ASE 2025 Trip Report

Seoul, South Korea

고려대학교 소프트웨어 분석 연구실

신지호

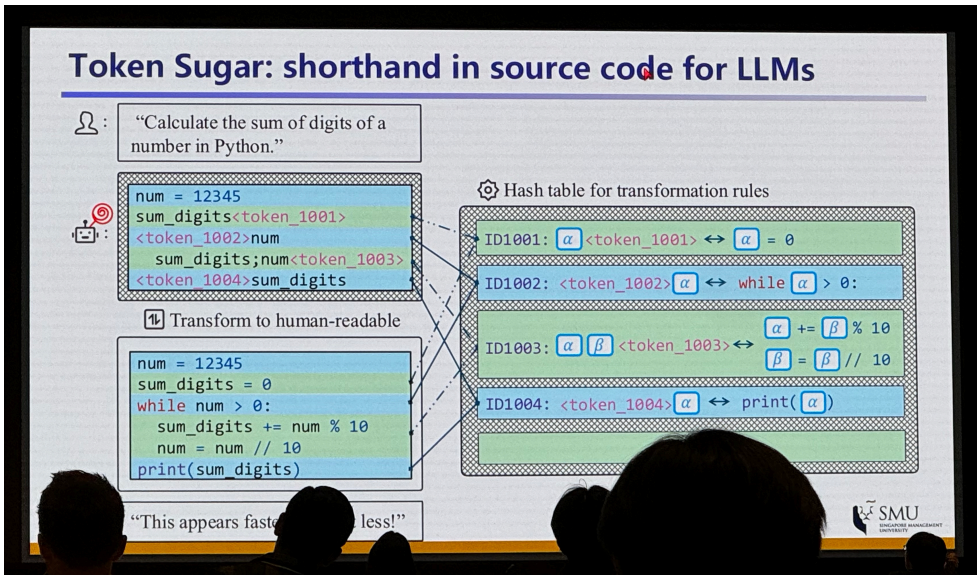
2025.11.17 ~ 2025.11.19

소개

ASE(Automated Software Engineering) 학회는 소프트웨어 개발 과정 전반을 자동화하기 위한 아이디어를 논의하는 자리이다. 분석·설계·구현부터 테스트, 유지보수에 이르기까지 복잡한 소프트웨어 시스템을 다루는 다양한 자동화 기법들이 소개되는 행사다. 올해 ASE는 서울 그랜드 워커힐에서 열렸다. 광나루역에서 행사장으로 걸어가는 길에 보이는 한강 풍경이 예뻤고, 워커힐 주변 산책길에는 아직 단풍이 남아 있어 점심시간에 산책하기 좋았다. 오프닝에서는 올해 ASE 제출 논문 수가 역대급으로 많았다는 이야기가 있었다. 제출 규모가 커지면서 논문 심사와 선정을 담당하는 인원도 예년보다 늘어났고, 행사 운영에도 약간의 변화가 있었던 것 같다. 특히 포스터 세션의 비중이 커져 시간이 더 넉넉하게 배정된 반면, 발표 시간은 이전보다 짧아진 듯하다. 한국에서 열린 행사라 한국인 참가자가 많았는데, 전체적으로는 중국 참가자가 가장 많고 그다음이 한국이라고 했다.

기억에 남는 연구들

Token Sugar: Making Source Code Sweeter for LLMs through Token-Efficient Shorthand



이 연구는 코드에서 반복적으로 등장하는 의미적 패턴을 추출하고, 이를 가역적인 token-efficient shorthand로 압축해 LLM이 더 적은 토큰으로도 동일한 의미를 처리할 수 있도록 하는 방법을 제안한다. 핵심적인 아이디어는 표현 방식이 조금씩 달라도 의미가 동일한 구조들을 일관된 짧은 표기로 묶어 모델에 학습시키는 것으로, 이를 통해 코드의 길이를 줄이면서도 성능 저하를 줄이려는 시도라고 볼 수 있다.

