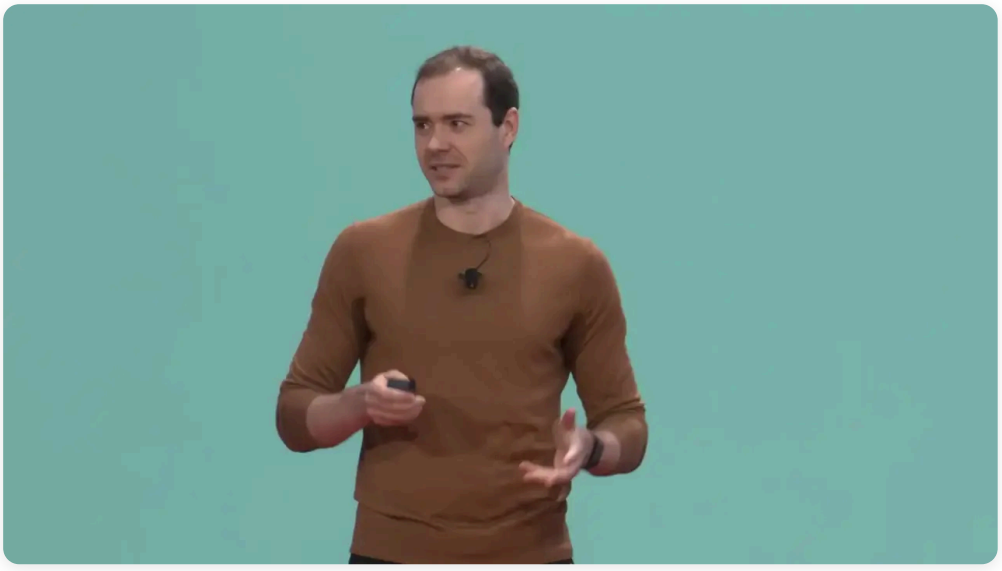
Andrej Karpathy가 말하는 소프트웨어의 미래: 'Software Is Changing (Again)' 발표 심층 분석

게시일: 2025년 9월 15일

OpenAI의 창립 멤버이자 Tesla의 AI 디렉터를 역임한 인공지능 분야의 석학, 안드레이 카파시(Andrej Karpathy)는 최근 YC AI Startup School에서 'AI 시대의 소프트웨어(Software in the era of AI)'라는 주제로 강연을 진행했습니다. 이 강연에서 그는 소프트웨어 개발 패러다임이 근본적으로 다시 한번 변화하고 있으며, 우리는 '소프트웨어 3.0'이라는 새로운 시대에 진입했다고 선언했습니다. 이 글에서는 그의 발표 내용을 바탕으로 소프트웨어의 미래가 어떻게 펼쳐질지, 그리고 개발자들은 어떤 관점을 가져야 할지에 대해 심도 있게 분석해보고자 합니다.



소프트웨어는 다시 변하고 있다 - 발표의 핵심 주제 선언

카파시는 지난 70년간 소프트웨어의 근본적인 형태는 크게 변하지 않았지만, 최근 몇 년 사이 두 번의 급격한 변화를 겪었다고 말합니다. 첫 번째는 뉴럴 네트워크의 등장으로 인한 '소프트웨어 2.0'이며, 두 번째는 바로 지금, 대규모 언어 모델(LLM)이 이끄는 '소프트웨어 3.0'입니다. 이 거대한 변화의 물결 속에서 우리는 무엇을 준비하고 어떤 기회를 포착해야 할까요?

