CRESSEM
CREATIVE ASSEMBLY

LLM 기반 PDF 생성 메커니즘의 기술적 한계 및 구현 방안 분석 보고서

문서번호	CRSM-AI-2026-AUTO
작성일	2026-07-21
작성	Cressem AI 시스템 (자동 생성)
보안등급	사내 비밀 (Confidential)
버전	v1.0

목 차

LLM 기반 PDF 생성 메커니즘의 기술적 한계 및 구현 방안 분석 보고서

본 보고서는 LLM 인터페이스에서 사용자의 PDF 생성 요청과 실제 시스템 동작 간의 기술적 괴리를 분석한다. LLM의 토큰 생성 원리와 파일 시스템 접근 권한의 차이를 규명하고, 이를 해결하기 위한 에이전트 기반 아키텍처 및 외부 도구 결합 방안을 제시한다.

개요/배경

최근 생성형 AI(Generative AI) 기술의 비약적인 발전으로 인해 대규모 언어 모델(Large Language Model, LLM)을 활용한 업무 자동화가 산업 전반에서 가속화되고 있습니다. 특히 사용자는 LLM과의 대화 과정에서 단순한 정보 검색이나 텍스트 요약을 넘어, 분석된 결과물을 즉시 실무에 활용할 수 있는 형태인 **PDF(Portable Document Format) 파일 생성**을 직접적으로 요청하는 경향이 매우 강하게 나타나고 있습니다. 이러한 사용자 요구는 데이터 분석부터 문서 편집, 최종 저장까지의 과정을 하나의 인터페이스에서 해결하려는 생산성 극대화의 니즈를 반영하고 있습니다.

그러나 실제 시스템 운영 환경에서는 사용자의 이러한 요청과 LLM의 근본적인 동작 방식 사이에서 심각한 **기능적 간극(Functional Gap)**이 발생하고 있습니다. 사용자가 "이 내용을 PDF 파일로 만들어서 출력 가능하게 해줘"라고 명령할 때, 시스템은 "파일을 생성할 수 없습니다"라거나 "다운로드 링크를 제공할 수 없습니다"라는 식의 응답을 내놓으며 사용자 경험(User Experience, UX)의 단절을 야기합니다. 본 보고서는 바로 이 지점, 즉 사용자의 높은 기대치와 LLM 시스템의 기술적 구현 능력 사이의 괴리를 정의하고 그 원인을 심층적으로 분석하고자 합니다.

1. 사용자 경험(UX) 측면에서의 기대치와 실재(Reality)

사용자는 LLM을 단순한 '텍스트 생성기'가 아닌, 복잡한 워크플로우를 완결할 수 있는 **'범용 AI 에이전트(General AI Agent)'**로 인식하는 경향이 있습니다. 이러한 인식은 다음과 같은 세 가지 차원의 기대치와 실제 시스템 응답 간의 간극을 발생시킵니다.

- **기능적 기대치와 실행 능력의 불일치 (Functional Mismatch):** 사용자는 자연어 명령을 입력했을 때, 시스템이 내부적으로 문서 구조(Layout)를 설계하고, 이를 바이너리(Binary) 데이터로 변환하여 파일 시스템에 저장한 뒤, 최종적으로 다운로드 가능한 URL을 제공할 것이라고 기대합니다. 하지만 표준적인 LLM 인터페이스는 기본적으로 텍₩₩토큰(Token)을 순차적으로 생성하여 화면에 출력하는 '텍스트 생성' 기능에 최적화되어 있으며, 파일 시스템에 직접 개입하여 물리적인 파일을 생성하는 권한을 갖지 않는 경우가 대부분입니다.
- **상호작용 방식의 인지 오류 (Cognitive Gap):** 사용자는 LLM이 마치 인간 비서처럼 '도구(Tool)'를 자유자재로 다룰 수 있다고 가정합니다. 따라서 "파일을 만들어 달라"는 요청을 수행하기 위해 LLM이 내부적으로 Python 스크립트를 작성하거나, PDF 라이브러리를 호출하는 등의 '행동(Action)'을 수행할 것이라 믿습니다. 그러나 LLM의 본질은 확률론적 언어 모델링에 기반한 '예측(Prediction)'이지, 독립적인 '실행(Execution)'이 아니라는 점이 사용자 경험의 혼란을 가중시킵니다.
- **결과물 형태의 가시성 문제 (Output Formality):** 사용자가 원하는 결과물은 규격화된 양식(Standardized Format)을 갖춘 PDF입니다. 반면 LLM이 제공하는 직접적인 결과물은 마크다운(Markdown)이나 일반 텍스트(Plain Text) 형태인 경우가 많습니다. 이 과정에서 사용자는 결과물을 얻기 위해 별도의 '복사 및 붙여넣기(Copy & Paste)'라는 추가적인 수동 작업을 수행해야 하며, 이는 AI를 통한 자동화의 본질적인 가치를 저해하는 요소가 됩니다.

2. 기술적 배경 및 분석의 필요성

이러한 간극을 이해하기 위해서는 LLM이 정보를 처리하고 출력하는 메커니즘을 기술적으로 분리하여 살펴볼 필요가 있습니다. LLM은 입력된 프롬프트에 대해 가장 확률이 높은 다음 토큰을 예측(Next Token Prediction)하여 문장을 구성하지만, 이 과정에서 생성되는 것은 오직 '텍스트 데이터'뿐입니다. PDF와 같은 복잡한 구조를 가진 파일 포맷은 텍스트 외에도 폰트 정보, 레이아웃 메타데이터, 이미지 바이너리 등이 결합된 구조적 데이터셋을 요구합니다.

따라서 본 보고서에서는 단순히 "LLM은 파일을 못 만든다"는 현상적 기술을 넘어, 다음과 같은 핵심 질문에 대한 기술적 해답을 제시하고자 합니다.

1. **아키텍처적 제약:** 왜 LLM은 샌드박스(Sandbox) 환경 내에서 파일 시스템에 접근하여 물리적 파일을 생성하는 것이 제한되는가?

2. **통합의 가능성:** LLM이 외부 도구(External Tools)나 코드 인터프리터(Code Interpreter)와 결합될 때, 어떻게 텍스트 예측을 넘어선 '파일 생성 프로세스'로 진화할 수 있는가?

3. **워크플로우 최적화:** 사용자가 LLM의 출력을 가장 효율적으로 PDF화할 수 있는 최적의 데이터 구조(예: Markdown 기반 추출)는 무엇인가?

