AI가 출시되기 전이라 학생들이 제출한 코드는 사람이 작성한 코드임이 명확한 점도 밝힌다. 또한 본 연구에서 사용한 데이터 세트는 실제 수업 환경을 반영한 비공개 실습 문제이다.

5-3 MOSS 적용

2장에서 기술한 스탠포드 대학이 운영하는 MOSS를 적용했는데, MOSS에 설정한 옵션은 다음과 같다.

Options -l java -m 100 -N 10000-l은 language로 java를 선택했고, -m은 이는 흔하게 반복되거나, 보일러플레이트(boilerplate) 코드, 또는 언어의 필수적인 구조 때문에 광범위하게 사용되는 코드가 표절로 잘못 식별되는 것을 방지하기 위해서 사용하는 옵션이다. MOSS는 제출된 모든 파일 쌍 간의 유사도를 계산한 후, 유사도가 높은 순서대로 결과를 정렬하는데, -N 옵션은 이 정렬된 목록에서 상위 몇 개의 파일을 웹 결과 페이지에 보여줄지 결정한다. 본 연구에서는 그림 1과 같이 윈도우 운영체제에서 GUI 앱을 사용하여 MOSS의 표절 탐지 결과를 얻었다. 그림 2는 5 주차 문제 3에 대한 표절 탐지를 보여준다. w5p3_5.java는 ChatGPT가 생성한 5번째 소스 코드라는 것을 의미하고, a로 시작하는 파일은 학생이 제출한 소스 코드이다. 결과 6번째 줄의 w5p3_10.java (67%) w5p3_6.java (77%)는 ChatGPT가 생성한 10번째 소스 코드가 6번째 생성한 소스 코드와 67% 일치한다는 것을 의미하는데, 임계값이 67% 이하라면 표절로 탐지해 낼 수 있다.

5-4 표절 탐지 실험

MOSS를 사용하여 표절 탐지 실험을 수행했는데, 그 결과를 다음과 같이 표 7에 정리했다. 표 7은 표절 탐지를 위한 임계값에 따른 결과를 보여주는데, 임계값은 20%에서 70%까지 5% 단위로 설정했고, 참조 코드 세트 수를 3개, 6개, 9개로 변화시키면서 ChatGPT가 생성한 문제마다 3개씩의 테스트 데이터에 대해 탐지율과 학생들이 제출한 표절하지 않은 데이터에 대한 오탐지율을 구했다. 최적의 결과로 임계값이 40일 때, 9개 모델을 사용한 탐지율은 96.6%이고, 오탐지율은 0.0%를 기록했다. 다시 말하면, 생성형 AI를 사용하지 않

은 학생들의 코드를 표절이라고 판정하는 오류가 전혀 없는 상태에서도 생성형 AI를 사용하여 표절한 학생들의 코드를 96.6% 제대로 탐지해 낼 수 있었다.

그림 3은 임계값에 따른 탐지율 변화의 추이를, 그림 4는 오탐지율 추이를 꺾은선 그래프로 그린 것을 보여준다. 그림에서 참조 코드 세트 크기 3과 6의 오탐지율이 정확히 일치해서 참조 코드 세트 크기 6인 오렌지색만 표시되고, 파란색은 표시되지 않았다. 그림 3에서 보이는 것과 같이 임계값이 작을수록 탐지율이 높아지는 것을 확인할 수 있었다. 또한 참조 코드 세트 크기를 늘릴수록 탐지율이 높아지는 것도 관찰되었다. 하지만, 탐지율을 높이기 위해 지나치게 작은 임계값을 사용하면 오탐지율이 높아지기 때문에 적절한 임계값을 찾는 것이 중요하다. 단, 본 연구의 표절 탐지 실험은 동일 문제 및 유사 프롬프트 조건에서 생성형 AI가 산출하는 코드가 일정 수준의 구조적 유사성을 보인다는 가정을 전제로 수행되었기 때문에 해당 조건 내에서의 탐지 가능성을 확인하는 데 의의가 있으나, 프롬프트 변형, 모델 버전 차이, 문제 유형의 다양화 등 환경이 달라질 경우 동일한 수준으로 일반화되기 어려울 수 있다.