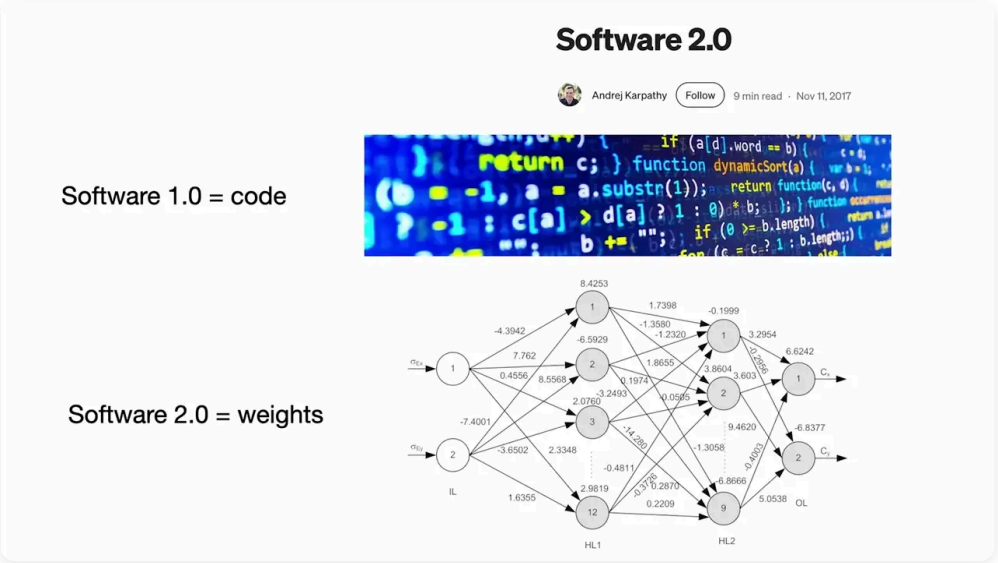
목차

1. 소프트웨어 진화의 3단계: 코드, 가중치, 그리고 프롬프트

카파시는 소프트웨어의 발전을 세 가지 단계로 명확하게 구분합니다. 이 프레임워크를 이해하는 것은 현재 우리가 겪고 있는 변화의 본질을 파악하는 데 매우 중요합니다.

- **소프트웨어 1.0 (코드):** 인간이 Python, C++, Java와 같은 프로그래밍 언어를 사용해 직접 명령어를 작성하는 전통적인 방식입니다. GitHub에 존재하는 대부분의 코드가 여기에 해당합니다.
- **소프트웨어 2.0 (가중치):** 방대한 데이터셋을 이용해 뉴럴 네트워크를 학습시키고, 그 결과물인 '가중치(weights)'가 소프트웨어의 역할을 하는 방식입니다. 이미지 인식 모델이나 번역 모델 등이 대표적인 예입니다. 개발자는 코드를 직접 짜는 대신, 데이터셋을 큐레이팅하고 최적화 알고리즘을 실행합니다.

- **소프트웨어 3.0 (프롬프트):** LLM에 자연어(예: 영어, 한국어)로 된 '프롬프트(prompt)'를 입력하여 원하는 기능을 수행하도록 프로그래밍하는 새로운 방식입니다. 이는 LLM이 단순한 고정 기능의 뉴럴 네트워크를 넘어 '프로그래밍 가능한' 존재가 되었기에 가능해졌습니다.



소프트웨어 1.0, 2.0, 3.0 개념 - 코드, 가중치, 프롬프트로의 진화

> *"가장 인기 있는 새로운 프로그래밍 언어는 영어입니다(The hottest new programming language is English)."*

카파시는 이 트윗을 인용하며 소프트웨어 3.0 시대의 가장 놀라운 점은 프로그래밍 언어가 인간의 모국어, 즉 자연어가 되었다는 사실이라고 강조합니다. 이제 코드를 모르는 사람도 자신의 생각을 글로 표현함으로써 컴퓨터를 프로그래밍할 수 있게 된 것입니다.

소프트웨어 2.0이 1.0을 잠식하다

이러한 패러다임 전환이 어떻게 일어나는지를 설명하기 위해, 카파시는 테슬라 오토파일럿 개발 경험을 예로 듭니다. 초기의 오토파일럿 시스템은 수많은 C++ 코드(소프트웨어 1.0)로 이루어져 있었습니다. 하지만 시스템이 발전하면서 이미지 인식, 경로 계획 등 더 많은 기능이 뉴럴 네트워크(소프트웨어 2.0)로 대체되었습니다. 결국, 기존의 C++ 코드는 점차 삭제되고 그 자리를 뉴럴 네트워크의 가중치가 차지하게 되었습니다. 카파시는 이를 "소프트웨어 2.0이 소프트웨어 1.0 스택을 먹어치웠다"고 표현합니다.

