5노드 AI 에이전트를 굴리며 실제로 터진 실패패턴 20선

부제: 프롬프트보다 먼저 설계해야 하는 운영 forcing function

이 문서는 실제 5노드 AI 에이전트 운영 중 쌓인 192개 운영 이슈에서 반복성이 높은 실패패턴 20개를 뽑은 것이다. 목표는 겁주기가 아니다. AI 에이전트를 실제 업무에 붙일 때 어떤 종류의 확인 경로와 forcing function이 필요한지 빠르게 점검하게 만드는 것이다.

핵심 thesis는 단순하다.

모델이 더 똑똑해져서 풀릴 문제가 아니라, 확인 경로와 forcing function을 설계해야 막히는 문제가 있다.

A. 도달과 확인 실패

1. 사용자가 보는 채널이 아닌 곳에 답한다

사고 한 줄: 사용자는 메신저에서 기다리는데 에이전트는 터미널에만 답을 써서 사용자는 아무것도 받지 못했다.

왜 모델로 안 풀리나: 모델은 답을 작성하면 완료라고 느낀다. 운영에서는 답이 사용자에게 도착해야 완료다.

forcing function: 입력 origin과 응답 channel을 묶고, 해당 channel 응답이 없으면 turn 종료를 막는다.

2. 크로스디바이스 보고를 내부 보고로 착각한다

사고 한 줄: 노드 보고는 처리했지만 사용자가 보는 채널로 2차 요약을 보내지 않아 사용자는 진행 상태를 몰랐다.

왜 모델로 안 풀리나: 노드 간 보고와 사용자 보고는 목적지가 다르다. 하나를 했다고 다른 하나가 자동으로 해결되지 않는다.

forcing function: cross-device report를 받은 turn에는 노드 reverse reply와 사용자-facing reply를 별도 체크한다.

3. "명령 성공"을 "대상 도달"로 착각한다

사고 한 줄: wrapper는 성공했지만 실제 세션, 입력창, mirror 채널에는 도착하지 않았다.

왜 모델로 안 풀리나: 중간 계층의 성공 코드는 최종 대상 상태를 보장하지 않는다.

forcing function: send 뒤에는 target capture, mirror count, live status 같은 도달 검증을 별도 단계로 둔다.

4. hook 의존성이 없어도 조용히 실패한다

사고 한 줄: 강제 응답 hook이 설치돼 있어도, 의존 도구가 없어서 조용히 실패했다.

왜 모델로 안 풀리나: 모델은 "hook이 있다"는 구조만 기억하기 쉽다. 런타임 의존성이 빠지면 구조는 존재하지만 작동은 하지 않는다.

forcing function: 노드 onboarding과 세션 시작 시 hook 의존성을 검사하고, 누락 시 경고를 크게 띄운다.

5. 컨텍스트 사용률을 체감으로 추정한다

사고 한 줄: 실제 사용량은 낮았는데, 체감으로 "한도에 가까워졌다"고 판단해 handoff와 clear를 강행했다.

왜 모델로 안 풀리나: turn 수와 token 사용량은 다르다. 모델은 자기 컨텍스트 사용률을 안정적으로 체감할 수 없다.

forcing function: context 관련 판단은 실측값만 쓴다. 추정 문구는 금지하고, 임계치 판단은 측정 도구나 hook이 맡는다.

B. stale과 근거 실패

6. 한 노드만 보고 "없다"고 단정한다

사고 한 줄: 한 기기의 로컬 secrets만 보고 "없다"고 단정했지만, SoT 노드에는 존재했다.

왜 모델로 안 풀리나: "없다"는 말은 전체 검색 범위를 전제로 한다. 부분 검색만으로는 negative assertion을 할 수 없다.

forcing function: SoT와 reachable 노드를 전수 확인한다. 닿지 않은 노드는 "미확인"으로 분리한다.

7. 핸드오프 메모를 사실로 취급한다

사고 한 줄: archive handoff 1줄을 믿고 directive를 작성했는데 실제 PR diff와 달랐다.

왜 모델로 안 풀리나: 메모는 context hint다. 시간이 지나면 코드와 task 상태가 바뀐다.

forcing function: directive 전 실제 파일, git diff, PR 상태를 확인한다.

8. 완료된 결정이 죽인 형제 task를 안 닫는다

사고 한 줄: 최종 결정은 끝났지만 대안 task가 열려 있어 자동 워커가 stale 작업을 다시 실행했다.