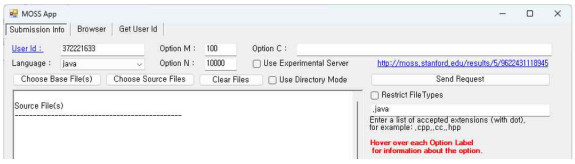

그림 1. GUI가 있는 MOSS 앱
Fig. 1. MOSS app with GUI

Moss Results

Tue Nov 18 17:21:29 PST 2025

Options -l java -m 100

[How to Read the Results | Tips | FAQ | Contact | Submission Scripts | Credits]

File 1	File 2	Lines Matched
w5p3_5.java (75%)	w5p3_7.java (82%)	42
w5p3_5.java (72%)	w5p3_9.java (62%)	38
w5p3_7.java (76%)	w5p3_9.java (60%)	38
w5p3_6.java (79%)	w5p3_7.java (73%)	34
w5p3_5.java (66%)	w5p3_6.java (77%)	35
w5p3_10.java (67%)	w5p3_6.java (77%)	33
w5p3_3.java (76%)	w5p3_4.java (77%)	36
w5p3_4.java (76%)	w5p3_6.java (74%)	35
w5p3_11.java (63%)	w5p3_9.java (53%)	33
w5p3_11.java (63%)	w5p3_7.java (67%)	33
w5p3_11.java (63%)	w5p3_5.java (62%)	33
w5p3_10.java (63%)	w5p3_7.java (66%)	29
...		
a200914548.java (50%)	a201513392.java (40%)	20
...		
a201513392.java (19%)	w5p3_12.java (14%)	7
...		

그림 2. 5주차 문제 3에 대한 표절 탐지 결과
Fig. 2. Plagiarism detection result for problem 3 of week 5

표 7. 모델 수에 따른 생성형 AI의 표절 탐지율과 오탐지율
Table 7. Plagiarism detection rate and false positive rate of generative AIs according to number of models

Threshold (%)	Detection rate (%)			False positive rate (%)		
	3	6	9	3	6	9
20	98.9	100.0	100.0	4.6	4.6	5.8
25	96.6	100.0	100.0	1.2	1.2	2.3
30	95.4	100.0	100.0	1.2	1.2	2.3
35	93.1	98.9	100.0	1.2	1.2	1.2
40	87.4	95.4	96.6	0.0	0.0	0.0
45	80.5	92.0	96.6	0.0	0.0	0.0
50	77.0	90.8	95.4	0.0	0.0	0.0
55	62.1	83.9	89.7	0.0	0.0	0.0
60	57.5	79.3	87.4	0.0	0.0	0.0
65	47.1	66.7	77.0	0.0	0.0	0.0
70	37.9	58.6	69.0	0.0	0.0	0.0