다만 이런 접근이 정말로 LLM에게 "달콤함"을 제공하는지는 조금 조심스러운 부분이 있다. 토큰 수가 줄어드는 것은 분명한 장점이지만, 동시에 모델이 새롭게 학습해야 하는 변환 규칙의 양이 많아지기 때문이다. 특히 모델이 기존의 일반 코드(no-sugar 데이터)로 사전학습된 상태에서 이후 sugarized 데

이터를 통한 continual pre-training을 추가로 진행하는 구조라면, 결과적으로 모델의 코드 이해 과정에서 인지적 부담이 증가하는 것은 아닐까 하는 생각이 든다. 이와 관련해 자연스럽게 이어진 궁금증은, 모델이 실제 추론 과정에서 일반 코드 ↔ shorthand 적용 코드 간의 변환을 수행하는지, 아니면 shorthand 적용 언어 자체를 하나의 독립적인 언어로 받아들이고 그 위에서 직접 reasoning을 하는지에 대한 것이다. 실험 결과에 따르면 평균적으로 11.2%의 토큰 절감이 이루어졌음에도 코드 생성 정확도가 베이스라인과 비슷하게 유지되었다고 하는데, 이 정도 성능 보존이 가능했다면 어느 정도 규칙이 내재화된 것으로 볼 수 있을지도 모르겠다.

이 주제는 예전에 SIGPL 여름학교 난상 토론에서 나온, "사람에게는 어렵지만 LLM에게는 읽고 쓰기 좋은 프로그래밍 언어가 존재할 수 있을까?"라는 질문과도 맞닿아 있다는 생각이 든다. 당시 한 교수님께서 이에 대해 다소 비관적인 답변을 하셨던 기억이 어렴풋이 남아있다. 만약 token sugar처럼 사람에게는 다소 난해하지만 모델에게는 효율적인 표현 체계가 실제 성능 향상에 기여한다면, 이 질문을 다시 생각해볼 근거가 될 수 있을 것 같다. 또한 shorthand는 본질적으로 사람이 규칙을 모르고는 해석하기 어려운 압축 표현 체계인데, 이 변환 규칙이 공개되지 않고 폐쇄적으로 유지될 경우에는 일종의 암호화 같이 될 것 같은 느낌이 든다. LLM이 얼마나 복잡한 규칙까지도 자연스럽게 습득하고 유창하게 사용할 수 있는지 역시 앞으로 더 탐구해볼 만한 지점으로 보인다.

LLM-Assisted Synthesis of High-Assurance C Programs

LLM을 활용해 명세를 기반으로 C 코드 후보를 생성하고, 이후 Rocq 위에서 동작하는 프로그램 검증 프레임워크인 Verified Software Toolchain(VST)으로 해당 코드의 정확성을 보증하는 파이프라인을 제안한 연구이다. LLM이 생성하는 코드를 검증 도구가 처리 가능한 형태로 제한하고, 심볼릭 도구를 활용한 검증 과정에서 풀리지 않는 증명 목표(goal)에 대해 LLM의 조력을 받는 하이브리드 검증 전략을 구성한다.

먼저 코드 생성 단계에서 두 가지 syntactic 제약을 둔다. 하나는 명세가 없는 함수를 호출하지 않는 것, 다른 하나는 루프를 허용하지 않는 것이다. 이는 VST가 다루기 까다로운 구조를 사전에 제거해 검증 과정의 복잡성을 줄이기 위한 설계이다. 검증 단계에서는 SEPAUTO라는 symbolic reasoning 엔진이 VST의 proof obligation을 자동으로 단순화하거나 해결을 시도한다. SEPAUTO가 목표를 해결하지 못한 경우, LLM이 개입해 적용 가능한 tactic 후보를 제안하거나, 경우에 따라 해당 목표가 사실상 해결 불가능하다고 판단해 상위 목표로 backtracking하도록 유도한다.

포스터 세션에서 저자분께 코드 생성 단계의 두 가지 syntactic 제약 외에 검증을 더 쉽게 만들 수 있는 방법이 있을지 여쭤봤는데, spec annotation 외에는 마땅한 수단이 없을 것이라는 답을 들었다. 또 tactic 적용 과정에서 progress를 어떻게 판단하는지 질문했을 때, 증명 목표나 상태가 어떤 형태로든 변화만 하면 진전으로 간주하는 기준을 사용했다고 설명해주셨다. 실제 실험 과정에서 의미 없는 상태 변환이 반복되는 사이클이 관찰된 적이 있었으나 이에 대한 해결 방안을 마련하지는 못했다고 하셨다.