결론적으로, 본 보고서는 LLM 기반의 PDF 생성 요청이 발생하는 근본적인 메커니즘을 규명하고, 현재의 기술적 한계를 극복하기 위한 **'도구 활용형(Tool-use) 아키텍처'**와 **'사용자 중심의 워크플로우 설계'** 방안을 제안함으로써, 차세대 AI 에이전트가 나아가야 할 방향성을 제시하는 데 그 목적이 있습니다.

LLM의 기본 동작 원리와 출력 메커니즘

사용자가 LLM(Large Language Model)에게 "PDF 파일을 만들어 달라"고 요청했을 때, 시스템이 이를 수행하지 못하고 텍스트 형태의 답변만을 내놓는 근본적인 이유는 LLM의 본질적인 동작 아키텍처가 **'텍스트 기반의 확률적 생성(Probabilistic Text Generation)'**에 국한되어 있기 때문입니다. 본 섹션에서는 LLM이 입력된 질의를 어떻게 이해하고, 어떤 과정을 거쳐 최종적인 출력물을 생성하는지, 그리고 왜 이 과정이 파일(Binary File) 생성이라는 물리적 행위와 분리되어 있는지를 기술적 관점에서 심층 분석합니다.

1. 텍스트의 수치화: 토큰화(Tokenization)와 임베딩(Embedding)

LLM은 인간이 사용하는 자연어(Natural Language)를 직접 이해할 수 없습니다. 컴퓨터가 처리할 수 있는 수치 데이터로 변환하는 첫 번째 단계가 바로 **토큰화(Tokenization)**입니다.

- **토큰화 프로세스:** 입력된 문장은 의미 있는 최소 단위인 '토큰(Token)'으로 분해됩니다. 토큰은 단어(Word) 단위일 수도 있고, 형태소(Morpheme)나 문자(Character) 단위일 수도 있습니다. 최근의 모델들은 Byte Pair Encoding(BPE)과 같은 알고리즘을 사용하여 빈번하게 등장하는 문자 조합을 하나의 토큰으로 묶음으로써 효율성을 극대화합니다.
- **임베딩 공간으로의 투영:** 분해된 각 토큰은 고차원의 벡터(High-dimensional Vector)로 변환됩니다. 이를 **임베딩(Embedding)**이라고 하며, 이 벡터 공간 내에서 의미가 유사한 토큰들은 서로 가까운 거리에 위치하게 됩니다. 예를를 들어, '반도체'와 '웨이퍼'라는 토큰은 '사과'라는 토큰보다 벡터 공간 상에서 더 높은 유사도를 가집니다.

이 단계에서 LLM은 사용자의 의도(Intent)를 수학적인 좌표값의 집합으로 변환하여 수용하게 됩니다. 하지만 중요한 점은, 이 과정에서 생성되는 모든 데이터는 '의미를 가진 수치 벡터'일 뿐, 그 자체로 파일의 구조나 형식을 갖춘 것이 아니라는 사실입니다.

2. 추론 엔진의 핵심: 차세대 토큰 예측(Next Token Prediction)

LLM의 가장 핵심적인 동작 원리는 **차세대 토큰 예측(Next Token Prediction)**입니다. 이는 트랜스포머(Transformer) 아키텍처를 기반으로 하며, 모델은 주어진 문맥(Context)을 바탕으로 다음에 올 확률이

가장 높은 토큰을 하나씩 찾아내는 과정을 반복합니다.

2.1. 어텐션 메커니즘(Attention Mechanism)과 문맥 이해

LLM이 "PDF 파일을 만들어줘"라는 문장을 처리할 때, **셀프 어텐션(Self-Attention)** 메커니즘은 문장 내 각 토큰 간의 상관관계를 계산합니다. '만들어줘'라는 동사가 'PDF 파일'이라는 목적어와 어떻게 연결되는지, 그리고 이 요청이 '파일 생성'이라는 물리적 동작을 요구하는 것인지 아니면 'PDF에 들어갈 내용'을 요구하는 것인지를 문맥적으로 파악합니다.

2.2. 확률 분포 기반의 샘플링(Sampling)

모델은 내부적으로 다음에 올 수 있는 모든 토큰 후보군에 대해 확률 분포(Probability Distribution)를 생성합니다.

1. **Softmax 함수 적용:** 모델의 마지막 레이어에서 각 토큰의 점수(Logits)를 0과 1 사이의 확률값으로 변환합니다.

2. **샘플링 전략:** 단순히 확률이 가장 높은 토큰만 선택하면 답변이 단조로워지므로, **Temperature(온도)** 파라미터 등을 조절하여 확률적 다양성을 부여합니다.

- Temperature → 0: 가장 확률이 높은 토큰만 선택 (결정론적, 반복적 답변)
- Temperature → 1: 확률 분포에 따라 다양한 토큰 선택 (창의적, 가변적 답변)

이 과정을 통해 모델은 한 글자씩, 혹은 한 토큰씩 문장을 이어 나갑니다. 즉, LLM의 출력은 **"다음에 올 확률이 가장 높은 텍스트 조각들을 순차적으로 이어 붙인 결과물"**입니다.

3. 출력의 한계: Text-only Output의 본질적 제약

여기서 사용자가 직면하는 기술적 간극이 발생합니다. LLM의 출력 메커니즘은 철저하게 **텍스트 전용 출력(Text-only Output)** 체계로 설계되어 있습니다.

구분	LLM의 텍스트 생성 (Inference)	파일 생성 (File Generation)
데이터 형태	유니코드(Unicode) 기반의 문자열	바이너리(Binary) 데이터 스트림
출력 단위	토큰(Token)	바이트(Byte) 및 파일 구조(Header, Body, Footer)
동작 방식	확률적 순차 생성	특정 규격(Spec)에 따른 데이터 쓰기(Write)
결과물 특성	화면에 표시되는 가독성 있는 글자	운영체제(OS)가 인식하는 파일 객체

표 1. 출력의 한계: Text-only Output의 본질적 제약

3.1. 확률적 생성 vs 규격적 생성

PDF 파일은 단순한 텍스트의 집합이 아닙니다. PDF는 파일 헤더(Header), 객체(Object), 크로스 레퍼런스 테이블(Cross-reference Table), 트레일러(Trailer) 등 엄격한 구조를 가진 **바이너리 규격**을 준수해야 합니다.

