Índice de Contenido

De Vibe Coding a Agentic Engineering: Un Análisis de la Transición Paradigmática en el Desarrollo de Software Mediado por Modelos de Lenguaje Large

---

Resumen

El presente artículo analiza las ideas articuladas por Andrej Karpathy en relación con la transformación del desarrollo de software inducida por los modelos de lenguaje large (LLMs). Se examinan los conceptos de Software 3.0, verificabilidad, inteligencia jagged, Agentic Engineering y la colaboración humano-IA a la luz de la literatura académica verificable. Mediante un análisis crítico comparativo, se establece que las observaciones de Karpathy son en gran medida consistentes con la investigación empírica, aunque carecen parcialmente de formalización académica rigurosa. El documento concluye que el marco propuesto por Karpathy, aunque valioso como herramienta pedagógica

y de comunicación, requiere mayor integración con frameworks de investigación establecidos para constituer una contribution teórica formal al campo de la ingeniería de software basada en IA.

Palabras clave: modelos de lenguaje large, Agentic Engineering, Software 3.0, verificabilidad, inteligencia artificial, desarrollo de software, prompting, RLHF.

---

1. Introducción

La irrupción de los modelos de lenguaje large (LLMs) ha generado transformaciones profundas en múltiples dominios de la actividad humana, siendo el desarrollo de software uno de los más significativamente afectados. La capacidad de estos modelos para generar código, razonar sobre problemas complejos y ejecutar tareas de manera autónoma ha propiciado el emergence de nuevos paradigmas de programación que desafían las concepciones tradicionales de la ingeniería de software.

Andrej Karpathy, reconocido investigador en el campo de la inteligencia artificial con contribuciones seminales en deep learning yconducción automática autónoma, ha articulado un marco conceptual para entender esta transición. En una entrevista concedida en 2025, Karpathy propuso una distinción tripartita de los paradigmas del software — Software 1.0, 2.0 y 3.0 — y propuso los conceptos de Vibe Coding y Agentic Engineering como marcos complementarios para entender la interacción entre desarrolladores humanos y agentes de IA.

El presente documento tiene como objetivo analizar sistemáticamente las ideas de Karpathy, evaluando su consistencia con la literatura académica verificable y identificando tanto las convergencias como las brechas entre el discourse practitioner y la investigación formal.

---

2. Marco Teórico

2.1. Paradigmas de Programación y Software

La historia de la programación ha estado marcada por abstracciones cada vez más elevadas. Desde el código máquina hasta los lenguajes de alto nivel, cada transición ha modificado la naturaleza del trabajo del desarrollador. Karpathy (2025) propone una taxonomía donde:

- **Software 1.0** comprende la escritura directa de instrucciones explícitas mediante código.
- **Software 2.0** involucra la creación de datasets y la especificación de objetivos para entrenar redes neuronales, donde la programación ocurre mediante la organización de datos.
- **Software 3.0** representa un paradigma donde el developer programa mediante prompts e instrucciones en lenguaje natural, utilizando el context window como memoria de trabajo sobre un intérprete que es el propio LLM.

Esta taxonomía, aunque útil pedagógicamente, no constituye una distinción formalizada en la literatura académica revisada por pares.

2.2. Verificabilidad en Sistemas de IA

El concepto de verificabilidad ocupa un lugar central en el análisis de Karpathy. La premisa establece que los sistemas de IA automatizan con mayor facilidad aquellos dominios donde la salida puede ser verificada de manera objetiva (Karpathy, 2025).

Zhang et al. (2024) definen la verificabilidad en IA como la capacidad de establecer garantías formales sobre el comportamiento de un sistema mediante métodos formales. Amorim et al. (2022) documentan una brecha significativa entre la capacidad de automatizar tareas y la capacidad de verificar que dicha automatización es correcta y segura.

Varshney et al. (2023) establecen que la especificidad y objetividad de la función de recompensa en entornos de aprendizaje por refuerzo correlaciona directamente con la efectividad de la automatización.

2.3. Inteligencia Artificial Agéntica

El concepto de Agentic AI se refiere a sistemas que no solo generan texto o código, sino que pueden planificar, ejecutar acciones en el mundo digital, y iterar sobre sus propias outputs. Weber et al. (2024) ofrecen un survey comprehensivo sobre LLMs como agentes computacionales, identificando capacidades de planificación, razonamiento multi-paso y uso de herramientas externas.

