2026년 오픈소스 개발자대회 결과보고서

항 목	내 용	항 목	내 용
팀 명	Nostalgian	팀 인 원 (팀장 포함)	1명
참가부문	학생	과제유형	자유과제

□ 결과보고서

프로젝트 개요	
프로젝트명	커널 상태와 출력을 재접속 시에도 복원하는 파이썬 REPL 노트북
프로젝트 등록 URL	https://github.com/rnoro/tithon
시연 영상	https://youtu.be/4b_HGs-fmow
프로젝트 소개	로컬 또는 원격 GPU 환경에서 커널 상태와 실행 결과를 보존해, 연결이 끊기거나 재시작되어도 작업 세션을 그대로 복원하고 다른 사용자와 공유할 수 있는 파이썬 REPL 노트북
프로젝트 세부 내용	
개발 배경 및 목적	인공지능 연구에 있어 파이썬 REPL(Read-Eval-Print Loop) 노트북 환경은 스크립트 실행에 비해 셀 단위 실행, 출력 시각화, 노트북 실행결과 공유 등 많은 장점을 가집니다. 특히 코드 수정이 필요할 때 전체 코드를 다시 실행할 필요 없이 수정된 셀부터 다시 실행함으로써 시간을 절약할 수 있어, 파라미터를 끊임없이 바꿔가며 실험해야 하는 연구자들에게는 필수적인 도구입니다. 학습 상태를 실시간으로 시각화하며 모니터링할 수 있다는 점에서도 Jupyter Notebook/Lab은 오랫동안 연구자들에게 사랑받아왔습니다. 그러나 현업에서는 원격 GPU 서버에서 장시간 학습이 필요한 상황에 JupyterLab이 아니라 스크립트 실행으로 학습 코드를 실행하는 모순이 발생합니다. 장시간 학습에는 무엇보다 안정성이 중요한데, 로컬 PC에서 SSH 터널로 원격 GPU 서버에 접속해 JupyterLab을 사용하는 일반적인 환경에서는 터널 연결이 끊기거나 로컬 클라이언트가 재시작되면 그동안 쌓인 출력과 위젯 상태가 날아가, 며칠씩 걸린 학습 결과를 처음부터 다시 실행해야 하는 문제가 있기 때문입니다.

	그래서 연구자들은 어쩔 수 없이 학습 코드를 스크립트로 실행하고 W&B나 별도 로그 파일로 상태를 모니터링하지만, 이는 노트북만큼 편리하지 않습니다. 저 역시 같은 문제를 겪으면서, 클라이언트 문제가 생겨도 재접속 시 커널 상태와 출력이 복구되는 REPL 노트북 환경의 필요성을 느꼈습니다. 이 문제 외에도, 최근 AI 에이전트를 활용한 연구가 늘면서 또 다른 문제를 발견했습니다. ipykernel은 애초에 사람이 직접 셀을 실행하고 눈으로 출력을 확인하는 것을 전제로 설계되어 있어, AI가 여러 세션을 관리하거나 상태를 구조적으로 조회하기에는 부족한 부분이 많았습니다. 그래서 단기적으로는 JupyterLab의 커널 상태·출력 손실 문제를 해결하는 것을 목표로 삼았고, 장기적으로는 작업 스케줄링, 구조화된 상태 조회, 리소스 제어 등 AI 네이티브 기능이 부족한 ipykernel을 다른 커널로 대체할 필요가 있다고 판단했습니다. 이번 구현에서 진행한 단기 개발 목표는 다음과 같습니다. - 클라이언트 문제가 생겨도 재접속 시 커널과 출력 복구 - 낮은 진입장벽을 위해 JupyterLab과 유사한 사용 경험 제공 - .py 파일로 스크립트 실행, REPL 노트북 실행 둘 다 가능한 환경 - 메타데이터(셀 상태, 출력 등)를 보존한 상태로 노트북 공유 가능 - 여러 연구자 또는 AI 에이전트가 같은 파일, 세션을 동시에 다룰 수 있는 다중 클라이언트 협업 환경
개발 환경	□ 하드웨어 구성 & 운영체제 - 개발 및 단위 테스트용: Mac mini M4 32GB & MacOS 26.6 - E2E(End-to-End) 테스트용 서버: Intel Xeon E5-2650, NVIDIA Tesla P100 PCIe 16GB×8 & Ubuntu 22.04.5 LTS □ 개발 도구 - VS Code □ 사용 언어 - Python (Daemon), TypeScript (VS Code Extension)
시스템 구성 및 아키텍처	- 전체 구조: 로컬 또는 원격 GPU 호스트에서 동작하는 ipykernel, Python 기반 Tithon Daemon, TypeScript 기반 VS Code Extension의 3계층 구조 - VS Code Extension: .py 파일의 # %% 셀을 노트북 화면으로 변환하고, 실행 요청 전송 및 출력·진행 상태·위젯 렌더링 담당 - Tithon Daemon: 파일별로 독립된 세션과 실행 대기열을 관리하며, Jupyter ZMQ 프로토콜로 커널과 통신 - 커널 관리: ipykernel을 데몬과 분리된 프로세스로 실행하고 접속 정보를 보존하여, 클라이언트 또는 데몬 재시작 후에도 기존 커널에 재접속 - 출력 저장: 커널 메시지를 SQLite WAL 저널에 기록하고, 현재 화면