LLM이 "PDF 파일을 생성합니다"라고 텍스트를 출력하는 것은, "다음에 'PDF'라는 단어가 올 확률이 높다"는 것을 계산하여 문자를 화면에 뿌려주는 행위일 뿐입니다. 모델은 텍스트를 생성하는 과정에서 PDF의 내부 바이너리 구조(예: `%PDF-1.7` 헤더나 객체 간의 오프셋 주소값)를 수학적으로 '예측'할 수는 있으나, 이를 실제 파일 시스템에 기록하여 물리적인 `.pdf` 파일을 완성하는 **I/O(Input/Output) 프로세스**를 수행할 수 있는 엔진을 갖추고 있지

않습니다.

3.2. 추론(Inference)과 실행(Execution)의 분리

LLM의 추론 단계는 모델의 파라미터(Weights)를 사용하여 입력에 대한 최적의 토큰 시퀀스를 찾는 과정입니다. 반면, 파일을 생성하기 위해서는 메모리 할당, 디스크 쓰기 권한 획득, 파일 시스템 인터페이스 호출 등 **컴퓨팅 자원의 직접적인 제어**가 필요합니다.

결론적으로, LLM의 기본 동작 원리는 '언어적 패턴의 모사'에 집중되어 있으며, 이는 '데이터 구조의 물리적 구현'과는 완전히 다른 차원의 문제입니다. 따라서 LLM은 사용자에게 PDF에 들어갈 **'내용(Content)'**은 완벽하게 작성해 줄 수 있지만, 그 내용을 담은 **'그릇(Container)'**인 파일 자체를 만들어내는 기능은 기본 아키텍처 내에 포함되어 있지 않습니다. 이러한 한계를 극복하기 위해서는 LLM이 스스로 코드를 작성하고 이를 외부 환경에서 실행하게 하는 'Tool-use' 또는 'Code Interpreter'와 같은 추가적인 계층(Layer)이 반드시 필요합니다.

PDF 생성 불가 원인 분석: 시스템 아키텍처 관점

사용자가 LLM 인터페이스를 통해 "PDF 파일을 만들어 달라"는 요청을 수행했을 때, 모델이 텍스트 기반의 응답은 생성하면서도 실제 파일(Binary File)을 물리적으로 생성하여 전달하지 못하는 현상은 단순한 기능의 부재가 아닌, 현대 LLM 서비스가 채택하고 있는 **보안 중심의 시스템 아키텍처(Security-centric Architecture)** 설계에 기인합니다. 본 섹션에서는 LLM의 추론 엔진과 이를 서비스하는 인프라 환경 사이의 구조적 제약 조건을 세 가지 핵심 관점인 샌드박스(Sandbox) 환경, 파일 시스템 접근 제한(File System Access Restriction), 그리고 권한 관리(Permission Management)를 중심으로 심층 분석합니다.

1. 격리된 실행 환경: 샌드박스(Sandbox) 구조의 제약

대규모 언어 모델을 서비스하는 클라우드 인프라는 모델의 추론 과정에서 발생할 수 있는 잠재적 위협으로부터 호스트 시스템과 사용자 데이터를 보호하기 위해 강력한 **샌드박스(Sandbox)** 환경을 운용합니다. 샌드박스는 외부의 악성 코드나 비정상적인 명령어가 시스템 전체로 확산되는 것을 방지하기 위해 구축된 격리된 가상 실행 공간을 의미합니다.

- **추론 엔진의 격리성:** LLM의 핵심인 트랜스포머(Transformer) 아키텍처 기반 추론 엔진은 입력된 프롬프트에 대해 최적의 토큰(Token) 확률 분포를 계산하여 텍스트를 출력하는 데 집중합니다. 이 과정은 메모리 상의 연산(In-memory computation)으로 이루어지며, 추론 엔진 자체가 운영체제(OS)의 커널이나 로컬 저장 장치에 직접 관여할 수 있는 권한을 갖지 않도록 설계되어 있습니다.
- **보안적 목적과 기능적 트레이드오프(Trade-off):** 만약 LLM이 생성하는 텍스트 내에 악성 스크립트가 포함되어 있고, 모델이 이를 실행하거나 파일로 저장할 수 있는 권한을 가진다면, 이는 즉각적인 시스템 침해(System Compromise)로 이어질 수 있습니다. 따라서 서비스 제공업체는 모델의 '자율성'을 의도적으로 제한하여, 모델이 생성한 결과물이 텍스트 스트림(Text Stream) 형태로만 사용자에게 전달되도록 아키텍처를 고착화합니다. 즉, 모델은 '무엇을 쓸 것인가(What to write)'는 결정할 수 있지만, '어떻게 물리적 형태로 저장할 것인가(How to store physically)'에 대한 실행권은 박탈당한 상태입니다.

2. 파일 시스템 접근 제한(File System Access Restriction) 및 I/O의 한계

LLM이 PDF와 같은 바이너리 파일을 생성하기 위해서는 파일 시스템(File System)에 쓰기(Write) 작업을 수행하고, 생성된 파일의 경로(Path)를 관리하며, 이를 네트워크를 통해 사용자에게 전송하는 일련의 I/O(Input/Output) 프로세스가 필요합니다. 그러나 표준적인 LLM 인터페이스 아키텍처는 다음과 같은 이유로 파일 시스템 접근을 엄격히 차단합니다.

- **비휘발성 저장소와의 단절:** LLM의 응답은 기본적으로 'Stateless(상태 비저장)' 특성을 지향합니다. 모델은 매 요청마다 컨텍스트 윈도우(Context Window) 내의 정보만을 참조하며, 모델의 출력 결과가 서버의

디스크(Disk)에 영구적으로 기록되는 것은 별도의 데이터베이스(DB)나 스토리지 서비스가 개입해야 하는 영역입니다. 일반적인 채팅 인터페이스는 모델의 출력을 데이터베이스에 '텍스트 로그'로 저장할 뿐, 모델이 임의의 파일 형식을 생성하여 디스크에 쓰는 행위를 허용하지 않습니다.

- **바이너리 데이터 처리의 복잡성:** PDF는 단순한 텍스트 파일이 아닌, 구조화된 객체(Object), 폰트(Font), 이미지(Image), 압축된 스트림(Compressed Stream) 등이 결합된 복잡한 바이너리 형식입니다. LLM이 텍스트 토큰을 생성하는 방식으로는 이러한 바이너리 구조를 정밀하게 제어하기 어렵습니다. 모델이 PDF의 내부 구조를 16진수(Hexadecimal) 데이터 수준에서 생성하려고 시도하더라도, 이를 파일로 변환하여 저장할 수 있는 '파일 시스템 인터페이스'가 차단되어 있다면 사용자는 결코 완성된 파일을 받아볼 수 없습니다.

