AI 프로덕트 셋프플ELD 가이드

평가부터 배포, 신뢰성까지

ThakiCloud (Hyojung Han)

AI 프로덕트 셋프플ELD 가이드

평가부터 배포, 신뢰성까지

ThakiCloud (Hyojung Han)

Contents

1 평가 설계

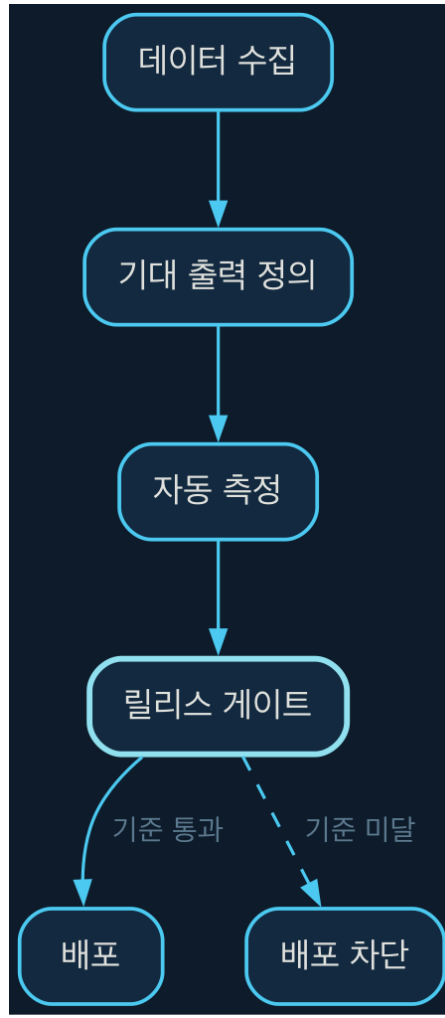

Figure 1.1: 평가 설계

AI 프로덕트를 배포할 때 가장 중요한 질문은 하나다. 이 모델이 충분히 좋은가. 이 장에서는 그 질문에 답하기 위해 필요한 평가 설계를 다룬다.

1.1 평가의 목적부터 명확히하기

평가는 개발이 아니라 의사결정을 위한 작업이다. 모델이 프로덕션에 적합한지 가르려면 판단 기준부터 있어야 한다. 기준 없이 평가하면 어떤 숫자가 나와도 해석할 수 없다.

평가 설계는 먼저 다음 질문에 답하며 원하는 동작을 정의하는 데서 시작한다.

어떤 입력이 들어오면 어떤 출력이 나와야 하는가. 허용할 수 있는 오답의 유형은 무엇인가. 오답이 나왔을 때 시스템은 어떻게 동작해야 하는가.

이 질문에 답하고 나면 평가 데이터셋을 어떻게 구성할지 방향이 잡힌다. 무작정 입력을 많이 모으기보다 의사결정에 실제로 영향을 미치는 입력에 집중해야 한다.

1.2 구조화된 평가 파이프라인 구성하기

평가 파이프라인은 네 단계로 나뉜다.

먼저 데이터 수집이다. 프로덕션에서 실제로 수집된 입력을 바탕으로 평가용 입력 집합을 구성한다. 이때 의사결정에 영향을 미치는 입력, 즉 모델 성능에 따라 비즈니스 결과가 달라지는 입력에 가중치를 준다.

다음은 기대 출력을 정의하는 단계다. 입력마다 모델이 내야 할 출력의 기준을 사람이 정한다. 이게 정답이다. 정답이 없으면 모델 출력이 좋은지 판단할 기준도 없다.

세 번째는 자동 측정이다. 사람이 정한 기대 출력과 모델의 실제 출력을 비교할 측정 도구를 만든다. 문자열 정확도만으로는 부족하다. 태스크 성격에 따라 정확도, 순위, 분류 결과, 생성 텍스트의 관련성 등에서 지표를 고른다.

마지막은 릴리스 게이트 설정이다. 측정 결과가 기준을 넘지 못하면 배포를 막는다. 이 게이트는 반드시 자동화되어 있어야 한다. 사람이 매번 판단하면 일관성이 떨어지고 속도도 느리다.