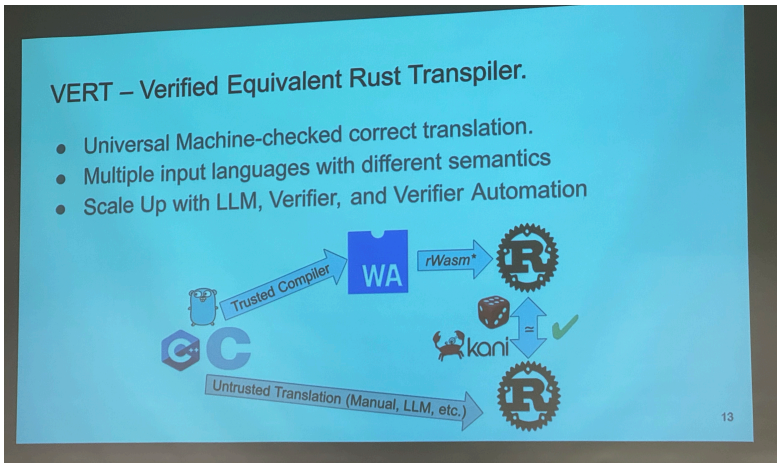
Automated Repair of Ambiguous Problem Descriptions for LLM-Based Code Generation

LLM을 활용해 자연어 설명으로부터 코드를 생성할 때, 모호한 자연어 설명에 대한 대응을 어떻게 하는 것이 적절할지에 대해 궁금증이 있었어서 이 연구에 대한 포스터 설명을 흥미롭게 들을 수 있었다. 이 연구는 자연어 설명의 애매한 부분을 포착하고 이를 자동으로 보완하기 위한 절차를 제안한다. 파이프라인의 흐름은 다음과 같다.

먼저 LLM이 자연어 설명과 함께 주어진 예제들을 바탕으로 테스트 입력들과 후보 프로그램들을 생성한다. 이후 LLM이 생성한 테스트 입력들에 대한 출력 결과를 기준으로 코드들을 클러스터링한다. 이렇게 만들어진 각 클러스터에 대해, 처음에 설명과 함께 주어진 input-output 예제를 얼마나 충족하는지를 나타내는 example consistency와 클러스터의 확률적 특징 (전체 후보 코드 중 각 클러스터에 배정된 코드의 비율)을 측정한다. 이들 중 가장 큰 example-consistent 클러스터 (모든 예제를 만족하는 클러스터 중 가장 많은 프로그램이 포함된 클러스터)를 "의도에 가장 가까운 의미"를 반영한 코드 집합으로 선택한다. 선택된 클러스터의 프로그램들과 제외된 클러스터의 프로그램들을 LLM에게 함께 제시해, 그 차이를 대조적으로 분석하도록 한다. 이 비교 과정에서 LLM은 원래의 자연어 설명에서 어떤 부분이 애매했는지를 추론하고, 그 모호함을 해소한 형태로 자연어 설명을 수정한다. 만약 모든 예제를 만족한 클러스터가 없다면, 예제를 기반으로 코드를 수정하도록 한 후 앞의 과정을 반복한다.

이 연구에서는 코드가 사용자의 의도에 정확히 부합하는지 판단하기 위해 처음에 제공되는 input-output 예제가 핵심적인 역할을 한다. 실험에서 사용된 코드 생성 벤치마크는 코드의 functional correctness를 판단할 수 있을 만큼 충분한 양과 질의 테스트케이스를 포함하기 때문에 제안된 접근법의 효과를 제대로 보일 수 있었을 것이다. 실제 자연어 기반 코드 생성 시나리오에서는 사용자가 제공하는 예제가 자연어 설명의 애매함을 걸러낼 만큼 충분히 제공되지 않는 경우가 있을 텐데, 그런 상황이라면 설명이 어떤 부분에서 여러 의미로 해석될 수 있는지 LLM이 사용자에게 알려주고 선택을 유도하는 방식이 현실적인 보완책이 되지 않을까 싶다.

VERT: Polyglot Verified Equivalent Rust Transpilation with Large Language Models



다양한 언어로 작성된 코드를 Rust로 변환하는 문제를 다루는 연구이다. 이를 위해 먼저 원본 프로그램을 WebAssembly로 컴파일한 후에 그 중간 표현을 Rust 코드로 역변환하는 compile-then-lift 방식을 사용하여 reference가 되는 코드를 생성한다. 이 방식은 규칙 기반이기 때문에 semantic equivalence를 보장할 수 있으나, 역변환된 Rust 코드는 중간 표현의 구조를 그대로 반영하기 때문에 고수준 언어보다는 어셈블리와 유사한 형태를 가지게 된다.

이 연구에서는 LLM을 이용해 보다 자연스러운 형태의 Rust 코드 후보를 생성하고, 이 코드가 compile-then-lift 방식으로 얻은 코드와 semantic equivalence를 만족하는지 검증한다. 이를 위해 property-based testing(PBT)과 model checking을 함께 활용해 두 프로그램의 동작을 비교한다. 검증 과정에서 반례가 발견되면, 해당 반례를 포함하여 LLM을 다시 프롬프트하여 변환 결과를 수정하도록 한다. 이 절차를 반복하며 점진적으로 변환 결과를 개선하는 구조를 취한다.