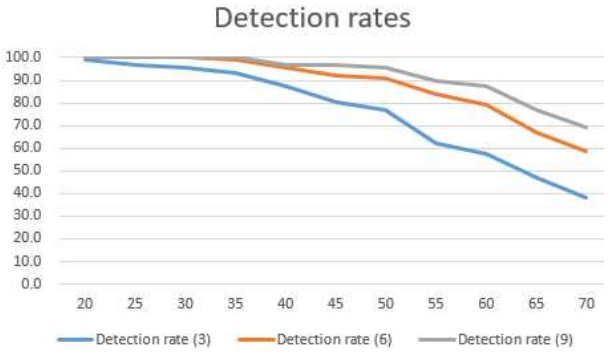

그림 3. 임계값에 따른 탐지율
Fig. 3. Detection rate according to threshold value

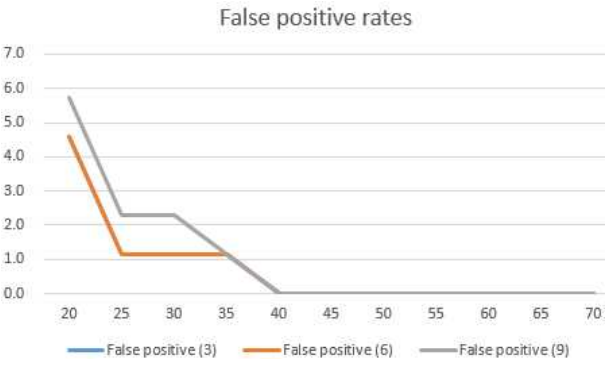

그림 4. 임계값에 따른 오탐지율
Fig. 4. False positive rate according to threshold value

5-5 기존 연구와의 비교

2-2절 관련 연구에서 기술한 바와 같이 Muntasir Hoq 등의 연구[5]도 생성형 AI를 사용하여 소스 코드를 제출할 때의 표절 탐지에 대한 것인데, 이 방식과 본 연구에서 제안한 방법의 주요 차이는 다음과 같다. 기존 연구는 생성형 AI가 작성하는 코드 스타일과 사람이 작성하는 코드 스타일에 근본적인 차이가 있을 것이라는 가정을 밑바탕에 깔고 있다. 기존 연구와 본 연구 사이에 사용된 문제도 다르고, 생성형 AI

의 버전도 달라 정확한 비교가 되기는 어렵더라도, 수천 개의 소스 코드를 딥러닝으로 학습한 기존 연구에 비해 본 연구 결과가 더 큰 효율성과 정확성을 보여준다고 볼 수 있다. 단, 기존 연구는 새로운 문제가 추가되더라도 모델 학습을 다시 할 필요가 없다는 점이 장점이다. 표 8은 기존 연구와 본 연구를 비교한 결과를 정리한 것이다. Muntasir Hoq 등의 실험 결과와 수치의 정밀도를 맞추기 위해 본 연구 결과에서도 소수점 첫째 자리에서 반올림한 수치(%) 결과를 기입했다. Muntasir Hoq 등의 최적의 실험 결과는 탐지율 97%, 오탐지율 3%이었고, 본 연구의 최적 결과는 탐지율 97%, 오탐지율 0%이어서 본 연구의 우수성을 보여준다고 볼 수 있다.

표 8. 기존 연구와의 결과 비교
Table 8. Comparison of results with existing research

	Detection rate (%)	False positive rate (%)	Remark
Muntasir Hoq (SANN)	97%	3%	Muntasir Hoq's best result
Muntasir Hoq (ASTNN)	99%	8%	
This research (threshold 40%)	97%	0%	Our suggestion
This research (threshold 35%)	100%	1%	

Ⅵ. 결 론

본 연구는 생성형 AI의 급격한 발전이 프로그래밍 교육 환경에 미치는 영향을 폭넓게 분석하고, 특히 교육의 전 주기(교육 콘텐츠 제작, 교육 과정, 학습 결과 평가)에 걸쳐 AI의 실질적인 활용 가능성을 살펴보았고, 학습 결과 평가에 활용할 때의 활용 가능성과 한계를 정량적으로 검증했다. 특히 학습 평가에서의 두 가지 핵심적인 이슈인 코드 생성 정확도와 표절 탐지 방안에 대해 특정 대학의 컴퓨터공학과 2학년 '문제해결프로그래밍' 과목을 대상으로 정량적인 실험 결과를 제시했다.