왜 모델로 안 풀리나: 열린 task 목록은 의미 관계를 모른다. "A가 끝났으니 B도 무효"라는 관계를 명시해야 한다.

forcing function: 완료 task와 함께 superseded task를 닫고, dispatch 전 의미 기반 staleness gate를 둔다.

9. 자동화 큐가 열린 task 텍스트만 읽고 의미를 모른다

사고 한 줄: `[ ]` 또는 `[~]` 상태만 보고 이미 무효화된 작업을 다시 집었다.

왜 모델로 안 풀리나: 큐는 상태 box를 읽는다. 결정 로그와 task 의미를 자동으로 결합하지 않는다.

forcing function: 최근 완료/취소 결정과 후보 task를 dispatch 직전에 대조한다.

C. 추적과 라이프사이클 실패

10. task ID가 커밋과 PR에 안 실린다

사고 한 줄: 수동큐 작업이 task ID 없이 내려가 커밋과 PR이 만들어졌고, 자동 완료 sweep이 task를 닫지 못했다.

왜 모델로 안 풀리나: 자동화는 공통 키가 없으면 같은 작업임을 알 수 없다.

forcing function: 브랜치, 커밋, PR 제목에 task ID를 박는다. task가 없는 작업은 `[no-task]`로 의도적 예외를 표시한다.

11. "발행/제출/발송" 뒤 검증과 ledger가 없다

사고 한 줄: 외부 작업은 성공처럼 보여도 실제 공개 상태, 심사 상태, 수신 상태가 어긋날 수 있다.

왜 모델로 안 풀리나: 외부 시스템은 async이고, 성공 응답과 최종 상태가 분리된다.

forcing function: 외부영향 직후 action ledger와 live verification을 한 세트로 묶는다.

D. 환경과 도구 함정

12. 테스트 하네스가 production push를 한다

사고 한 줄: 테스트 fixture를 올린 줄 알았는데 리뷰 전 commit까지 main에 push됐다.

왜 모델로 안 풀리나: git은 파일 단위 의도보다 commit graph를 따른다. 같은 branch에서 테스트와 배포를 섞으면 경계가 사라진다.

forcing function: 테스트는 별도 ref, 별도 worktree, 별도 fixture repo에서만 production-like push를 수행한다.

13. 복붙 블록 안 placeholder가 실제 값처럼 실행된다

사고 한 줄: `<여기에 토큰>` 같은 placeholder가 그대로 실행되어 설정 파일을 망가뜨렸다.

왜 모델로 안 풀리나: 사람에게는 placeholder가 명확해 보여도, 실행 가능한 블록 안에 있으면 그대로 붙여넣기 쉽다.

forcing function: 그대로 실행할 명령 블록에는 placeholder를 넣지 않는다. 비밀값과 환경별 값은 별도 경로로 전달한다.

14. 문자열을 byte 단위로 잘라 UTF-8을 깨뜨린다

사고 한 줄: Linux 환경에서 byte 단위 문자열 절단이 한글/이모지를 중간에서 잘라 API가 거부했다.

왜 모델로 안 풀리나: macOS와 Linux coreutils 동작 차이는 기억만으로 안정적으로 처리되지 않는다.

forcing function: 멀티바이트 문자열 절단은 글자 단위 도구를 사용하고, macOS/BSD와 GNU 양쪽에서 검증한다.

15. family 파일 중 하나만 고친다

사고 한 줄: 같은 상수를 쓰는 파일 family 중 하나만 고쳐 다음 실행 경로가 stale 설정을 계속 썼다.

왜 모델로 안 풀리나: 모델은 지금 보이는 파일을 고치기 쉽다. 같은 책임을 공유하는 파일 family를 자동으로 전수 확인하지 않는다.

forcing function: 관련 상수/함수명으로 targeted grep을 하고 family 파일을 한 묶음으로 패치한다.

16. GUI 자동화가 기존 사용자 세션과 합쳐진다

사고 한 줄: 자동화 브라우저가 기존 사용자 브라우저 세션과 합쳐져 제어 가능한 브라우저를 잡지 못했다.

왜 모델로 안 풀리나: OS 앱 singleton, 프로필, binary path 같은 GUI 런타임 모델은 코드만 봐서는 드러나지 않는다.

forcing function: 자동화용 브라우저 binary와 profile을 사용자 세션과 분리한다.

17. liveness 판정이 false negative/false positive를 낸다

사고 한 줄: 살아있는 노드를 죽었다고 보거나, idle 화면 문구를 busy로 오판해 작업이 배정되지 않았다.