3. 권한 관리(Permission Management) 및 보안 거버넌스

시스템 아키텍처 관점에서 가장 결정적인 원인은 **권한(Permission)**의 부재입니다. 현대의 클라우드 네이티브(Cloud-native) 환경은 '최소 권한의 원칙(Principle of Least Privilege)'을 준수합니다.

구분	LLM 추론 엔진의 권한 범위	파일 생성/전송에 필요한 권한	비고
컴퓨팅 자원	GPU/NPU 연산 및 메모리 할당	CPU 기반 파일 시스템 I/O 제어	연산과 저장의 분리
네트워크	API 호출 및 데이터 수신	외부 다운로드 링크 생성 및 전송	보안 프로토콜 충돌
저장소	컨텍스트 윈도우 내 임시 메모리	비휘발성 스토리지(S3, NAS 등) 쓰기	데이터 영속성 문제
운영체제	제한된 런타임 환경(Runtime)	파일 시스템 핸들(File Handle) 획득	OS 레벨의 차단

표 2. 권한 관리(Permission Management) 및 보안 거버넌

- **사용자 세션과 파일 시스템의 분리:** 사용자가 받는 '다운로드 링크'는 단순한 텍스트가 아니라, 서버의 특정 경로에 존재하는 실제 파일을 가리키는 포인터(Pointer)여야 합니다. 하지만 LLM은 권한 문제로 인해 서버 내의 특정 경로에 파일을 생성할 수 없으므로, 존재하지 않는 경로를 가리키는 가짜(Fake) URL을 생성하게 되는 오류를 범하게 됩니다.
- **데이터 유출 방지(DLP, Data Loss Prevention):** 만약 LLM이 자유롭게 파일을 생성하고 외부로 전송할 수 있다면, 이는 기업의 기밀 데이터나 개인정보가 모델의 생성 과정을 통해 외부로 유출되는 통로로 악용될 수 있습니다. 따라서 기업용 솔루션이나 고도화된 AI 서비스일수록 모델의 파일 생성 및 외부 전송 권한을 엄격히 통제하는 거버넌스 체계를 구축하고 있습니다.

4. 소결: 아키텍처적 격차의 해결 방향

결론적으로, LLM이 PDF를 생성하지 못하는 것은 모델의 지능(Intelligence) 문제가 아니라, **'지능(Reasoning)'과 '실행(Execution)'이 분리된 아키텍처** 때문입니다. 모델은 PDF의 내용을 구성하는 '설계도'를 그릴 수는 있지만, 그 설계도를 바탕으로 실제 건물을 짓는 '공사 장비(File System & I/O)'에 접근할 권한이 없습니다.

이러한 격차를 극복하기 위해서는 향후 AI 에이전트 아키텍처가 단순한 텍스트 생성을 넘어, **도구 사용(Tool-use)** 또는 **함수 호출(Function Calling)** 메커니즘을 통해 격리된 코드 실행 환경(예: Python Interpreter)을 호출하고, 해당 환경 내에서 생성된 파일을 안전하게 사용자에게 전달할 수 있는 '샌드박스 간 통신(Inter-sandbox Communication)' 프로토콜을 확보해야 합니다.

LLM과 외부 도구의 통합 모델 (Tool-use/Function Calling)

앞선 섹션에서 살펴본 바와 같이, 순수한 언어 모델(Pure LLM)은 텍스트 토큰을 예측하여 출력하는 확률적 생성 엔진에 불과하며, 자체적인 파일 시스템 접근 권한이나 바이너리 생성 능력을 결여하고 있습니다. 사용자가 "PDF 파일을 만들어 달라"는 명령을 내렸을 때, 모델이 단순히 "파일을 생성할 수 없습니다"라고 답변하는 것은 모델의 지능 문제가 아닌, 아키텍처상의 **기능적 격리(Functional Isolation)** 때문입니다. 이러한 한계를 극복하고 LLM을 실질적인 '행동하는 에이전트(Acting Agent)'로 진화시키기 위한 핵심 기술적 돌파구가 바로 **도구 활용(Tool-use)** 및 **함수 호출(Function Calling)** 메커니즘입니다.

1. Function Calling: 의도 파악에서 실행 명령으로의 전환

전통적인 LLM은 사용자의 질의를 해석하여 그에 적합한 텍스트를 생성하는 데 그쳤으나, 현대적인 에이전트 구조에서의 LLM은 사용자의 자연어 의도(Intent)를 분석하여 사전에 정의된 **외부 함수(External Function)**의 규격에 맞는 인자(Arguments)를 추출하는 '오케스트레이터(Orchestrator)' 역할을 수행합니다.

이 과정에서 핵심은 모델이 단순히 텍스트를 나열하는 것이 아니라, 특정 API나 함수를 호출하기 위한 정형화된 데이터 구조(주로 JSON 형식)를 생성하도록 훈련되었다는 점입니다. 예를 들어, 사용자가 "보고서 내용을 바탕으로 'report.pdf'라는 이름의 PDF를 생성해줘"라고 요청하면, 모델은 다음과 같은 논리적 단계를 거칩니다.

1. **의도 분석(Intent Recognition):** 사용자의 요청이 단순 정보 검색인지, 아니면 파일 생성이라는 '행동'을 요구하는 것인지 판별합니다.

2. **도구 선택(Tool Selection):** 현재 가용한 도구 목록(예: `generatepdf`, `searchweb`, `calculatemath`) 중 `generatepdf` 함수가 적합함을 인지합니다.

3. **인자 추출(Argument Extraction):** 함수 실행에 필요한 필수 파라미터인 `filename="report.pdf"`와 `content=[추출된 텍스트 데이터]`를 구조화된 형태로 생성합니다.

이때 LLM은 직접 파일을 만드는 것이 아니라, **"이 함수를, 이러한 값들을 넣어 실행해달라"**는 일종의 '실행 계획서'를 작성하는 것입니다.

2. Code Interpreter 및 Python Interpreter를 통한 실행 환경의 확장

Function Calling이 '무엇을 할 것인가'에 대한 계획을 세우는 단계라면, **코드 인터프리터(Code Interpreter)**는 그 계획을 실제로 구현하는 '실행 엔진'입니다. 특히 PDF 생성과 같이 복잡한 라이브러리(예: ReportLab, PyFPDF, Pandoc)와 정교한 레이아웃 계산이 필요한 작업의 경우, 단순한 API 호출보다 프로그래밍 코드를 직접 생성하고 실행하는 방식이 훨씬 강력한 유연성을 제공합니다.