1.3 작은 데이터셋으로 신뢰할 수 있는 평가하기

현실에서는 평가 데이터셋을 무한히 모을 수 없다. 작은 데이터셋으로도 신뢰할 수 있는 평가를 하려면 몇 가지 원칙을 지켜야 한다.

먼저 표본의 다양성을 확보한다. 여러 입력 유형을 골고루 담아야 한다. 특정 유형에 치우친 데이터셋은 전체 성능을 실제보다 부풀리거나 낮춰 보이게 만든다.

다음은 오답 유형 분석이다. 정확도가 90%라면 나머지 10%의 오답이 어떤 유형인지 분류한다. 오답 유형을 알면 개선 방향이 뚜렷해진다. 결과가 우연히 좋게 나온 건지, 핵심 기능에서 실제로 잘 작동해서 좋은 건지 구분할 수 있다.

마지막은 샘플 크기 계산이다. 목표하는 정확도 차이를 검출하는 데 최소 몇 개의 샘플이 필요한지 통계적으로 계산한다. 이 계산이 있어야 데이터셋 크기를 근거 있게 정할 수 있다.

1.4 평가 결과를 신뢰할 수 있는 조건

평가 결과로 프로덕션 성능을 예측하려면 평가 데이터셋이 프로덕션 데이터의 분포를 반영해야 한다. 평가 데이터와 프로덕션 입력이 너무 다르면 결과가 왜곡된다.

새 모델 버전을 배포할 때는 늘 기존 모델과 새 모델의 평가 결과를 나란히 놓고 비교한다. 새 모델이 기존보다 못하면 배포를 멈춘다. 문제는 나중이 아니라 지금 찾아야 고치기 쉽다.

평가 파이프라인을 CI/CD에 통합하면 배포 여부를 사람이 아니라 시스템이 판단한다. 이 자동화가 일관된 품질 기준을 지키는 핵심이다.

2 프로덕션 파이프라인

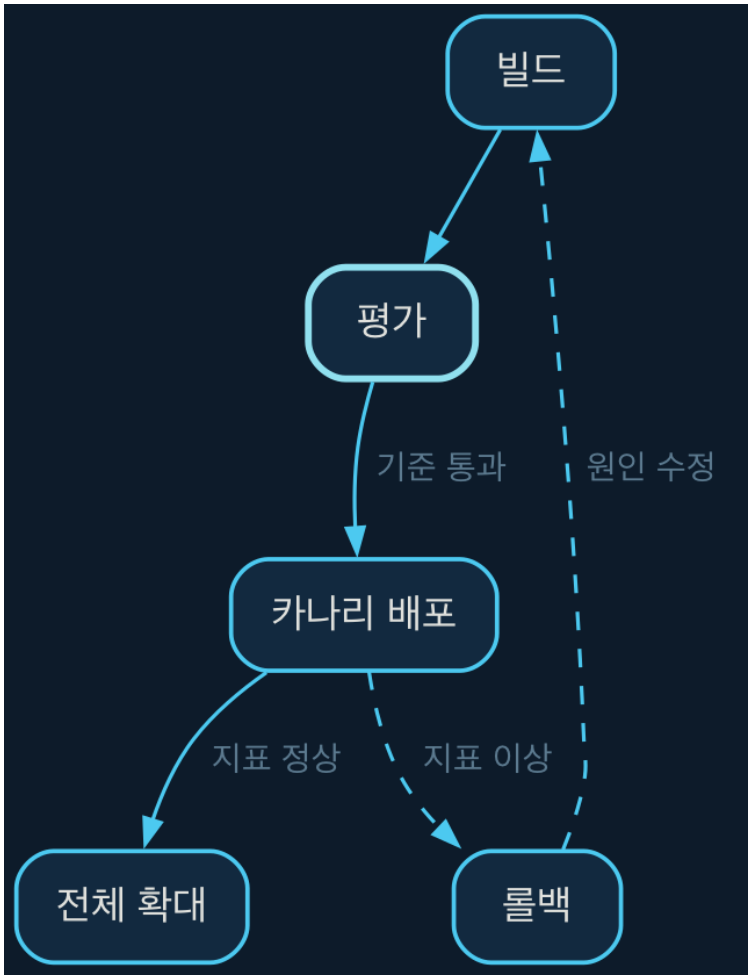

Figure 2.1: 프로덕션 파이프라인

평가를 통과한 모델을 실제 사용자에게 내보내려면 파이프라인이 필요하다. 파이프라인 구축은 모델을 단순히 Hosting하는 데서 그치지 않고, 신뢰할 수 있고 확장 가능한 서비스를 만드는 작업이다.