왜 모델로 안 풀리나: 단일 regex는 UI 문구 변화와 OS별 PATH 차이에 취약하다.

forcing function: liveness를 `RUNNING / ABSENT / UNREACHABLE`처럼 3상태로 반환하고, pane 내용과 프로세스 상태를 함께 본다.

E. 보안과 정책 실패

18. 정당한 자동화가 정책 false-positive에 걸린다

사고 한 줄: 정상적인 개인 자동화 directive였지만 표현이 위험하게 읽혀 provider 정책 분류기에 걸렸다.

왜 모델로 안 풀리나: 안전한 의도도 표면 텍스트가 위험하게 보이면 차단될 수 있다.

forcing function: 안전 맥락을 명시한다. 사용자 소유 자산, 권한 범위, 외부 영향 여부, 목적을 분명히 쓰고 민감한 단어를 불필요하게 나열하지 않는다.

19. 비밀값과 토큰이 prompt, history, grep 출력에 노출된다

사고 한 줄: 토큰과 PAT가 파일, 대화, git history에 남아 rotate가 필요해졌다.

왜 모델로 안 풀리나: 모델은 "검증을 위해 보여달라"는 습관이 있고, 로그와 prompt는 생각보다 오래 남는다.

forcing function: secret 검색은 파일명과 key만 출력한다. 값은 출력하지 않는다. 노출이 확인되면 rotate를 완료 조건에 넣는다.

20. 공개 상태와 내부 상태가 다른데 "완료"로 닫는다

사고 한 줄: 내부 빌드나 업로드는 끝났지만 실제 심사, 공개, 수신 상태가 다른 경우가 있었다.

왜 모델로 안 풀리나: 외부 시스템은 내부 작업 완료와 공개 완료를 분리한다.

forcing function: 완료 조건을 "명령 성공"이 아니라 "사용자에게 영향을 주는 최종 상태 확인"으로 정의한다.

결론

AI 에이전트 운영에서 제일 위험한 문장은 "됐습니다"다. 됐다는 말은 너무 자주 "내 도구가 에러를 안 냈다"는 뜻으로 쓰인다. 실제 운영에서는 사용자가 봤는지, 대상에 도착했는지, 최신 사실과 맞는지, 같은 작업이 다시 살아나지 않는지까지 확인해야 한다.

좋은 에이전트 시스템은 프롬프트가 긴 시스템이 아니다. 실패가 반복될 때마다 그 실패를 막는 작은 forcing function이 하나씩 늘어나는 시스템이다.

부록으로 1페이지 체크리스트를 붙인다. 이 체크리스트를 통과하지 못하는 자동화는 아직 production 운영에 붙일 준비가 덜 된 것이다.

AI 에이전트 운영 forcing function 체크리스트

상태: PDF 부록 1페이지 초안

---

1. 도달

- 사용자가 입력한 channel과 응답 channel이 연결되어 있는가?
- 답변 작성 성공이 아니라 실제 사용자-facing delivery를 확인하는가?
- cross-device 보고와 사용자 보고를 별도 완료 조건으로 보는가?

- send 후 target capture, mirror count, live status 중 하나 이상을 확인하는가?

---

2. 근거

- "없다"는 보고 전에 SoT와 reachable 노드를 전수 확인했는가?
- handoff, memory, archive를 hint로만 취급하고 최신 파일/PR/task를 다시 봤는가?
- stale 가능성이 있는 상태 전제는 실행 직전에 재검증했는가?

---

3. 라이프사이클

- task ID가 branch, commit, PR, report 중 최소 하나 이상에 들어가는가?
- task가 없는 작업은 `[no-task]`처럼 예외 마커를 남기는가?
- 완료된 결정이 무효화한 형제 task를 같이 닫는가?
- 외부영향 작업 직후 ledger와 live verification을 남기는가?

---

4. 환경

- hook과 worker의 런타임 의존성을 세션 시작 시 검사하는가?
- macOS/BSD와 Linux/GNU 차이를 실제 양쪽에서 확인했는가?
- GUI 자동화는 사용자 브라우저/프로필과 분리되어 있는가?
- liveness는 단일 regex가 아니라 3상태 또는 다중 신호로 판단하는가?

---

5. 보안과 정책

- secret 검색 결과에서 값이 출력되지 않도록 막았는가?
- 토큰, PAT, key가 prompt, log, git history에 들어가지 않게 preflight가 있는가?
- 정상 자동화의 목적, 권한 범위, 사용자 소유 자산, 외부 영향 여부가 명확히 쓰였는가?
- provider 정책상 오해될 표현을 줄이고 안전 맥락을 설명하는 표현인가?