이 아키텍처에서는 LLM이 사용자의 요구사항을 만족시키기 위한 Python 스크립트를 작성합니다. 작성된 코드는 모델의 텍스트 출력창이 아닌, 격리된 **샌드박스(Sandbox) 환경** 내의 인터프리터로 전달됩니다.

- **동적 코드 생성(Dynamic Code Generation):** 사용자가 요청한 표(Table), 이미지(Image), 수식(Formula) 등을 포함하기 위해 모델은 적절한 라이브러리를 임포트하고, 객체 지향적인 방식으로 PDF 문서를 구성하는 코드를 작성합니다.
- **런타임 실행 및 피드백 루프(Runtime Execution & Feedback Loop):** 인터프리터가 코드를 실행하면, 그 결과로 파일이 생성되거나 혹은 코드 오류(Runtime Error)가 발생할 수 있습니다. 만약 오류가 발생하면, 시스템은 에러 로그를 다시 LLM에게 전달합니다. LLM은 이 로그를 분석하여 코드를 수정(Self-Correction)하고, 성공할 때까지 이 과정을 반복합니다.

이러한 방식은 정해진 규격의 API만 사용하는 것보다 훨씬 높은 자유도를 가지며, 사용자가 요구하는 미세한 서식 변경(예: "폰트 크기를 12pt로 키워줘", "여백을 더 넓게 해줘")에 실시간으로 대응할 수 있게 합니다.

3. 외부 API 및 시스템 통합 아키텍처

고도화된 AI 에이전트는 코드 인터프리터를 넘어, 기업용 솔루션이나 클라우드 서비스와 연동되는 **외부 API(External API)**를 도구로 활용합니다. PDF 생성 시 단순 텍스트를 넘어 기업 내부의 데이터베이스(DB)에서 최신 수치를 가져오거나, 전문적인 디자인 도구의 API를 호출하여 고품질의 문서를 완성하는 방식입니다.

이 과정에서 보안과 안정성은 가장 중요한 기술적 과제입니다. LLM이 생성한 코드가 시스템의 핵심 자원에 접근하거나 악의적인 동작을 수행하지 않도록, 모든 실행은 엄격하게 통제된 컨테이너(Container) 환경에서 이루어져야 합니다.

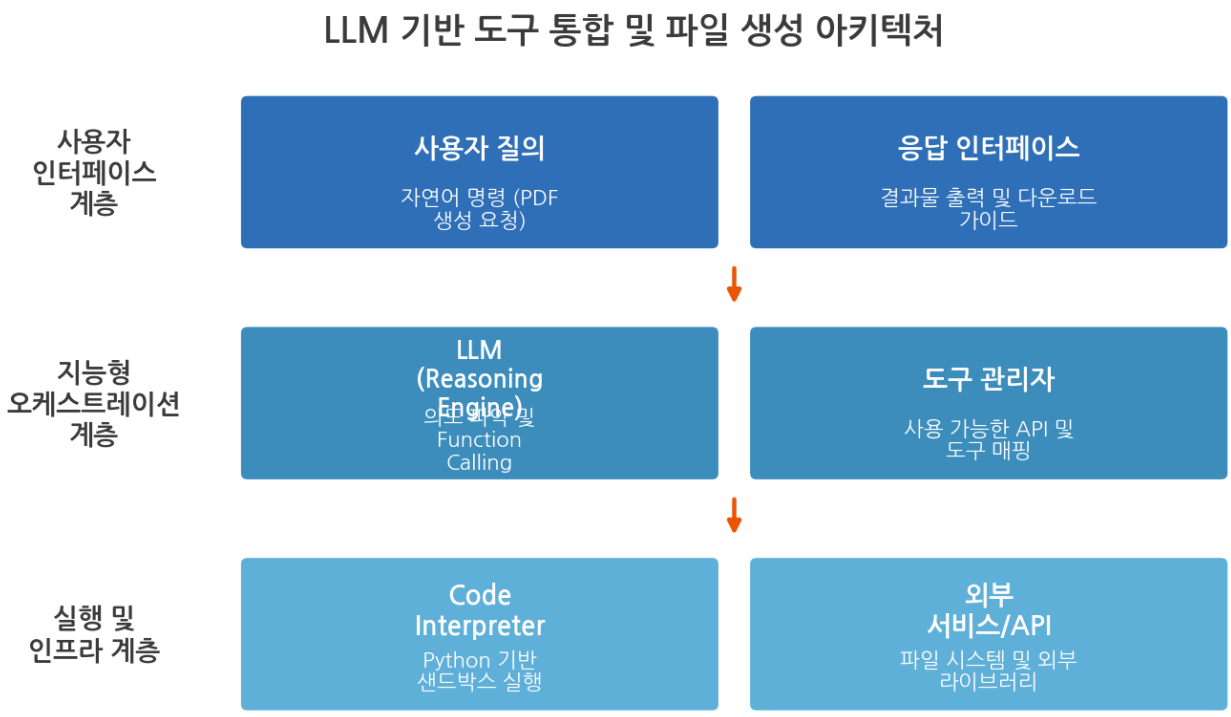

그림 1. LLM 기반 도구 통합 및 파일 생성 아키텍처

4. 기술적 도전 과제: 신뢰성과 정밀도

도구 통합 모델이 완벽한 해결책처럼 보이지만, 실제 산업 현장(예: 반도체 검사 보고서 생성 등)에 적용하기 위해서는 다음과 같은 기술적 난제들이 존재합니다.

첫째, **할루시네이션(Hallucination)의 전이** 문제입니다. LLM이 존재하지 않는 함수 이름을 호출하거나, 잘못된 파라미터 형식을 생성할 경우 전체 워크플로우가 중단됩니다. 이를 방지하기 위해 함수 스키마(Schema)를 엄격하게 정의하고, 생성된 인자가 규격에 맞는지 검증하는 **가드레일(Guardrails)** 기술이 필수적입니다.

둘째, **상태 유지(State Management)**의 어려움입니다. PDF 생성 과정은 대개 여러 단계(데이터 추출 → 레이아웃 설계 → 코드 작성 → 실행 → 검증)를 거칩니다. 각 단계의 중간 결과물과 컨텍스트를 모델이 일관되게 유지하면서 최종 결과물까지 도달하게 만드는 긴 문맥(Long-context) 관리 능력이 요구됩니다.