	은 실행별 스냅샷으로 축약해 빠르게 복원 - 대용량 데이터 관리: 이미지 등 리치 출력은 별도 파일로 저장하고, 위젯은 최신 상태를 Daemon에 미러링 - 동기화 방식: 단조 증가하는 순번을 기준으로 최초 접속 시 스냅샷, 이후 변경분만 전송하여 여러 클라이언트의 실행 결과를 동기화 - 통신 및 보안: VS Code Extension과 Daemon은 권한이 제한된 Unix 도메인 소켓으로 통신하며 별도 외부 포트를 노출하지 않음
프로젝트 주요 기능	JupyterLab과는 달리 파이썬 커널의 정보와 출력을 호스트 서버에 저장해, 클라이언트에 문제가 생기더라도 커널 상태와 출력을 복구하여 실행 중이던 세션을 계속 사용할 수 있다는 것이 프로젝트의 주요 기능입니다. 이를 위해 VS Code 확장 형태의 클라이언트 렌더러와, 커널 상태·출력을 저장하고 클라이언트로 전송하는 Daemon을 새로 개발했으며, .ipynb 대신 .py 파일 하나로 스크립트 실행과 노트북 실행이 모두 가능하도록 JupyterLab과 차이를 두었습니다. .ipynb은 JSON 구조로 코드와 실행결과, 메타데이터를 함께 담을 수 있다는 장점이 있지만, Git diff가 불편하고 셀 실행마다 달라지는 메타데이터로 불필요한 변경사항이 추적되거나 merge conflict가 자주 발생하는 등 단점이 많습니다. 게다가 AI 에이전트로 연구를 진행할 때 .ipynb의 JSON 구조와 base64 인코딩 이미지, 긴 출력이 그대로 입력에 들어가면 토큰과 비용 낭비로 이어지기 때문에 .ipynb 사용을 배제했습니다. 다만 노트북 실행 결과를 공유하는 기능은 필요하므로, 최종 실행 결과를 프로젝트 root에 .tithon/*.json으로 별도 저장해 해당 폴더만 공유하면 다른 사용자도 같은 노트북 화면을 그대로 볼 수 있도록 구현했습니다. Tithon Daemon은 PyPI에, VS Code Extension은 Marketplace에 배포되어 있어 pip install tithon과 확장 설치만으로 바로 사용 가능합니다. VS Code에서 .py 파일을 notebook view로 연 다음 평소 Jupyter Notebook처럼 사용하면 되며, 개발 목적으로 테스트하고 싶은 경우 프로젝트 폴더의 테스트 코드와 스크립트를 사용하면 됩니다.
기대효과 및 활용 분야	Tithon을 사용하면 인공지능 연구자들이 더는 장시간 학습 시 안정성을 이유로 편리한 REPL 노트북 환경을 포기하지 않아도 됩니다. 이는 연구자들이 더 적은 시간으로 더 뛰어난 연구 결과를 만들어낼 수 있는 밑바탕이 되어 줄 것입니다. 특히 원격 GPU 서버에서 대규모 학습을 자주 돌리는 연구실이나 개인 연구자에게 유용할 것으로 기대합니다. 연결이 끊겨도 커널과 출력이 그대로 보존되기 때문에, 스크립트와 로그 파일에 의존하던 기존의 타협 없이 셀 단위 실행과 즉각적인 시각화의 이점을 그대로 누릴 수 있습니다.