2.1 추론 서비스를 제공하는 파이프라인 아키텍처

AI 추론 서비스를 설계할 때 고려해야 할 축은 세 가지다. 지연 시간, 처리량, 비용이다. 셋을 동시에 최적화할 수는 없어서 어플리케이션 성격에 맞춰 우선순위를 정해야 한다.

대화형 어플리케이션에서는 지연 시간이 가장 중요하다. 사용자가 응답을 기다리는 시간이 길어질수록 체감 품질은 나빠진다. 반면 배치 처리 기반 서비스라면 처리량이 더 중요해진다.

파이프라인의 기본 구조는 전처리, 모델 추론, 후처리 세 단계로 나뉜다. 전처리 단계는 입력을 모델이 받아들일 수 있는 형태로 바꾸고, 모델 추론 단계는 실제 추론을 수행한다. 후처리 단계에서는 모델의 출력을 어플리케이션이 필요로 하는 형태로 다시 변환한다.

각 단계에서 일어날 수 있는 실패를 미리 정의하고 실패 시 동작을 명시해야 한다. 네트워크 오류, 메모리 부족, 모델 로드 실패 같은 다양한 장애 상황에 대응할 수 있어야 한다.

2.2 모델 버전 관리와 롤백 전략

모델도 소프트웨어의 일부이니 버전을 관리해야 한다. 버전 관리 없이 모델을 배포하면 문제가 생겼을 때 이전 상태로 되돌아갈 방법이 없다.

모델 버전 관리에는 세 가지 원칙이 따른다.

우선 모든 모델 버전을 저장한다. 모델 파일, 설정 값, 평가 결과, 배포 시점을 함께 기록해두면 문제가 생겼을 때 언제 어떤 모델이 배포되었는지 추적할 수 있다.

카나리와 블루프린트 배포를 활용하는 것도 중요하다. 카나리 배포는 트래픽의 일부에만 새 모델을 노출하고 문제가 없으면 전체로 확대하는

방식이고, 블루프린트 배포는 새 모델을 전체에 배포하되 이전 모델을 그대로 남겨두는 방식이다. 문제가 발견되면 즉시 이전 모델로 전환하면 된다.

마지막으로 롤백 조건을 사전에 정의해둔다. 어떤 지표가 어떤 값에 도달하면 롤백할지 명확히 정해야 하며, 지연 시간 급증이나 오류율 상승, 특정 입력 유형에서의 성능 저하 등이 롤백 트리거가 될 수 있다.

2.3 인프라 의존성 줄이기

AI 모델 추론에는 컴퓨팅 자원이 많이 든다. 하지만 모든 것을 고성능 GPU 인프라에 맡기면 비용은 오르고 가용성은 떨어진다.

추론 요구 수준에 따라 처리 방식을 나누는 편이 유용하다. 지연 시간에 민감한 중요 추론은 실시간으로 처리하고, 그렇지 않은 추론은 배치로 돌린다. 이렇게 하면 GPU 비용을 줄이면서도 핵심 서비스 품질은 지킬 수 있다.

모델을 경량화하는 것도 한 방법이다. 양자화나 가지치기처럼 모델 크기를 줄이는 기법을 적용하면 더 작은 리소스로도 추론이 가능해진다. 다만 정확도 손실이 허용 가능한 수준인지는 평가에서 반드시 확인해야 한다.

캐싱을 활용하는 것도 효과적이다. 같은 입력이 반복해서 들어올 때 이전 추론 결과를 재사용하면 불필요한 추론을 줄일 수 있다. 모든 입력마다 추론을 새로 수행할 필요는 없다.

2.4 배포 자동화의 파이프라인

모델 배포를 자동화하면 배포 속도가 빨라지고 사람의 실수도 줄어든다. 자동 배포 파이프라인은 대체로 다음 단계를 거친다.