셋째, **비용 및 지연 시간(Latency)** 문제입니다. 복잡한 추론과 코드 실행, 그리고 반복적인 피드백 루프는 사용자에게 즉각적인 응답을 제공하기 어렵게 만듭니다. 따라서 효율적인 토큰 사용과 최적화된 실행 환경 구축은 상용 에이전트 구현의 핵심적인 엔지니어링 요소가 됩니다.

PDF 생성 자동화 워크플로우 설계

LLM이 단순히 텍스트를 출력하는 단계를 넘어, 사용자가 즉시 활용 가능한 형태의 결과물(Artifact)을 제공하기 위해서는 **'텍스트 생성'**과 **'파일 직렬화(Serialization)'**라는 두 개의 서로 다른 도메인을 연결하는 정교한 자동화 워크플로우(Automation Workflow)가 필수적입니다. 본 섹션에서는 LLM의 추론 결과물을 기반으로 구조화된 마크다운(Markdown) 데이터를 추출하고, 이를 물리적인 PDF 바이너리 파일로 변환하여 사용자에게 전달하기까지의 전 과정을 단계별로 설계합니다.

1. 워크플로우의 핵심 아키텍처: Markdown-to-PDF 파이프라인

단순한 텍스트 나열은 문서로서의 가치가 낮습니다. 따라서 자동화 워크플로우의 출발점은 LLM이 생성한 비정형 텍스트를 **구조화된 마크다운(Structured Markdown)** 형식으로 강제하는 것입니다. 마크다운은 헤더(#), 목록(-), 표(|), 강조(**) 등 문서의 계층 구조를 명확히 정의할 수 있어, 이후 PDF 변환 엔진이 문서의 논리적 흐름을 파악하는 데 결정적인 역할을 합니다.

이 워크플로우는 크게 **[추론 및 구조화] → [데이터 정제] → [렌더링 엔진 구동] → [바이너리 생성]**의 4단계 프로세스를 따릅니다. 특히, LLM이 생성한 내용 중에 포함될 수 있는 수식(LaTeX), 이미지 경로, 표(Table) 데이터가 깨지지 않고 PDF 상에 정확히 렌더링되도록 하는 것이 기술적 구현의 핵심입니다.

2. 단계별 프로세스 상세 분석

Step 1: 구조화된 콘텐츠 생성 (Structured Content Generation)

LLM은 사용자의 요청에 따라 내용을 생성할 때, 단순 텍스트가 아닌 마크다운 문법을 준수하도록 유도됩니다. 이때 'Constrained Generation(제약 생성)' 기술을 적용하여, 문서의 논리적 위계(Hierarchy)가 유지되도록 합니다. 예를 들어, 보고서의 제목은 `#`, 소제목은 `##`로 시작하도록 엄격한 포맷팅 가이드를 프롬프트에 포함합니다.

Step 2: 중간 데이터 추출 및 정제 (Extraction & Sanitization)

생성된 텍스트 블록에서 순수 마크다운 소스만을 추출합니다. 이 과정에서 LLM의 서술적 답변(예: "네, 요청하신 보고서를 작성했습니다.")을 제거하고, 오직 문서 본문에 해당하는 코드 블록이나 마크다운 텍스트만을 분리해내는 파싱(Parsing) 작업이 수행됩니다.

Step 3: 렌더링 엔진을 통한 시각화 (Rendering via Engine)

추출된 마크다운은 PDF 변환 엔진(예: Pandoc, WeasyPrint, 또는 Python의 `FPDF`/`ReportLab` 라이브러리)으로 전달됩니다. 이 단계에서 엔진은 마크다운의 스타일 가이드를 해석하여 폰트 크기, 줄 간격, 여백(Margin), 페이지 번호 등을 결정합니다. 특히 전문적인 보고서의 경우, 표(Table)의 너비를 자동으로 조절하거나 수식을 고해상도 벡터 그래픽으로 변환하는 과정이 포함됩니다.

Step 4: 바이너리 파일 직렬화 및 저장 (Binary Serialization & Storage)

렌더링된 결과물은 메모리 상에서 PDF 바이너리 데이터로 직렬화됩니다. 생성된 데이터는 시스템의 임시 저장소(Temporary Storage)나 지정된 파일 시스템 경로에 `.pdf` 확장자를 가진 파일로 기록됩니다. 최종적으로 이 파일의 경로 또는 접근 가능한 URL이 사용자 인터페이스(UI)를 통해 사용자에게 노출됩니다.

3. 자동화 워크플로우 개념도

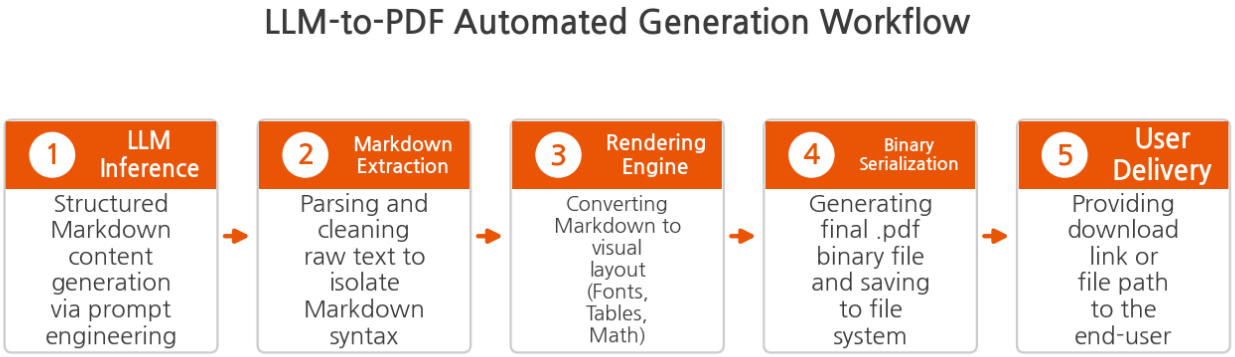

그림 2. LLM-to-PDF Automated Generation Workflow

4. 구현 시 고려해야 할 기술적 변수

성공적인 자동화 워크플로우 구축을 위해서는 다음과 같은 변수들에 대한 정밀한 제어가 필요합니다.