최소 통과 기준

운영 자동화에 붙이기 전 아래 5개는 반드시 통과해야 한다.

1. 응답/보고가 실제 사용자-facing channel에 도달한다.

2. stale 방지를 위해 실행 직전 최신 fact를 한 번 확인한다.

3. 작업 증거가 task, commit, PR, ledger 중 하나로 연결된다.

4. 실패해도 조용히 넘어가지 않고 surface된다.

5. secret 값과 정책 민감 정보가 로그와 prompt에 남지 않는다.

실패패턴 20선 근거 맵

검증일: 2026-06-20 KST

SoT: `~/claude-skills/issues/`

실측 기준:

- `~/claude-skills/issues/*.md` = 192개 (`INDEX.md` 제외)
- `~/daejong-page/public/issues/*.md` = 157개 (`INDEX.md` 제외)

A. 도달과 확인 실패

1. 사용자가 보는 채널이 아닌 곳에 답한다

근거: `2026-04-20-terminal-only-reply-missed-telegram.md`

2. 크로스디바이스 보고를 내부 보고로 착각한다

근거: `2026-05-19-mac-report-telegram-reply-missed.md`

3. "명령 성공"을 "대상 도달"로 착각한다

근거: `2026-06-08-directive-submit-stuck.md` , `2026-05-08-codex-inject-websocket-silent-fail.md`

4. hook 의존성이 없어도 조용히 실패한다

근거: `2026-04-20-terminal-only-reply-missed-telegram.md` , `2026-05-20-wsl-bot-bare-path-no-poll.md`

5. 컨텍스트 사용률을 체감으로 추정한다

근거: `2026-05-24-context-percent-mis-estimation.md`

B. stale과 근거 실패

6. 한 노드만 보고 "없다"고 단정한다

근거: `2026-06-07-premature-credential-not-found.md`

7. 핸드오프 메모를 사실로 취급한다

근거: `2026-05-29-autopilot-oauth-directive-stale-assumption.md`

8. 완료된 결정이 죽인 형제 task를 안 닫는다

근거: `2026-06-04-stale-orphan-task-redispatch.md`

9. 자동화 큐가 열린 task 텍스트만 읽고 의미를 모른다

근거: `2026-06-04-stale-orphan-task-redispatch.md`, `2026-06-06-dispatcher-staleness-gate-design.md`

C. 추적과 라이프사이클 실패

10. task ID가 커밋과 PR에 안 실린다

근거: `2026-06-09-task-mark-no-taskid.md`

11. "발행/제출/발송" 뒤 검증과 ledger가 없다

근거: `2026-05-09-play-managed-publishing-on.md`, `2026-06-16-bonjin-comms-card-and-telegram.md`

D. 환경과 도구 함정

12. 테스트 하네스가 production push를 한다

근거: `2026-05-12-fleet-state-tests-auto-push-leak.md`

13. 복붙 블록 안 placeholder가 실제 값처럼 실행된다

근거: `2026-05-10-placeholder-paste-loss.md`

14. 문자열을 byte 단위로 잘라 UTF-8을 깨뜨린다

근거: `2026-06-01-directive-mirror-utf8-400-cut-bytes.md`

15. family 파일 중 하나만 고친다

근거: `2026-05-26-five-node-cascade-failure.md`

16. GUI 자동화가 기존 사용자 세션과 합쳐진다

근거: `2026-05-26-five-node-cascade-failure.md`

17. liveness 판정이 false negative/false positive를 낸다

근거: `2026-05-30-macmini-tmux-liveness-false-negative.md`, `2026-06-15-worker-tokens-false-busy.md`

E. 보안과 정책 실패

18. 정당한 자동화가 정책 false-positive에 걸린다

근거: `2026-05-25-claude-cyber-verification-block.md`, `2026-05-26-five-node-cascade-failure.md`

19. 비밀값과 토큰이 prompt, history, grep 출력에 노출된다

근거: 내부 운영 이슈 3건(토큰 교체, 자동 루틴 secret 노출, 로그 출력 redaction). 실제 값은 redaction 확인 후 비공개.

20. 공개 상태와 내부 상태가 다른데 "완료"로 닫는다

근거: `2026-05-09-play-managed-publishing-on.md`, `2026-06-16-bonjin-comms-card-and-telegram.md`

공개 링크 기준

근거 파일명은 내부 SoT 기준이다. 공개 미러에 없는 항목은 이 문서에서 링크하지 않았다.