발표 후반부를 들으면서 개인적으로 상당히 익숙하다는 느낌을 받았다. 이번 학기 프로그램 분석 수업 프로젝트에서, 특정 code transformation이 의미를 보존하는지 확인하기 위해 differential testing과 bounded symbolic execution을 이용해 반례를 찾고, 발견된 반례로 transformation rule을 고치는 방식으로 실험했었는데, 연구에서의 접근 방식과 구조적으로 매우 유사했기 때문이다.

Agentic Specification Generator for Move Programs

이 연구는 스마트 컨트랙트 검증용 언어인 Move로 작성된 프로그램에 대해, 해당 프로그램의 명세를 자동으로 생성하는 방법을 제시한다. Move에서는 Move Specification Language(MSL)을 통해 프로그램의 동작을 형식적으로 기술할 수 있는데, 저자들은 명세를 loop invariants, abort conditions, global modification markings(modifies clauses), axiomatic semantics for state changes (ensure clauses)라는 네 종류의 절로 구분하고, 각 절을 전담하는 ClauseGen 에이전트들이 개별적으로 생성하도록 구성한다. 이후 Ensembler 에이전트가 네 종류의 절을 합쳐 하나의 명세로 정리한다.

통합된 명세가 Move Prover에서 검증에 실패할 경우, Prover가 제공하는 진단 정보, 에러 요약, 흔히 발생하는 오류에 대한 가이드, 반례 등을 프롬프트에 포함해 명세를 다시 생성하도록 한다. 반대로 검증이 성공했을 때에는, 해당 명세가 실제로 complete한지 확인하는 절차를 거친다. 명세가 complete하다는 것은 명세가 프로그램의 실행 의미를 모두 담고 있어야 한다는 것이다. 저자들은 이를 확인하기 위해 specification coverage라는 메트릭을 제안한다. complete한 명세라면 코드의 일부를 무작위로 삭제해 잘못된 프로그램을 만들었을 때 검증이 되면 안 된다. 삭제된 코드가 있음에도 검증이 통과한다면, 그 부분은 현재 명세가 포착하지 못한 의미이며, 명세가 약하다는 뜻이 된다. 이런 방식으로 명세가 프로그램의 의미를 얼마나 정확하게 커버하는지를 평가하여 complete한 명세 생성을 유도한다.

마치며

발표는 10분 단위로 지나가고 트랙은 대여섯 개씩 병렬로 진행되니, 원하는 발표들을 들으려면 바삐 돌아다녀야 했다. 그래도 그 와중에 다채로운 주제를 구경하는 재미가 있었고, 근 몇 달 동안 내가 둘러봤던 주제들을 다루는 연구들을 꽤 볼 수 있어서 더 좋았다.

점심 도시락이 맛있어서 점심시간이 되면 괜히 기분이 좋아졌고, 뱅퀫에서는 ASE 40주년 기념으로 누가 더 오래된 ASE 기념품을 갖고 있는지 겨루는 귀여운 이벤트가 있었다. 정확히 기억나진 않지만 나보다 오래된 물건이 있었던 것 같기도 하다. 오래된 물건들의 사진을 보면서 어떻게 한 분야를 그렇게 오래 연구할 수 있는 건가 싶었다. 이 행사에서 수상하신 분들의 연구실이나 집에는 어떤 물건들까지 있을지도 궁금해졌다. 또 한국 전통 음악을 기반으로 활동하는 가수 한 분이 초청되어 노래를 들을 수 있었다. 나에게도 그리 익숙하지 않은 느낌이었는데, 외국인들에게는 또 어떻게 들렸을지 모르겠다.

ASE가 아시아에서 열린 건 이번이 세 번째라고 하는데, 생각보다 횟수가 적어서 약간 놀랐다. 올해는 PLDI와 ASE가 둘 다 한국에서 열려서, 내가 특별히 준비한 것 없이 국제 학회 두 개를 연달아 경험하는 수혜를 입었다. 그러고 보니 이번 ASE가 내 첫 SE 학회이기도 했다. 한국 연구자분들의 꾸준한 활동 덕분에 이런 기회를 얻은 셈이라 감사했고, 나도 앞으로는 더 열심히 해야겠다는 생각을 하게 됐다.


근처 풍경 😎


비스타홀 가는 길에 있는 공들


단풍 든 담쟁이 덩굴?


은행나무 잎 나비
by. | ㅁ령언ㄴ |


핏자 😋