- **수식 및 특수 기호 처리:** 반도체 기술 보고서와 같이 복잡한 수식(μm, Ω, Δ)이 포함되는 경우, 이를 단순 텍스트가 아닌 MathJax 또는 LaTeX 엔진을 통해 렌더링할 수 있는 환경을 구축해야 합니다.
- **표(Table)의 복잡도:** 데이터가 많은 표의 경우, 페이지 경계를 넘어갈 때 자동으로 다음 페이지로 이어지는(Table Breaking) 기능이 구현되어야 문서의 가독성이 유지됩니다.
- **에러 핸들링(Error Handling):** LLM이 마크다운 문법을 실수로 파괴하거나(예: 닫히지 않은 코드 블록), 렌더링 엔진이 지원하지 않는 특수 문자를 생성할 경우, 워크플로우가 중단되지 않도록 예외 처리(Exception Handling) 로직이 반드시 포함되어야 합니다.

사용자 경험(UX) 개선을 위한 대안적 접근법

LLM이 직접적인 파일 바이너리(Binary)를 생성하여 전달하지 못하는 기술적 제약 상황에서, 사용자가 느끼는 서비스의 불완전함을 상쇄하기 위해서는 **사용자 경험(User Experience, UX)** 측면의 전략적 설계가 필수적입니다. 시스템이 파일을 직접 '제공'할 수 없다면, 사용자가 생성된 결과물을 가장 쉽고 빠르게 '자신의 문서로 변환'할 수 있도록 유도하는 **복사/붙여넣기 전략(Copy-Paste Strategy)**과 **마크다운(Markdown) 최적화**가 핵심적인 대안이 됩니다.

1. 마크다운(Markdown) 기반의 구조화된 출력 최적화

LLM이 생성하는 결과물을 단순한 평문(Plain Text)이 아닌, 구조화된 **마크다운(Markdown)** 형식으로 제공하는 것은 UX 개선의 첫걸음입니다. 마크다운은 텍스트 기반의 경량 마크업 언어로, 헤더(#), 목록(*, -), 표(|---|), 볼드(**) 등의 문법을 사용하여 문서의 계층 구조를 명확히 표현합니다.

- **시각적 인지 효율성 증대:** 마크다운 문법을 활용하면 사용자가 텍스트를 복사하여 워드(MS Word), 노션(Notion), 옵시디언(Obsidian) 등의 도구에 붙여넣었을 때, 별도의 서식 재설정 없이도 즉시 완성도 높은 문서 형태를 유지할 수 있습니다.
- **데이터 무결성 유지:** 특히 표(Table) 데이터의 경우, 마크다운 형식을 사용하면 엑셀(Excel)이나 스프레드시트로의 데이터 전이(Data Transfer)가 용이하여, 사용자가 수동으로 데이터를 재배열해야 하는 번거로움을 획기적으로 줄여줍니다.

2. 복사/붙여넣기 가이드(Copy-Paste Strategy) 및 사용자 안내

시스템은 사용자에게 단순히 답변을 출력하는 것을 넘어, 해당 답변을 어떻게 활용해야 하는지에 대한 **명시적 가이드(Explicit Guide)**를 제공해야 합니다. 이는 사용자의 '기능적 기대치'와 '실제 시스템 능력' 사이의 간극을 메우는 심리적 완충 작대 역할을 합니다.

단계	전략적 접근 (Strategy)	기대 효과 (Expected Effect)
출력 단계	답변 하단에 '마크다운 복사' 또는 '텍스트 복사' 버튼 배치	사용자의 조작 단계 최소화 및 편의성 증대
안내 단계	"위 내용을 PDF로 저장하려면 [방법]을 따르세요"라는 안내 문구 삽입	시스템 한계에 대한 인지 및 대안 제시를 통한 불만 감소
변환 단계	마크다운을 PDF로 변환하는 외부 도구(Pandoc, Typora 등) 추천	사용자의 최종 목적(PDF 생성) 달성을 위한 경로 제공

표 3. 복사/붙여넣기 가이드(Copy-Paste Strategy) 및 사용

3. 사용자 맞춤형 워크플로우 제안

사용자의 숙련도에 따라 서로 다른 변환 경로를 제안함으로써 UX의 범용성을 확보할 수 있습니다.

1. **초급 사용자(Non-technical User):** "답변 내용을 복사하여 워드(Word) 프로그램에 붙여넣은 후, [다른 이름으로 저장] -> [PDF]를 선택하세요"와 같은 직관적인 단계별 지침을 제공합니다.

2. **전문 사용자(Power User):** 마크다운 문법을 지원하는 전문 에디터나, VS Code의 마크다운 익스텐션을 활용한 고품질 PDF 렌더링 방식을 안내하여 생산성을 극대화합니다.

결론적으로, LLM의 출력 형식을 **구조화된 마크다운**으로 표준화하고, 이를 활용한 **효율적인 변환 프로세스(Conversion Process)**를 사용자에게 적극적으로 가이드하는 것은, 기술적 한계를 극복하고 서비스 신뢰도를 유지할 수 있는 가장 현실적이고 강력한 UX 개선 방안입니다.

결론 및 시사점

본 보고서는 사용자가 LLM 인터페이스를 통해 요청하는 'PDF 파일 생성' 명령이 왜 표준적인 텍스트 생성 메커니즘만으로는 충족될 수 없는지를 시스템 아키텍처와 출력 프로세스의 관점에서 심층 분석하였습니다. 분석 결과, 현재의 기술적 한계는 단순한 기능의 부재가 아니라, LLM의 근본적인 동작 원리인 **차세대 토큰 예측(Next Token Prediction)** 방식과 파일 시스템이라는 **물리적/논리적 저장 계층(Storage Layer)** 사이의 단절에서 기인함을 확인하였습니다.

1. 기술적 요약 및 한계점의 재정의

LLM은 본질적으로 확률적 텍스트 생성 엔진이며, 이는 사용자의 의도를 파악하고 논리적인 문장을 구성하는 데 탁월한 성능을 보이지만, 스스로 바이너리(Binary) 데이터를 조작하거나 운영체제(OS)의 파일 시스템에 직접 쓰기(Write) 작업을 수행할 수 있는 권한을 갖지 않습니다.

- **텍스트 기반 출력의 한계:** 현재의 LLM 인터페이스는 텍스트 토큰을 시각화하여 전달하는 데 최적화되어 있어, PDF와 같은 복잡한 구조의 파일 포맷을 생성하기 위해서는 별도의 **코드 실행 환경(Code Interpreter)** 또는 **도구 활용(Tool-use)** 메커니즘이 필수적입니다.
- **워크플로우의 단절:** 사용자가 기대하는 '원클릭 파일 생성'이 이루어지기 위해서는 [텍스트 생성 → 코드 변환 → 샌드박스 내 실행 → 파일 시스템 저장 → 다운로드 링크 생성]으로 이어지는 다단계 파이프라인이 유기적으로 결합되어야 합니다.