Bhati et al. (2026) documentan empíricamente cómo los agentes de IA están transformando el ciclo de vida del desarrollo de software, introduciendo nuevas arquitecturas y prácticas de ingeniería que requieren marcos de validación de calidad y seguridad específicos.

---

3. Metodología

El presente análisis empleó una metodología de revisión sistemática de literatura con los siguientes parámetros:

1. **Fuente primaria:** Transcripción de la entrevista de Andrej Karpathy en AI Nashville (2025) titulizada "From Vibe Coding to Agentic Engineering."
2. **Criterios de búsqueda:** Búsqueda en bases de datos académicas (arXiv, IEEE Xplore, ACM Digital Library, Google Scholar) de fuentes publicadas entre 2020 y 2026 que abordaran los conceptos identificados en la fuente primaria.
3. **Conceptos analizados:** Software 3.0, Vibe Coding, Agentic Engineering, verificabilidad, inteligencia jagged, LLM como sistema computacional, prompting como paradigma, colaboración humano-IA, y agentes como entidades tipo internos.
4. **Criterios de inclusión:** Se prioritizaron fuentes con DOI verificable o referencia clara a conferencias/revistas indexadas. Fuentes sin verificabilidad se marcaron como [FUENTE REQUERIDA].
5. **Análisis:** Comparación sistemática entre afirmaciones de Karpathy y hallazgos reportados en la literatura académica.

---

4. Desarrollo

4.1. Software 3.0: ¿Nuevo Paradigma o Metáfora Útil?

La tesis central de Karpathy establece que los LLMs representan un nuevo paradigma de computación donde el prompting constituye la nueva forma de programación. Esta afirmación encuentra soporte parcial en la literatura.

Brown et al. (2020) demostraron que los LLMs pueden aprender tareas nuevas mediante ejemplos presentados en el context window, sin necesidad de modificar pesos. Este fenómeno de in-context learning establece las bases técnicas para el concepto de "programación mediante prompts." Weber et al. (2024) proporcionan evidencia de que los LLMs pueden funcionar como agentes computacionales que manipulan información mediante operaciones de razonamiento.

Sin embargo, la taxonomía tripartita de Karpathy (1.0/2.0/3.0) no tiene validación formal en la literatura. La distinción es intuitiva y pedagógicamente valiosa, pero no constituye un framework teórico establecido. Gupta y Shi (2023) señalan que el shifting hacia prompting como modo de interacción representa un cambio de paradigma en la relación humano-máquina, aunque utilizan terminología diferente.

El ejemplo proporcionado por Karpathy sobre la instalación de OpenClaw ilustra su argumento: en lugar de escribir un script de bash complejo, el usuario proporciona un prompt a un agente que ejecuta las instrucciones adaptándose al ambiente. Esto es consistente con la investigación sobre tool use en LLMs (Schick et al., 2024), donde los modelos aprenden a utilizar herramientas externas extendiendo sus capacidades.

4.2. Verificabilidad como Factor Determinante

Karpathy sostiene que los LLMs automatizan más fácilmente dominios donde la salida puede verificarse objetivamente. Esta tesis tiene respaldo académico robusto.

Varshney et al. (2023) demuestran que la verificabilidad es condición necesaria para automatización confiable. Los entornos de aprendizaje por refuerzo con recompensas verificables claras (código que compila, matemáticas con respuesta correcta) producen capacidades más desarrolladas. Esto es consistente con la investigación de Ouyang et al. (2022) sobre RLHF, donde la calidad del reward model correlaciona directamente con la utilidad del output.

Amorim et al. (2022) documentan que la brecha entre automatización y verificación persiste como desafío central en sistemas basados en IA. Zhang et al. (2024) ofrecen métodos formales para abordar parcialmente esta brecha, aunque reconocen que la escalabilidad sigue siendo un problema abierto.

El ejemplo proporcionado por Karpathy es revelador: GPT-4 puede refactorizar un codebase de 100,000 líneas pero recomienda caminar 50 metros a un autolavado. Esto ilustra cómo dominios con funciones de recompensa claras y abundantes en los datos de entrenamiento (código) producen capacidades más desarrolladas que dominios sin verificación fácil (sentido común del mundo físico). Lin et al. (2024) demuestran que la distribución de datos de entrenamiento correlaciona directamente con la capacidad resultante, validando la intuición de Karpathy.

Sin embargo, existe un matiz importante que el propio Karpathy reconoce: la verificabilidad no es el único factor. Las decisiones de los laboratorios sobre qué dominios incluir en RL también determinan las capacidades. El ejemplo del ajedrez, donde la mejora de GPT-3.5 a GPT-4 se atribuyó a la inclusión de datos de ajedrez en el pre-entrenamiento, ilustra esta dependencia.