소프트웨어 2.0이 1.0을 잠식하다 - Tesla 오토파일럿 사례

그리고 지금, LLM(소프트웨어 3.0)이 바로 이와 동일한 방식으로 기존의 소프트웨어 1.0과 2.0을 잠식하기 시작했습니다. 이제 개발자들은 세 가지 패러다임(코드, 가중치, 프롬프트)을 모두 이해하고, 주어진 문제에 가장 적합한 방식을 유연하게 선택하고 조합할 수 있어야 합니다.

2. LLM에 대한 새로운 관점: 단순한 도구가 아닌 '새로운 운영체제'

카파시는 LLM의 본질을 이해하기 위해 여러 가지 비유를 제시합니다. 그중 그가 가장 적절하다고 생각하는 비유는 바로 LLM을 '새로운 종류의 운영체제(Operating System)'로 보는 것입니다.

다양한 비유: 전기, 반도체 공장, 그리고 OS

- **유틸리티(전기):** LLM은 전기처럼 인프라의 성격을 가집니다. 거대한 자본 지출(CAPEX)로 모델을 훈련하고(송전망 건설), 운영 비용(OPEX)을 들여 API를 통해 지능을 제공합니다. 우리는 전기처럼 저지연, 고가용성, 안정적인 품질을 요구하며, OpenAI가 다운되면 '지능의 정전(intelligence brownout)'을 경험합니다.
- **반도체 공장(Fab):** LLM 개발은 막대한 R&D와 비밀 기술이 필요한 반도체 산업과 유사합니다. NVIDIA GPU로 모델을 훈련하는 기업은 팹리스(Fabless) 기업과 같고, 자체 TPU를

개발하는 구글은 자체 팹을 가진 인텔과 같습니다.

- **운영체제(OS):** 하지만 카파시는 LLM이 단순한 상품(commodity)이 아니라, 점점 더 복잡해지는 소프트웨어 생태계라는 점에서 OS 비유가 가장 적절하다고 말합니다. Windows/macOS와 같은 폐쇄형 소스(GPT, Claude)와 Linux와 같은 오픈소스(Llama)가 경쟁하는 구도가 이미 형성되고 있습니다. LLM의 코어는 CPU, 컨텍스트 창은 메모리(RAM)와 유사하며, 이 자원을 활용해 다양한 기능 모듈(툴)을 조작하여 문제를 해결하는 모습은 영락없는 OS입니다.

1960년대 시분할 컴퓨팅 시대로의 회귀

더 나아가, 그는 현재 LLM 환경이 1960년대의 메인프레임 및 시분할(time-sharing) 컴퓨팅 시대와 매우 흡사하다고 지적합니다. LLM의 연산 비용은 아직 매우 비싸기 때문에 클라우드에 중앙 집중화되어 있고, 우리 모두는 네트워크를 통해 접속하는 '씬 클라이언트(thin client)'에 불과합니다. 아직 '개인용 컴퓨터(PC) 혁명'은 일어나지 않은 것입니다. 우리가 ChatGPT와 텍스트로 대화하는 것은 마치 GUI가 발명되기 전, 터미널을 통해 OS와 직접 상호작용하는 것과 같습니다.

기술 확산의 역전 현상

그러나 한 가지 매우 독특한 차이점이 있습니다. 과거의 혁신적인 기술(전기, 컴퓨터, 인터넷 등)은 정부나 기업이 먼저 사용하고 나중에 소비자에게 확산되었습니다. 하지만 LLM은 정반대입니다. 소비자들이 먼저 '계란 삶는 법'을 묻는 데 사용하고, 오히려 기업과 정부가 이 기술을 도입하는 데 뒤처지고 있습니다. 카파시는 이를 '기술 확산의 역전'이라 부르며, 이는 LLM이 가진 전례 없는 특징이라고 분석합니다.