먼저 빌드 단계에서 모델 아티팩트를 생성해 레지스트리에 저장한다.

이어서 평가 단계로 넘어가 자동 평가 파이프라인을 실행하고, 결과가 기준을 넘으면 배포 단계로 진행한다. 배포 단계에서는 미리 정의한 전략에 따라 모델을 서비스 환경에 내보낸다.

배포 후에는 모니터링 단계에서 서비스 지표와 모델 지표를 계속 추적한다. 이상이 발견되면 미리 정의해둔 롤백 절차가 실행된다. 이 모든 단계가 자동화되어 있어야 비로소 빠른 피드백 루프가 작동한다.

3 실패 처리

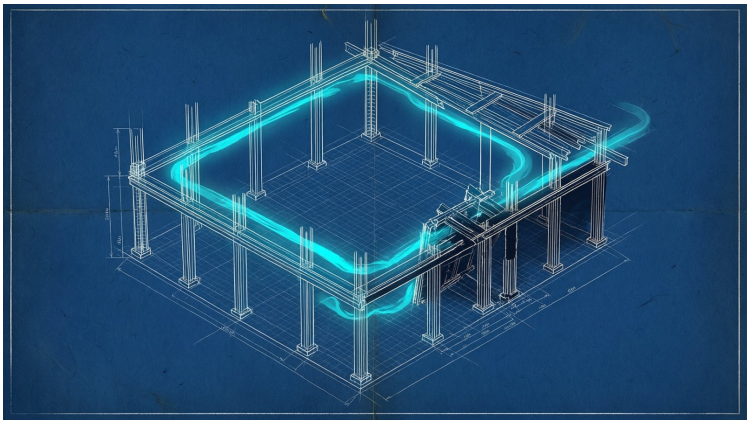

Figure 3.1: 실패 처리

AI 시스템은 전통적인 소프트웨어와는 다른 방식으로 실패한다. 코드가 정확히 명시된 연산을 그대로 수행하지 않는 경우가 흔하다. 입력의 Variation이 크고 모델의 동작 자체가 불확정적이기 때문이다. 이 장에서는 AI 시스템에서 나타나는 실패 유형과 그 대응 방법을 살펴본다.

3.1 AI 추론 실패의 유형

AI 추론 실패는 네 가지로 분류할 수 있다.

먼저 런타임 오류가 있다. 메모리 부족, GPU 고장, 네트워크 중단 등으로 추론 자체가 실패하는 경우로, 전통적인 소프트웨어와 같은 방식으로 다루면 된다. 재시도나 폴백, 회로 차단기 패턴을 적용하는 식이다.

다음은 출력 형식 오류다. 모델이 기대한 형식으로 결과를 내놓지 않는 경우인데, JSON 파싱 실패나 필수 필드 누락, 형식 불일치가 여기에 해당한다. 이럴 때는 입력을 다시 처리하거나 다른 추론 경로로 우회한다.

세 번째로는 품질 저하가 있다. 추론 자체는 성공했지만 출력의 품질이 기대에 못 미치는 경우다. 모델이 사실과 어긋나는 내용을 만들어내거나 입력의 의도를 잘못 파악하거나, 상황에 맞지 않는 출력을 내놓기도 한다.

마지막은 지연 시간 증가다. 평소보다 추론 시간이 갑자기 늘어나는 경우로, 서비스 수준 계약 위반으로 이어질 수 있어 꾸준히 모니터링할 필요가 있다.

3.2 부분 실패를 견디는 서비스 설계

AI 시스템은 완전히 동작하거나 완전히 멈추거나 둘 중 하나가 아니라, 부분적으로만 동작하는 경우가 많다. 예를 들어 여러 모델을 함께 쓰는 시스템이라면 그중 하나가 실패해도 나머지 모델의 출력으로 응답을 이어갈 수 있다.

이런 부분 실패에 대응하려면 서비스를 설계할 때 몇 가지를 염두에 두어야 한다.

우선 핵심 기능과 부가 기능을 나눈다. 모델 추론이 실패해도 핵심 기능만은 살아 있어야 한다. AI 응답 생성이 실패하더라도 검색 결과는 그대로 보여줄 수 있어야 한다는 뜻이다.