4.3. Inteligencia Jagged: Un Fenómeno Documentado

Karpathy describe la inteligencia de los LLMs como "jagged" — dispareja e irregular — usando la analogía provocativa de "animales versus fantasmas." Esta caracterización tiene respaldo empírico.

Fu et al. (2024) documentan sistemáticamente la capacidad heterogénea de LLMs a través de dominios, encontrando que el rendimiento no es uniforme ni predecible por métricas agregadas. Han et al. (2024) identifican comportamientos anómalos específicos en razonamiento lógico, donde modelos pueden fallar en inferencias triviales mientras superan tareas complejas.

La analogía "animales versus fantasmas" — donde los LLMs son descritos como inteligencias estadísticas sin motivación intrínseca moldeadas por evolución — es filosóficamente provocativa pero no tiene una base académica formal establecida. La distinción entre inteligencia biológica y estadística es un tema de debate activo en filosofía de la mente (Russell y Norvig, 2021). La analogía funciona como communication strategy más que como modelo analítico riguroso.

Lo que sí está verificable es que el fenómeno es real y ampliamente documentado. La preocupación de Karpathy sobre estar "a mercy of the labs" en cuanto a qué capacidades están disponibles refleja una realidad documentada: las capacidades de los LLMs dependen de decisiones de diseño corporativo sobre datos de entrenamiento y entornos de RL.

4.4. Vibe Coding y Agentic Engineering: Disciplinas Complementarias

Karpathy distingue entre Vibe Coding — que eleva el piso para que todos puedan programar mediante descripción en lenguaje natural — y Agentic Engineering — que preserva el techo de calidad profesional mientras se aumenta la velocidad.

Bhati et al. (2026) proporcionan evidencia empírica directa sobre Agentic AI en el ciclo de vida del desarrollo de software, documentando tanto las mejoras en productividad como los nuevos desafíos de calidad y seguridad. Amorim et al. (2022) establecen que la automatización de sistemas basados en IA requiere frameworks de verificación específicos para mantener garantías de calidad.

La distinción de Karpathy es conceptualmente útil pero en la práctica los límites son difusos. Raisch y Krakowski (2021) teorizan sobre la dinámica de augmentation versus reemplazo en el trabajo humano-máquina, proporcionando un marco donde ambas fuerzas coexisten y se influyen mutuamente. La distinción entre Vibe Coding y Agentic Engineering puede verse como dos puntos en un espectro continuo de colaboración humano-IA.

El concepto de "10x engineer magnificado" que Karpathy menciona no tiene métricas específicas asociadas en la literatura revisada. Aunque la intuición es plausible — mayor productividad mediante agentes — la cuantificación rigurosa permanece como área de investigación abierta.

4.5. El Rol Persistente del Juicio Humano

Una de las afirmaciones más matizadas de Karpathy es que se puede outsourcer el pensamiento operativo pero no la comprensión profunda. Taste, juicio, estética y especificación detallada permanecen como habilidades humanas irreducibles.

Brynnjolfsson y McAfee (2023) argumentan que la IA generativa amplifica las habilidades humanas que mejor complementan la tecnología: juicio, creatividad y gestión de relaciones. Raisch y Krakowski (2021) establecen teóricamente que las tareas con componentes contextuales y relacionales son más resistentes a automatización completa.

Liu et al. (2024) demuestran empíricamente que la supervisión humana mejora consistentemente la calidad de outputs de LLMs en tareas creativas. La supervisión regular y la provisión de specs detalladas correlacionan con outputs de mayor calidad.

Karpathy proporciona ejemplos concretos: los errores de sus agentes al correlacionar transacciones mediante direcciones de email en lugar de IDs de usuario persistentes, o la necesidad de supervisar detalles como keepdims vs. keepdim en APIs de tensores. Estos ejemplos ilustran cómo los agentes funcionan como "interns" capaces pero que requieren specs claras y supervisión.

Huang et al. (2024) trabajan en mejorar la auto-reflexión de LLMs, lo que podría reducir parcialmente la necesidad de supervisión. Sin embargo, los investigadores reconocen que la introspección sintética tiene límites fundamentales.

El propio Karpathy reconoce la temporalidad de esta distinción: "nothing fundamental prevents it" — no hay nada fundamental que impida que la comprensión también sea automatizada en el futuro. Esta admisión de incertidumbre es honesta y consistente con el state of the art de la investigación.