주요 연구 결과는 다음과 같다. 코드 생성 정확도 분석했는데, 최신 생성형 AI 모델 4가지 버전(ChatGPT 4o, ChatGPT 5, Gemini 2.5 Pro, Gemini 3 Pro)를 대상으로 프로그래밍 문제 해결 능력을 비교한 결과, Gemini 3 Pro가 26개 문제 중 73.1%의 문제 만점 통과율과 82.9%의 테스트 케이스 통과율을 기록하며 가장 우수한 성능을 보였다. 이는 현존하는 생성형 AI가 프로그래밍 실습 과제에 실용적으로 활용될 수 있음을 확인시켜 주었다. 또한 생성형 AI 표절 탐지 방법론을 제시했다. 생성형 AI가 비결정적 패턴으로 코드를 생성하기 때문에 기존의 표절 탐지 도구(MOSS)로는 탐지가 어렵다는 한계가 있다. 이를 보완하기 위해, 다수의 AI 생성 코드(참조 코드 세트)를 비교 데이터로 활용하는 새로운 MOSS 기반 탐지 방법을 제안하고 그 정확도를 검증했다. 실험 결과, 9개의 참조 코드 세트와 임계값 40%를 적용했을 때 탐지율 96.6%에 오탐지율 0.0%를 달성하여, AI를 사용하지 않은 학생의 코드를 오인하는 오류 없이 AI 생성 코드를 매우 높은 정확도로 탐지할 수 있음을 입증했다. 임계값을 35%로 설정했을 때는 탐지율 100.0%와 오탐지율 1.2%를 기록하여, 딥러닝 기반의 복잡한 모델(SANN)을 사용한 기존 연구의 성능(탐지율 97%, 오탐지율 3%)과 비교했을 때 더욱 뛰어나거나 유사한 결과를 단순한 방법으로 얻어낼 수 있었다.

본 연구는 생성형 AI 시대 프로그래밍 교육의 전 주기의 상당 부분에 대한 실질적인 가이드라인을 제시했다는 점에서 학술적 및 실무적 기여가 크다. 교육 콘텐츠 제작 및 학습 과정에 생성형 AI를 활용하는 방법을 제시했다. 정확도 비교 결과는 교수자가 실습 문제 및 자동 평가 시스템을 구축하는 데 필요한 성능 지표를 제공한다.

성능이 우수한 AI 모델은 학생의 개별 튜터, 코드 디버깅 보조 도구 등 학습 과정에 적극적으로 통합되어, 교수자의 부담을 줄이고 학생들의 자기 주도 학습을 지원할 수 있는 근거를 마련했다. 학습 결과 평가 및 학업 윤리 확보도 본 연구의 주요 기여인데, 생성형 AI 시대의 프로그래밍 교육에서 필수적인 학업 윤리 확보 방안을 구체적이고 정량적인 데이터를 통해 제시했다는 점에서 큰 의의가 있다. 교수자는 본 연구에서 제안된 방법을 활용하여 생성형 AI를 활용한 표절을 효과적으로 관리하고, 학생들에게는 AI를 정직하고 효과적으로 사용하는 방법을 지도하는 근거를 마련할 수 있다. 본 연구에서 새로운 표절 탐지 알고리즘을 제안한 것은 아니지만, 교육 현장에서 즉시 적용 가능한 실용적 운영 방법론을 제시했다는데 의미가 있다. 본 연구의 탐지 성과는 동일 문제 및 유사 프롬프트 조건부 결과로 해석될 필요가 있으며, 향후 연구에서는 다양한 모델·프롬프트·과제 맥락을 포함한 확장 검증이 요구된다. 또한, 최신 AI 모델의 정확도 비교는 교육 콘텐츠 제작 및 자동 평가 시스템 구축을 위한 실질적인 가이드라인을 제공한다.