3. LLM의 심리학: '결함 있는 슈퍼맨'을 이해하는 법

LLM을 효과적으로 프로그래밍하기 위해서는 그들의 '심리'를 이해해야 합니다. 카파시는 LLM을 '인간 정신의 확률적 시뮬레이션'으로 간주하며, 그들이 가진 초인적인 능력과 동시에 나타나는 인지적 결함을 분석합니다.

초인적인 능력과 인지적 결함

- **초인적 기억력:** LLM은 인터넷의 방대한 텍스트로 학습했기 때문에, 어떤 인간보다도 훨씬 많은 것을 기억합니다. 마치 영화 '레인맨'의 주인공처럼 전화번호부 전체를 외울 수 있는 '서번트 증후군' 환자와 같습니다.
- **환각(Hallucination):** 하지만 동시에 그들은 매우 자주 사실이 아닌 내용을 그럴듯하게 지어냅니다.
- **들쭉날쭉한 지능(Jagged Intelligence):** 어떤 분야에서는 초인적인 문제 해결 능력을 보이지만, '딸기(strawberry)에는 r이 두 개 있다'고 주장하거나 '9.11이 9.9보다 크다'고 틀리는 등 인간이라면 절대 하지 않을 기본적인 실수를 저지릅니다.
- **선행성 기억상실(Anterograde Amnesia):** LLM은 대화가 끝나면 모든 것을 잊어버립니다. 컨텍스트 창은 '작업 기억(working memory)'일 뿐, 새로운 경험을 통해 영구적으로 더 똑똑해지지 않습니다. 이는 매일 아침 기억이 리셋되는 영화 '메멘토'나 '첫 키스만 50번째'의 주인공과 같습니다.
- **속기 쉬움(Gullibility):** 프롬프트 인젝션 공격에 매우 취약하여, 악의적인 지시에 쉽게 속아 사용자 데이터를 유출할 수 있습니다.

결론적으로 LLM은 '결함 있는 슈퍼맨'과 같습니다. 우리는 이들의 결함을 피하면서 초능력을 최대한 활용하는 방법을 배워야 합니다. 이것이 바로 소프트웨어 3.0 시대의 핵심 과제입니다.

4. 가장 큰 기회: '부분 자율성' 앱과 아이언맨 슈트

카파시는 현재 가장 큰 기회가 완전 자율 AI가 아닌, 인간을 보조하는 '부분 자율성(Partial Autonomy)'을 가진 제품에 있다고 주장합니다. 그는 화려한 완전 자율 에이전트 데모보다, 인간과 AI가 효율적으로 협력하는 '아이언맨 슈트' 같은 도구를 만들어야 한다고 역설합니다.

부분 자율성 앱의 4가지 공통 특성

그는 성공적인 부분 자율성 앱의 예로 코딩 보조 도구 'Cursor'와 AI 검색 엔진 'Perplexity'를 들며, 이들이 가진 4가지 공통적인 특성을 분석합니다.

1. **컨텍스트 관리(Context Management):** 사용자가 신경 쓰지 않도록, 앱이 알아서 필요한 파일, 정보, 대화 기록 등을 컨텍스트 창에 효율적으로 채워줍니다.
2. **LLM 호출 오케스트레이션(Orchestration):** 임베딩 모델, 채팅 모델, 코드 분석 모델 등 여러 종류의 LLM을 목적에 맞게 조율하고 호출하는 복잡한 과정을 자동화합니다.
3. **애플리케이션 특화 GUI(Application-specific GUI):** 텍스트로만 결과를 보여주는 대신, 코드 변경 사항을 시각적으로 보여주는 'diff' 뷰나, 정보의 출처를 표시하는 각주처럼, 인간이 AI의 작업을 빠르고 직관적으로 검증(audit)할 수 있는 맞춤형 UI/UX를 제공합니다.
4. **자율성 슬라이더(Autonomy Slider):** 사용자가 작업의 성격에 따라 AI에 위임할 자율성의 수준을 조절할 수 있게 합니다. 예를 들어 Cursor에서는 간단한 자동 완성(Tab)부터, 특정 코드 블록 수정(Cmd+K), 파일 전체 수정(Cmd+L), 나아가 프로젝트 전체를 분석하고 수정하는 에이전트 모드(Cmd+I)까지 다양한 수준의 자율성을 선택할 수 있습니다.