	또한 늘어나는 AI 에이전트 기반 연구 워크플로우에서도 활용 가능성이 있습니다. .py 파일 기반으로 동작해 불필요한 메타데이터나 인코딩된 이미지로 인한 토큰 낭비 없이 코드 자체에 집중할 수 있고, 구조화된 상태 조회를 통해 에이전트가 여러 실험을 동시에 관리하는 작업도 한결 수월해질 것입니다. 마지막으로, 셀 상태와 출력을 메타데이터와 함께 보존한 채로 공유할 수 있다는 점은 공동 연구나 랩 단위 협업에도 도움이 될 것으로 기대합니다. .tithon 폴더만 전달하면 동료가 동일한 노트북 화면을 그대로 재현해서 볼 수 있어 결과 공유 시간이 절약되고, 여러 클라이언트가 같은 세션에 동시 접속할 수 있다는 점은 여기서 더 나아가 실시간 협업까지 가능하게 합니다. 지도교수가 학생의 학습 진행 상황을 실시간으로 지켜보거나, 여러 연구자가 하나의 실험 세션에서 중간 결과를 보며 다음 단계를 논의하는 협업도 가능해질 것으로 기대합니다.
기타	□ 개발 과정 프로젝트 목표 설정 후 아키텍처를 설계해 SPEC을 만들었고, 핵심 기능인 커널 상태·출력 저장/복구 기능과 .py 파일 노트북 렌더링 VS Code 확장을 가장 먼저 구현했습니다. 이후 단위·통합 테스트와 원격 GPU 서버 E2E 테스트로 엣지케이스를 찾아 버그를 수정했고, 문서화를 마친 뒤 PyPI와 VS Code Marketplace에 배포했습니다. 배포 이후에도 지속적으로 버그 수정과 기능 추가를 이어가고 있으며, 현재는 다른 사용자들의 피드백을 받기 위해 베타테스터를 모집 중입니다. □ 프로젝트의 혁신성 및 차별성 JupyterLab의 출력·커널 관리 문제를 해결하기 위해 jupyter-server-documents는 상태 관리를 서버로 위임했고, jupytext는 .ipynb을 .py 파일이나 Markdown 형식과 동기화하는 방식으로 인기를 얻었습니다. 하지만 저는 Jupyter에 기능을 더하는 방식이 아니라, 에이전트 기반 연구에 맞지 않는 Jupyter 생태계 자체를 대체하는 방향을 택했습니다. 다만 단기간에 완전히 새로운 도구를 만들기는 무리라 판단해, 지금의 Tithon은 Jupyter 표준 프로토콜과 ipykernel 위에서 서버만 우선 대체하는 과도기적 형태이며, 장기적으로는 AI 네이티브 커널과 클라이언트 환경을 개발하는 것이 목표입니다. □ 한계점 및 향후 발전 로드맵 ipykernel은 사용자 코드가 커널과 같은 프로세스에서 실행되는 구조라, 멀티 GPU 학습이 CUDA spawn 제약으로 노트북 셀에서 실행

	되지 않고, 크래시나 교착 시 그동안 쌓은 상태를 통째로 잃는 등 단일 프로세스 실행 모델에서 비롯된 구조적 한계가 있습니다. 장기적으로는 GPU 작업을 프로세스 단위로 격리·감독하고, AI 에이전트가 실험을 계획·병렬 실행·재시도할 수 있도록 작업 스케줄링, 구조화된 상태 조회, 명시적 체크포인트를 제공하는 AI 네이티브 런타임을 개발하는 것이 목표이며, 새로운 커널은 이 런타임에서 실행 격리를 담당하는 교체 가능한 백엔드가 될 것입니다. 클라이언트는 현재 VS Code만 지원하며, 추후 API를 정비해 PyCharm 등 다른 클라이언트에도 쉽게 붙일 수 있도록 할 계획입니다. □ 소감 및 후기 Tithon은 인공지능 전공자로서 휴학 후 군 복무 중 개발한 프로젝트입니다. 개인정비 시간에 code-server를 설치해 사지방에서 개발을 이어가던 중, 입대 전 하던 연구도 여기서 할 수 있지 않을까 하는 생각이 들었습니다. 하지만 연결이 끊기면 한번에 상태가 날아가는 사지방 환경에서는 Jupyter Notebook/Lab을 쓸 수 없었고, 며칠간 돌린 학습 결과를 통째로 잃을 위험도 있었습니다. 그래서 어떤 상황에서도 커널과 출력을 잃지 않는 도구가 필요하다는 생각으로 Tithon을 만들게 되었습니다. 처음엔 개인적으로 쓰려던 도구였지만, 많은 연구자들이 같은 불편을 겪는다는 걸 깨닫고 오픈소스로 공개하기로 했습니다. 많은 사람이 써야 개발 목적도 비로소 이뤄지는 것이기에, 설치와 사용이 쉬운 친절한 프로젝트를 만드는 데 많은 노력을 기울였습니다.