후속 연구에서는 본 연구의 기술적 타당성을 기반으로 교육적 효과를 심층적으로 탐구해야 하는데, 다양한 과제 유형 확장이 필요하다. 본 연구는 텍스트 기반 과제에 집중되었으므로, 향후 GUI 설계, 클래스 구조, 멀티파일 기반 프로젝트 등 비정형적이고 복잡한 과제에 대한 생성형 AI의 성능과 평가 방안을 연구할 필요가 있다. 본 연구에서는 자바 언어에 국한해서 연구했으나, 향후 Python, C++ 등 다른 프로그래밍 언어와 다양한 난이도 수준으로 확장할 필요도 제기된다. 향후 연구에서 문제 표본 수 확대와 실험 간 AI 모델 버전 통일을 통해 연구의 신뢰도를 높임이 바람직하다. 또한 AI 콘텐츠 품질 보증 체계 구축 연구도 필요하다. 교육 콘텐츠 및 문제 자동 생성 시 발생하는 난이도의 적절성, 허위 정보 생성(hallucination) 방지 등 품질 관리를 위한 체계적인 품질 보증(QA) 프레임워크 개발이 요구된다. 전통적인 교육 설계 관점에서의 ADDIE 모델과 Dick & Carey 모델에서 제시하고 있는 분석 및 피드백까지 포함하는 교육의 전 주기(full cycle)에 대해 생성형 AI 적용에 관한 연구와, 학습 성과에

대한 정성적, 정량적 연구도 필요하다.

감사의 글

이 논문은 2025년도 강원대학교 대학회계의 지원을 받아 수행한 연구임.

부 록

본문에 싣기 어려운 내용을 부록에 정리했다. 여기에서는 문제해결프로그래밍 과목의 주차별 주제(표 9)와 문제 샘플을 보여준다.

• 문제 샘플 : 11주차 마피아게임(Mafia game)

마피아게임 문제는 그래프를 적용해야 풀 수 있는 문제로 다음과 같다.

문제 설명

마피아 게임은 참가자들 사이에 숨어 있는 마피아를 찾는 게임이다. 마피아 게임의 룰은 다음과 같다. 참가자 일부는 마피아가 되고, 마피아를 제외한 나머지 참가자는 시민이 된다. 마피아끼리는 누가 마피아인지 알지만, 시민은 누가 마피아인지 모른다. 마피아끼리는 서로를 마피아라고 지목할 수 없다. 참가자들 모두 자기 자신을 마피아라고 지목할 수 없다. 현재 N명의 살아남은 참가자들이 있고, 생존자들은 누가 마피아인지 밝혀내야 한다. 시민들은 각각 짐작을 통해 마피아를 지목하고, 마피아는 시민들을 죽여나가야 한다. 누가 마피아인지 모를 때, 이 참가자 중에 존재할 수 있는 마피아의 최대 수를 출력하라.

입력 첫째 줄에는 정수 N(3 <= N <= 10000)이 주어지며, 이후 N개의 줄이 주어지는데, K번째 줄의 값은 K번 참가자가 지목한 참가자다. 예를 들어, 3 2 1 1이라는 입력이 주어지면, 참가자 수는 3명이고,1번 참가자가 '이 사람이 마피아다'라고 지목한 사람은 2번 참가자, 2번 참가자가 '이 사람이 마피아다'라고 지목한 사람은 1번 참가자, 3번 참가자가 '이 사람이 마피아다'라고 지목한 사람은 1번 참가자라고 해석할 수 있다.

입력으로 여러 개의 테스트 케이스가 올 수 있으며 테스트 케이스 사이에는 빈 줄이 하나 삽입된다. 입력의 끝까지 각 테스트 케이스마다 한 줄씩 정수를 출력하면 된다.

[힌트] 가장 기본이 되는 규칙은 어떤 참가자를 마피아라고 가정을 했을 때, 해당 참가자가 지목하는 사람은 무조건 시민이 되는 것입니다. 또한, 이 문제는 참가자 사이에 존재하는 최대 마피아의 수를 구하는 문제로, 3번 규칙에 의해 마피아끼리는 서로를 지목하지 않으므로, 한 번도 지목당하지 않은 참가자는 무조건 마피아라고 볼 수 있습니다. 3번 규칙에서 또 다른 규칙을 유추해 볼 수 있습니다. 마피아끼리 지목하지 못하므로 마피아인 참가자가 지목당하는 경우는 시민들에게 지목당하는 경우입니다. 그러므로, 마피아와 시민 그리고 지목 대상으로 이루어진 그래프를 통해 어떤 참가자(정점)를 방문했을 때, 해당 참가자가 지목당한 횟수를 적절히 처리해 주시면 됩니다. 문제를 해결하는데 활용된 변수는 최대 마피아의 수, 참가자가 지목당한 횟수, 참가자가 지목한 다른 참가자, 방문한 정점 확인 등입니다. 위에 나열된 조건들을 전제로 문제를 푸시면 됩니다.