'생성-검증' 루프와 '고삐'의 중요성

부분 자율성 앱의 핵심은 AI가 '생성(Generate)'하고 인간이 '검증(Verify)'하는 협업 루프를 최대한 빠르게 만드는 것입니다. 이를 위해 카파시는 두 가지를 강조합니다.

- **검증 속도 높이기:** GUI는 인간 두뇌의 '컴퓨터 비전 GPU'를 활용하여 텍스트를 읽는 것보다 훨씬 빠르게 정보를 처리하게 해주므로, 검증 속도를 극적으로 높이는 데 필수적입니다.
- **AI에 고삐 채우기(Keeping agents on the leash):** AI가 한 번에 수천 줄의 코드를 변경하는 등 과도하게 반응하면, 인간인 내가 검토하는 데 병목이 생깁니다. 따라서 AI가 너무 앞서나가지 않도록 '고삐'를 채워, 작고 점진적인 제안을 하도록 유도하고 인간이 이를 빠르게 검토하며 나아가는 것이 훨씬 효율적입니다.

아이언맨 슈트 vs. 아이언맨 로봇

카파시는 자동 운전 기술 개발 경험을 상기시키며, 완전 자율성은 생각보다 훨씬 어렵고 긴 시간이 걸리는 문제라고 경고합니다. "2025년은 에이전트 원년"이라는 식의 성급한 기대를 경계하며, 앞으로는 '에이전트의 10년'이 될 것이라고 말합니다.

> *우리는 '아이언맨 로봇'(완전 자율 에이전트)이 아니라, '아이언맨 슈트'(인간 능력 증강 도구)를 만들어야 합니다.*

아이언맨 슈트는 토니 스타크의 능력을 증강시키는 동시에, 필요할 때는 스스로 날아서 주인을 찾아오는 에이전트의 역할도 합니다. 이것이 바로 '자율성 슬라이더'의 개념입니다. 지금은 신뢰도가 낮은 LLM을 고려할 때, 인간을 돕는 '슈트'를 만드는 데 집중하되, 점차 자율성 슬라이더를 오른쪽으로 옮겨갈 수 있는 제품을 설계해야 한다는 것이 그의 핵심 주장입니다.

5. 프로그래밍의 민주화: '바이브 코딩'의 시대

소프트웨어 3.0이 가져온 또 하나의 혁신은 프로그래밍의 장벽을 극적으로 낮췄다는 점입니다. 카파시는 이를 '바이브 코딩(Vibe Coding)'이라는 새로운 용어로 설명합니다.

바이브 코딩은 코딩에 대한 깊은 지식 없이도, LLM의 도움을 받아 "이런 느낌으로 만들어 줘"라고 지시하며 직관과 '바이브(vibe)'에 의존해 소프트웨어를 만드는 방식을 의미합니다. 카파시 자신도 Swift를 전혀 모르지만 LLM의 도움만으로 간단한 iOS 앱을 만들었고, 레스토랑 메뉴판 사진을 음식 이미지로 바꿔주는 'MenuGen'이라는 웹 앱을 개발한 경험을 공유합니다.

이 경험을 통해 그가 얻은 가장 큰 통찰은, 정작 '코딩' 자체는 프로젝트에서 가장 쉬운 부분이었다는 것입니다. 오히려 사용자 인증, 결제, 배포 등 '데브옵스(DevOps)' 작업이 훨씬 더 어렵고 시간이 많이 걸렸습니다. 이는 LLM이 아직 해결해주지 못하는, 브라우저에서 수많은 버튼을 클릭해야 하는 지루한 작업들이기 때문입니다.