모델마다 실행 환경을 격리한다. 한 모델의 문제가 다른 모델까지 번지면 안 되기 때문이다. 각 모델을 독립된 프로세스나 컨테이너에서 돌리면 이런 격리를 구현할 수 있다.

Graceful Degradation도 구현해 둔다. 최선의 모델을 쓸 수 없을 때 차선의 옵션으로 넘어가는 방식으로, 시스템 전체가 한꺼번에 멈추는 사태를 막아준다.

3.3 재시도 정책 설계

일시적인 실패에는 재시도가 효과적이다. 다만 무분별한 재시도는 자원을 낭비하고 지연 시간만 늘릴 뿐이다. 효과적인 재시도 정책은 다음과 같이 설계한다.

먼저 재시도 조건을 명확히 정한다. 일시적 장애로 보일 때만 재시도하고, 영구적 장애라면 재시도해봐야 의미가 없다. GPU 메모리 부족은 대개 일시적이지만 모델 로드 실패는 영구적일 가능성이 높다는 점을 기준으로 삼을 수 있다.

지수적 백오프도 적용한다. 재시도 간격을 1초, 2초, 4초처럼 점점 늘려가는 방식이다. 이렇게 하면 장애가 갑자기 터졌을 때 모든 재시도가 한꺼번에 몰리는 상황을 피할 수 있다.

마지막으로 최대 재시도 횟수를 정해둔다. 무한정 재시도하면 시스템 전체가 블로킹될 수 있기 때문이다. 최대 횟수에 도달하면 실패로 처리하고 적절한 대안을 제공하면 된다.

3.4 실패를 학습의 기회로 만들기

실패 기록은 시스템을 개선하는 데 귀중한 자료가 된다. 실패한 입력과 출력을 데이터셋에 추가하면 모델이 그 유형을 학습하게 할 수 있다.

다만 실패 기록을 학습 데이터로 쓰려면 품질 관리를 거쳐야 한다. 잘못된 출력을 그대로 학습 데이터에 넣으면 모델이 그 오답을 오히려 강화할 위험이 있다. 사람이 검증한 데이터만 골라 학습에 사용하는 편이 안전하다.

실패 패턴은 주기적으로 분석하는 습관도 필요하다. 특정 유형의 입력이 반복해서 실패한다면 시스템 자체에 구조적인 문제가 있을 가능성이 크다. 이럴 때는 모델을 수정하거나 입력 전처리를 개선하고, 경우에 따라서는 그 유형의 입력을 아예 처리하지 않겠다고 명시적으로 선언하는 방법도 있다.

4 옵저버빌리티

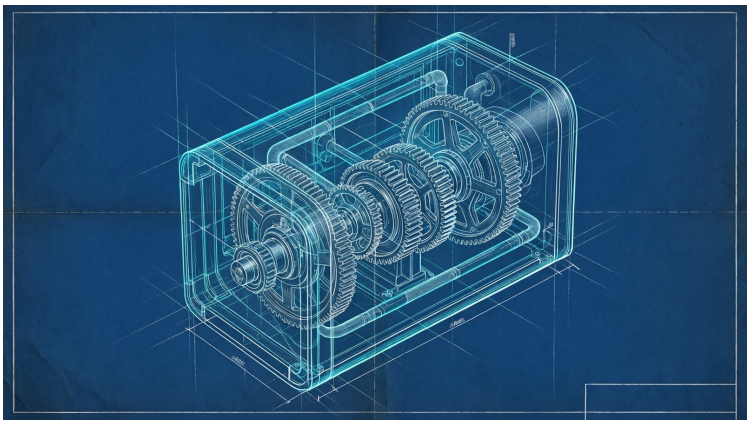

Figure 4.1: 옵저버빌리티

AI 시스템이 프로덕션에서 실제로 어떻게 움직이는지 파악하려면 옵저버빌리티가 필수다. 시스템 내부 상태를 외부에서 관찰할 수 있는 능력을 가리킨다.

4.1 관측 가능한 신호 설계하기

AI 시스템에서 관측 가능한 신호는 크게 세 갈래로 나눌 수 있다.

우선 시스템 신호가 있다. CPU 사용률, 메모리 사용량, GPU 활용률, 네트워크 지연 시간처럼 인프라 수준에서 잡히는 지표들이다. 평상시 값에서 크게 벗어나면 시스템 이상을 의심할 수 있다.