표 9. 문제해결프로그래밍 과목의 주차별 주제
Table 9. Weekly topics of problem solving programming

Week	Topics	Problems
1	Introduction	counting characters decimal vs hexadecimal 3n+1 variation
2	Data structure1	alphabet calculator mushroom farm
3	Data structure2	card extraction mushroom bomb mushroom farm2
4	Strings1	anagram finding Timo
5	Strings2	alphanumeric/Hangul sorting code breaking
6	Sorting	KDA mushroom sorting
7	Combinatorics	look-and-say sequence anagram
8	Midterm exam	written exam
9	Backtracking1	Timo prime number card play
10	Backtracking2	Knight game contest
11	Graph traversal	Mafia game old Timo
12	Graph algorithms	shortest path grasshoper catching
13	Dynamic programming1	canyon escape sand castle
14	Final exam	written exam

입력

```
3
2
1
1

3
2
3
1
```

출력

```
2
1
```

참고문헌

[1] S. Kim. “Developing Code Generation Prompts for Programming Education with Generative AI,” *The Journal of Korean Association of Computer Education*, Vol. 26, No. 5, pp. 107-117, 2023. https://doi.org/10.32431/kace.2023.26.5.009

[2] S. Sarsa, P. Denny, A. Hellas and J. Leinonen, “Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models,” in *Proceedings of the 2022 ACM Conference on International Computing Education Research (ICER’22)*, Lugano: Switzerland, pp. 27-43, August, 2022. https://doi.org/10.1145/3501385.3543957

[3] H.-J. Han, “Exploring Instructor Roles in Operating and Integrating ChatGPT into Classroom in Higher Education,” *Journal of Digital Contents Society*, Vol. 25, No. 2, pp. 465-474, February 2024. http://dx.doi.org/10.9728/dcs.2024.25.2.465

[4] R. Yilmaz and F. G. K. Yilmaz, “The Effect of Generative Artificial Intelligence (AI)-based Tool Use on Students’ Computational Thinking Skills, Programming Self-Efficacy and Motivation,” *Computers and Education: Artificial Intelligence*, Vol. 4, June 2023. https://doi.org/10.1016/j.caeai.2023.100147

[5] M. Hoq, Y. Shi, J. Leinonen, D. Babalola, C. Lynch and T. Price, “Detecting ChatGPT-Generated Code Submissions in a CS1 Course Using Machine Learning Models,” in *SIGCSE 2024: Proceedings of the 55th ACM Technical Symposium on Computer Science Education*, Portland: OR, pp. 526-532, March 2024. https://doi.org/10.1145/3626252.3630826

[6] Y. Lee, J. Kim and C. J. Park, “Development of a GPT-based Multiple-Choice Question Generation Program for the Programming Area in High School Informatics Subject,” *Journal of The Korean Association of Information Education*, Vol. 28, No. 4, pp. 473-484, 2024. https://doi.org/10.14352/jkaie.2024.28.4.473

[7] E. L. Ouh, B. K. S. Gan, K. J. Shim and S. Wlodkowski, “ChatGPT, Can You Generate Solutions for My Coding Exercises? An Evaluation on Its Effectiveness in an Undergraduate Java Programming Course,” in *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education (ITiCSE 2023)*, Turku: Finland, pp. 54-60, 2023. https://doi.org/10.1145/3587102.3588794

[8] M. Molenda, “In Search of the Elusive ADDIE Model,” *Performance Improvement*, Vol. 42, No. 5, pp. 34-36, May 2003. https://doi.org/10.1002/pfi.4930420508

[9] W. Dick, L. Carey, and J. O. Carey, “The Systematic Design of Instruction: Origins and Key Principles of the Dick and Carey Model,” *Educational Technology Research and Development*, Vol. 50, No. 4, pp. 5-23, December 2002.