2. 차세대 AI 에이전트(AI Agent)를 위한 발전 방향

미래의 AI 기술은 단순한 '대화형 인터페이스'를 넘어, 사용자의 목표를 달도하기 위해 도구를 능동적으로 사용하는 **자율형 AI 에이전트(Autonomous AI Agent)**로 진화해야 합니다. 이를 위해 다음과 같은 세 가지 핵심 역량이 요구됩니다.

① 멀티모달 출력(Multimodal Output) 능력의 확장

현재의 LLM이 텍스트(Text)와 이미지(Image)를 이해하는 '멀티모달 입력(Multimodal Input)' 단계에 머물러 있다면, 차세대 모델은 결과물을 텍스트를 넘어 PDF, Excel, PPT, 혹은 실행 가능한 소프트웨어 패키지 형태로 출력할 수 있는 **멀티모달 출력(Multimodal Output)** 능력을 갖추어야 합니다. 이는 AI가 생성한 지식이 단순히 화면에 머무는 것이 아니라, 실무 현장에서 즉시 사용 가능한 '자산(Asset)'으로 변환됨을 의미합니다.

② 도구 활용 및 환경 제어(Tool-use & Environment Control)

AI 에이전트는 외부 API, Python 라이브러리, 데이터베이스, 그리고 파일 시스템과 상호작용할 수 있는 능력을 갖추어야 합니다. 예를 들어, 사용자가 "분석 보고서를 PDF로 만들어줘"라고 요청할 경우, 에이전트는 스스로 `ReportLab`이나 `Pandas`와 같은 라이브러리를 호출하여 데이터를 구조화하고, 이를 바이너리 파일로 렌더링하는 일련의 과정을 자율적으로 수행할 수 있어야 합니다.

③ 미래 로드맵: 통합형 AI 워크스테이션(Integrated AI Workstation)

결국 미래의 AI는 개별적인 채팅창의 형태를 벗어나, 사용자의 운영체제나 업무용 소프트웨어 내에 깊숙이 통합된 **AI 워크스테이션**의 형태로 발전할 것입니다. 이 단계에서는 사용자가 별도의 복사/붙여넣기 과정 없이도 AI가 생성한 결과물을 즉시 로컬 디렉토리에 저장하거나, 클라우드 스토리지로 전송하고, 협업 도구로 공유하는 매끄러운(Seamless) 사용자 경험을 제공하게 될 것입니다.

3. 시사점: 기업 및 개발자를 위한 제언

본 분석을 통해 도출된 시사점은 다음과 같습니다. 기업 및 서비스 개발자들은 사용자의 '파일 생성' 요구를 단순한 기능적 요구사항으로 치부할 것이 아니라, **AI 에이전트의 실행 환경(Runtime Environment)을 어떻게 설계할 것인가**에 대한 아키텍처적 과제로 인식해야 합니다.

단순히 텍스트를 잘 쓰는 모델을 넘어, **"행동할 수 있는 모델(Actionable Model)"**을 구축하는 것이 차세대 AI 시장의 패권을 결정짓는 핵심 요소가 될 것입니다. 이는 크레셈(CRESSEM)과 같은 첨단 기술 기업이 AI 비전 검사 솔루션에 AI 에이전트 기술을 접목할 때, 단순한 결함 탐지를 넘어 '결함 분석 보고서 자동 생성 및 공정 시스템 전송'과 같은 고차원적인 자동화 솔루션으로 나아갈 수 있는 중요한 기술적 이정표가 될 것입니다.

참고 자료

1. [백엔드 개발자의 시선으로 풀어본 Llm 내부 동작 원리: 6단계로 ...](https://tech.kakaopay.com/post/how-llm-works/)

2. [Ollama와 RAG를 활용한 PDF 기반 로컬 챗봇 LLM 만들기](https://god-logger.tistory.com/205)

3. [llm.pdf: PDF 파일에서 대용량 언어 모델을 실행하는 실험적 ...](https://aisharenet.com/ko/llmpdf/)

4. [Llm 사용하는 방법 - 5 파일/문서 업로드 — 더경이로운기술블로그](https://wonderful-isla.tistory.com/131)

5. [Guiding LLMs The Right Way: Fast, Non-Invasive Constrained Generation - arXiv.org](https://arxiv.org/html/2403.06988v1)

6. [Guiding LLMs The Right Way: Fast, Non-Invasive Constrained Generation](https://arxiv.org/abs/2403.06988)

7. [Architecture | EvanZhouDev/llm.pdf | DeepWiki](https://deepwiki.com/EvanZhouDev/llm.pdf/1.1-architecture)

8. [PDF Llm Mechanics and Architecture Gen Ai Series: Part 2](https://www.rcac.purdue.edu/files/training/GenAI2mechanicsarchitecture.pdf)

9. [PDF An Architecture for Integrating Large Language Models with Digital Twins and ...](https://yuchenxia.github.io/files/architectureforintegratinglargelanguagemodels.pdf)

10. [PDF A Comprehensive Guide to Large Language Model](https://www.solulab.com/wp-content/uploads/2024/03/LLLM-PDF.pdf)

11. [(PDF) Large Language Models: A Comprehensive Survey of its Applications, Challenges ...](https://www.researchgate.net/publication/372278221LargeLanguageModelsAComprehensiveSurveyofitsApplicationsChallengesLimitationsandFutureProspects)

12. [PDF Large language models: Technology, intelligence, and thought](https://link.springer.com/content/pdf/10.1007/s42524-025-5004-3.pdf)

13. llm.pdf (사내 자료)

14. GenAI2mechanicsarchitecture.pdf (사내 자료)

15. architectureforintegratinglargelanguagemodels.pdf (사내 자료)

16. LLLM-PDF.pdf (사내 자료)

17. s42524-025-5004-3.pdf (사내 자료)

18. pdf_인하대학교컴퓨터공학과의배승환20260509001.pdf (사내 자료)

19. ProfessorSeungHwanBaeReport.pdf (사내 자료)

20. pdfHBMHighBandwidt20260508001.pdf (사내 자료)

21. pdfHBMHighBandwidt20260508002.pdf (사내 자료)

22. 매뉴얼HBMAdvancedInspectionSyst20260509001.pdf (사내 자료)

23. pdf1서론HBM시장의패러다임변화20260508001.pdf (사내 자료)

24. 분석_크레셈CRESSEM20260509001.pdf (사내 자료)