다음은 모델 신호다. 추론 지연 시간, 추론 성공률, 입력 크기 분포, 출력 크기 분포처럼 모델 동작 자체를 보여주는 지표로, 성능 변화를 추적하는 데 쓴다.

마지막으로 Business 신호다. 모델 출력이 Business 결과에 미치는 영향을

나타내며, AI 응답 후 사용자 전환율, AI 추천 클릭률, AI 대화 길이 등이 여기에 속한다. 모델 성능이 실제 Business에 어떤 영향을 주는지 이 신호로 확인할 수 있다.

이 세 신호를 함께 추적해야 시스템 전체를 이해할 수 있다. 시스템 신호만 보면 모델 성능 변화를 감지하기 어렵고, Business 신호만 보면 원인 분석이 힘들다.

4.2 모델 성능 저하 감지하기

AI 모델은 시간이 지나면서 성능이 떨어지기도 하는데, 이를 모델 드리프트라 부른다. 데이터 분포가 바뀌거나 사용 패턴이 달라지면 성능 저하로 이어질 수 있다.

드리프트를 감지하려면 먼저 기준선을 설정해야 한다. 배포 시점의 성능을 기준선으로 삼고, 새로운 성능 데이터가 여기서 유의미하게 벗어나면 알림을 발생시킨다.

드리프트 감지는 두 수준에서 이루어진다. 데이터 드리프트는 입력 데이터의 분포 변화를 감지하고, 예측 드리프트는 모델 출력 분포의 변화를 감지한다. 둘 다 모니터링 시스템으로 추적해야 한다.

데이터 드리프트가 감지되면 데이터 수집 파이프라인에 문제가 있을 수 있고, 예측 드리프트가 감지되면 모델 재학습이 필요한 시점일 수 있다. 두 경우 모두 즉시 조치가 필요하지는 않지만, 꾸준한 모니터링으로 추세는 파악해야 한다.

4.3 디버깅을 위한 로깅 전략

AI 시스템에서 문제가 생겼을 때 원인을 찾으려면 적절한 로깅이 필요하다. 모든 추론의 입력과 출력을 기록하면 좋겠지만 비용과 저장소 제약이

따른다.

효과적인 로깅 전략은 다음 세 가지로 정리할 수 있다.

먼저 실패한 추론을 기록한다. 성공한 추론보다는 실패한 추론의 원인을 분석하는 쪽이 더 중요하므로, 입력과 출력, 오류 유형, 발생 시점을 함께 남긴다.

다음으로 이상치 추론을 기록한다. 출력 길이가 극단적으로 길거나 짧은 경우, 특정 키워드가 유난히 많거나 적은 경우처럼 이상한 출력을 남긴 추론이 대상이다. 아직 오류로 분류되지는 않았지만 모델의 잠재적 문제를 드러내는 신호다.

마지막으로 일부 추론을 샘플링해 기록한다. 모든 추론을 남길 수 없다면 일정 비율만큼 무작위로 뽑아 기록하고, 이 방식으로 시스템 전체의 동작을 가늠할 수 있다.

기록하는 정보에 민감한 내용이 섞이지 않도록 주의해야 한다. 입력에 개인정보가 있다면 마스킹하거나 제외한다.

4.4 대시보드와 알림 설계

수집한 지표를 활용하려면 시각화와 알림이 필요하다.

대시보드는 서비스 운영에 필요한 정보만 보여줘야 한다. 핵심 지표를 한눈에 파악할 수 있어야 하고, 이상 징후는 즉시 알아챌 수 있어야 한다. 정보를 지나치게 많이 욱여넣으면 오히려 중요한 신호를 놓치기 쉽다.

알림은 실제로 조치가 필요한 경우에만 보내야 한다. 매번 알림이 오면 경고를 무시하게 된다. 알림의 조건과 심각도를 명확히 정의하고, 불필요한 알림은 제거한다.

알림을 받았을 때 따라야 할 대응 절차는 미리 문서화해둔다. 누구에게 연락할지, 어떤 초기 조치를 취할지, 언제 Escalate할지를 정해두면 문제

대응 속도가 빨라진다.