[10] Google. Google Gemini Release Note [Internet]. Available: https://gemini.google/release-notes/.

[11] A. Pande, R Patil, R. Mukkemwar, R. K. Panchal and S. B. Bhoite, “Comprehensive Study of Google Gemini and Text Generating Models: Understanding Capabilities and Performance,” *Grenze International Journal of Engineering and Technology,* pp. 856-863, 2024.

[12] M. Omar, S. Nassar, K. Hijazi, B. S. Glicksberg, G. N. Nadkarni and E. Klang, “Generating Credible Referenced Medical Research: A Comparative Study of OpenAI's GPT-4 and Google's Gemini,” *Computers in Biology and Medicine*, Vol. 185, 2025. https://doi.org/10.1016/j.compbiomed.2024.109545

[13] C. D. Kloos, C. Alario-Hoyos, I. Estévez-Ayres, P. Callejo-Pinardo, M. A. Hombrados-Herrera, and P. J. Muñoz-Merino, “How can Generative AI Support Education?,” in *Proceedings of 2024 IEEE Global Engineering Education Conference (EDUCON)*, Kos Island: Greece, 2024, pp. 1-7, https://doi.org/10.1109/EDUCON60312.2024.10578716

[14] GitHub. Sharif Judge Download [Internet]. Available: https://github.com/mjnaderi/Sharif-Judge.

[15] Professor Alex Aiken's website. A System for Detecting Software Similarity [Internet]. Available: https://theory.stanford.edu/~aiken/moss/.

[16] A. Aiken, MOSS, a System for Detecting Software Plagiarism, Stanford University, Stanford: CA, Technical Report, January 2002.

[17] X. Li and X. J. Zhong, “The Source Code Plagiarism Detection Using AST,” in *Proceedings of 2010 International Symposium on Intelligence Information Processing and Trusted Computing*, Huanggang: China, 2010, pp. 406-408. https://doi.org/10.1109/IPTC.2010.90.

[18] S. Arabyarmohamady, H. Moradi and M. Asadpour, “A Coding Style-Based Plagiarism Detection,” in *Proceedings of 2012 International Conference on Interactive Mobile and Computer Aided Learning (IMCL)*, Amman: Jordan, 2012, pp. 180-186, https://doi.org/10.1109/IMCL.2012.6396471

[19] E. M. Hambi and F. Benabbou, “A Deep Learning Based Technique for Plagiarism Detection: A Comparative Study,” *IAES International Journal of Artificial Intelligence (IJ-AI)*, Vol. 9, No. 1, pp. 81-90, March 2020. http://doi.org/10.11591/ijai.v9.i1.pp81-90

[20] A. Venables and L. Haywood, “Programming Students NEED Instant Feedback!,” in *Proceedings of The Fifth Australasian Conference on Computing Education*, Adelaide: Australia, pp. 267-272, January 2003.

[21] J. Roelle, S. Hiller, K. Berthold and S. Rumann, “Example-Based Learning: The Benefits of Prompting Organization before Providing Examples,” *Learning and Instruction*, Vol. 49, pp. 1-12, 2017. https://doi.org/10.1016/j.learninstruc.2016.11.012

하진영(Jin-Young Ha)

1987년 : 서울대학교 컴퓨터공학과 (공학사)

1989년 : KAIST 전산학과 (공학석사-인공지능)

1994년 : KAIST 전산학과 (공학박사-인공지능)

1994년～1997년: ㈜ 핸디소프트 기술연구소 책임연구원

2000년～2001년: IBM T. J. Watson Research Center 방문연구원

1997년～현 재: 강원대학교 컴퓨터공학과 교수

※관심분야 : 인공지능, 패턴인식, HCI