붙임1 SBOM(소프트웨어 자재명세서)

번호	라이브러리명	버전	라이선스	공식 저장소 URL(GitHub 등)	사용 목적 및 주요 기능
1	jupyter-client	8.9.1	BSD-3-Clause	https://github.com/jupyter/jupyter_client	Jupyter 커널 연결·제어 API, 커널 채널 및 connection file 처리
2	ipykernel	7.3.0	BSD-3-Clause	https://github.com/ipython/ipykernel	Python 코드 실행을 담당하는 Jupyter 커널
3	websockets	16.0	BSD-3-Clause	https://github.com/python-websockets/websockets	데몬의 WebSocket 서버·클라이언트 및 Unix-domain transport
4	@jupyter-widgets/base	6.0.11	BSD-3-Clause	https://github.com/jupyter-widgets/ipywidgets	JavaScript 위젯 기본 모델·뷰·관리자 기능
5	@jupyter-widgets/controls	5.0.12	BSD-3-Clause	https://github.com/jupyter-widgets/ipywidgets	Slider, Button 등 표준 Jupyter 위젯 컨트롤
6	@jupyter-widgets/ html-manager	1.0.14	BSD-3-Clause	https://github.com/jupyter-widgets/ipywidgets	Notebook webview에서 위젯 모델 등록·렌더링·comm 처리
7	ws	8.21.0	BSD-3-Clause	https://github.com/websockets/ws	VS Code 확장과 Tithon 데몬 간 WebSocket 통신

※ 필요 시, 행을 추가하여 기재해 주시기 바랍니다.

붙임2 AI 모델 활용 및 라이선스 기술 명세서

[작성 안내]
- 본 서식은 프로젝트에 탑재·적용된 AI 모델의 오픈소스 요건 및 라이선스 준수 여부를 검증하기 위한 기술 명세서입니다.
- 참가팀은 본인의 AI 개발 유형(유형1, 2, 3)을 먼저 선택한 후 해당하는 항목의 명세를 정확히 작성하여 제출해 주시기 바랍니다.

1. AI 모델 활용 유형 (해당하는 항목에 ✓ 표시)

☐ 유형 1: 외부 모델 그대로 활용 (추가 학습 없이 기존 공개 모델을 프로젝트에 연동·구동한 경우)

☐ 유형 2: 외부 모델 파인튜닝 (기존 공개 모델을 가져와 준비한 데이터셋으로 추가 미세조정한 경우)

☐ 유형 3: 자체 개발 모델 (기반 모델 없이 참가팀이 처음부터 가중치를 직접 전체 학습시킨 경우)

※ 개발 과정에서 코딩 및 디버깅 보조용으로 상용 AI(GPT, Claude 등)를 단순 활용한 경우는 체크하지 않음(4번 항목에 기재)

2. 기반(베이스) 모델 정보 (유형 1, 유형 2 작성 필수 / 유형 3은 '해당 없음'으로 기재)

기반 모델명 및 개발사	예) Gemma-2-9b (Google) / LLaMA-3-8B (Meta) 등 정확한 명칭 기재	**기반 모델 라이선스**	예) Gemma Terms of Use / LLaMA 3 Communitiy License / Apache 2.0 등

3. 데이터셋 정보 및 가중치 배포 명세 (유형 2, 유형 3 작성 필수)

학습 데이터셋 정보 (출처 및 규모)	[원본 데이터셋 파일 제출 불필요, 명세만 기술] 예) AI Hub 한국어 금융 언어 데이터셋 / 총 54,000건(약 3,200만 토큰) / JSONL 포맷
데이터 정제/ 가공 방법 요약	[개인정보 비식별화(마스킹) 조치 내용, 오픈소스 출품을 위한 프롬프트 포맷 변환 및 필터링 과정을 간략히 기술]

새로 생성된 가중치 공개 저장소 URL	[Hugging Face 모델 리포지토리 주소 등 승인 절차 없이 누구나 접근 가능한 공개 주소 기재]		
가중치 파일 정보 및 배포방식	예) 파일명 : adapter_model.safetensors / LoRA 어댑터 형태 배포 / 용량 : 120MB		
4. 소스코드 라이선스 및 개발 환경 정보 (모든 유형 필수 작성)			
직접 작성한 코드의 오픈소스 라이선스	MIT License	**학습/추론 소스코드 공개 저장소 URL**	https://github.com/rnoro/tithon
상용 AI 보조도구 활용 여부 및 범위	프로젝트 설계, 코드 작성 및 코드 리뷰 보조 용도로 Codex, Claude Code 활용. 전체 개발의 약 50% 비중 차지		