> *"한 컴퓨터가 다른 컴퓨터(나)에게 무엇을 클릭해야 하는지 지시하고 있다. 왜 네가 직접 하지 않는가?"*

이러한 경험은 다음 주제인 '에이전트를 위한 개발'의 필요성으로 자연스럽게 이어집니다.

6. 다음 개척지: 인간이 아닌 '에이전트'를 위한 개발

카파시는 이제 디지털 정보의 소비자로 '인간(GUI)'과 '컴퓨터(API)' 외에 '에이전트(Agent)'라는 새로운 종이 등장했다고 말합니다. 따라서 우리는 이 새로운 소비자를 위해 디지털 인프라를 재설계해야 합니다.

에이전트 친화적 인프라 구축하기

인간을 위해 설계된 현재의 웹사이트와 문서는 LLM 에이전트가 이해하고 조작하기 매우 어렵습니다. 카파시는 이를 해결하기 위한 몇 가지 아이디어를 제시합니다.

- **llm.txt:** 웹 크롤러를 위한 `robots.txt`처럼, 웹사이트의 목적과 주요 기능, API 엔드포인트 등을 마크다운 형식으로 설명하는 `llm.txt` 파일을 제공할 수 있습니다.
- **기계가 읽기 쉬운 문서:** 복잡한 HTML 대신, LLM이 파싱하기 매우 쉬운 마크다운(Markdown) 형식으로 기술 문서를 제공하는 것입니다. Vercel, Stripe 등 일부 기업은 이미 이를 시작했습니다.
- **'클릭'을 '명령어'로:** 문서에 '이 버튼을 클릭하세요'라고 쓰는 대신, 에이전트가 직접 실행할 수 있는 `curl` 같은 커맨드 라인 명령어를 제공해야 합니다.
- **에이전트용 데이터 변환 도구:** GitHub URL을 약간만 변경하면 전체 리포지토리 코드를 하나의 텍스트 파일로 합쳐주는 `get ingest` 같은 도구들은 에이전트가 정보를 소비하는 것을 크게 돕습니다.

물론 미래에는 에이전트가 인간 중심의 웹을 스스로 탐색하는 능력이 향상될 것입니다. 하지만 우리가 먼저 에이전트에게 '다가가' 그들이 작업하기 쉬운 환경을 만들어주는 것이 훨씬 더 효율적이고 가치 있는 일이라고 카파시는 강조합니다. 이는 에이전트의 작업 비용과 실패율을 크게 낮출 수 있기 때문입니다.

7. 결론: 지금이 바로 새로운 시대로 뛰어들 때

카파시는 강연을 마무리하며 지금이 바로 이 업계에 뛰어들기에 가장 흥미로운 시기라고 다시 한번 강조합니다. 우리는 소프트웨어 3.0이라는 거대한 변혁의 초입에 서 있으며, 이는 기존의

수많은 소프트웨어를 다시 작성하고 새로운 종류의 애플리케이션을 창조할 엄청난 기회를 의미합니다.

그의 핵심 메시지를 요약하면 다음과 같습니다.

- LLM은 1960년대의 OS와 같은 초기 단계의 **새로운 운영체제**입니다.
- LLM은 초인적인 능력과 명백한 결함을 동시에 가진 **'결함 있는 슈퍼맨'**이므로, 이들의 특성을 이해하고 협력하는 법을 배워야 합니다.
- 가장 큰 기회는 완전 자율 로봇이 아닌, 인간의 능력을 증강시키는 **'부분 자율성'을 가진 '아이언맨 슈트'** 같은 제품에 있습니다.
- 이제 우리는 인간뿐만 아니라 **'에이전트'를 위한 인프라**를 구축해야 합니다.

소프트웨어의 패러다임이 바뀌고 있습니다. 자연어가 새로운 프로그래밍 언어가 되고, 모든 사람이 소프트웨어를 만들 수 있는 시대가 열리고 있습니다. 카파시의 통찰처럼, 이 거대한 변화의 물결에 올라타 미래를 함께 만들어갈 준비를 해야 할 때입니다.

본 내용은 Andrej Karpathy의 'Software in the era of AI' 발표와 관련 자료를 참고하여 작성되었습니다.