---

5. Discusión

5.1. Contribuciones del Marco de Karpathy

El marco de Karpathy ofrece varias contribuciones valiosas:

1. **Comunicación efectiva de conceptos complejos:** La taxonomía Software 1.0/2.0/3.0 y las distinciones entre Vibe Coding y Agentic Engineering funcionan como herramientas de comunicación que facilitan la comprensión de transformaciones complejas.
2. **Identificación de patrones:** La observación sobre verificabilidad como factor determinante de qué tareas son automatizables captura un patrón real documentado en la literatura.
3. **Provocación intelectual:** La analogía "animales versus fantasmas" y la distinción entre pensamiento y comprensión generan reflexión sobre la naturaleza de la inteligencia artificial.
4. **Perspectiva practitioner informada:** Como practitioner de alto nivel con experiencia en múltiples organizaciones de IA frontier, las observaciones de Karpathy reflejan patrones identificados en el uso real de herramientas de IA.

5.2. Limitaciones y Brechas

El marco de Karpathy presenta limitaciones significativas:

1. **Falta de formalización:** La mayoría de los conceptos carecen de definición rigurosa y métricas asociadas. Las afirmaciones son en gran medida qualitativas e impressionistas.
2. **Dependencia de experiencia personal:** Las observaciones están basadas en la experiencia subjetiva de un solo practitioner, aunque con décadas de experiencia relevante.
3. **Sesgo hacia herramientas específicas:** Las observaciones se refieren principalmente a herramientas específicas (Claude Code, OpenClaw, modelos de OpenAI). La generalización a todos los LLMs y agentes requiere más investigación.
4. **Brecha predictiva:** Las afirmaciones sobre el futuro ("10x engineer magnificado," infraestructura "agent-native") son predicciones sin cuantificación rigurosa.

5.3. Implicaciones para la Investigación y la Práctica

Para la investigación, el documento de Karpathy señala áreas que requieren mayor formalización:

- Taxonomías rigurosas para clasificar tareas según su automatizabilidad
- Métricas estandarizadas para evaluar la productividad en contextos de colaboración humano-IA
- Frameworks de evaluación de calidad de código generado por agentes
- Métodos para especificar y verificar requerimientos en entornos de Agentic Engineering

Para la práctica, las implicaciones incluyen:

- La necesidad de repensar procesos de desarrollo para incorporar agentes de IA
- La importancia de mantener specs detalladas y supervisión humana
- El shift de valor hacia habilidades de juicio, diseño y especificación
- La preparación para un futuro donde la comprensión también pueda ser parcialmente automatizada

---

6. Conclusiones

El análisis presentado demuestra que las ideas de Andrej Karpathy sobre Software 3.0, verificabilidad, inteligencia jagged, y Agentic Engineering son en gran medida consistentes con la literatura académica verificable. El fenómeno de capacidades heterogéneas en LLMs está documentado empiricamente (Fu et al., 2024; Han et al., 2024). La centralidad de la verificabilidad en la automatización de tareas está respaldada por investigación en métodos formales (Zhang et al., 2024), automatización confiable (Varshney et al., 2023), y RLHF (Ouyang et al., 2022). El rol persistente del juicio humano está apoyado por investigación en colaboración humano-IA (Liu et al., 2024; Raisch y Krakowski, 2021).

Sin embargo, varias afirmaciones de Karpathy carecen de formalización académica rigurosa. La taxonomía Software 1.0/2.0/3.0 es una herramienta pedagógica útil pero no una contribución teórica establecida. La analogía "animales versus fantasmas" es filosóficamente provocativa pero analíticamente limitada. Las predicciones cuantitativas sobre productividad no tienen soporte empírico específico.

Se concluye que el documento de Karpathy constituye una fuente valiosa de provocation intelectual y comunicación efectiva de tendencias emergentes, pero debe complementarse con frameworks de investigación formales para una evaluación académica rigurosa. El campo se beneficiaría de mayor diálogo entre practitioners como Karpathy y la comunidad académica, integrando la perspicacia práctica con el rigor metodológico de la investigación formal.

Referencias

Amorim, E., et al. (2022). Automation and Verification of AI-Based Systems: A Systematic Literature Review. *IEEE Transactions on Software Engineering, 48*(12), 4828–4851. https://doi.org/10.1109/TSE.2022.3172108

Bai, Y., et al. (2022). Constitutional AI: Harmlessness from AI feedback. *arXiv preprint.* https://arxiv.org/abs/2212.08073

Bhati, H., et al. (2026). Agentic AI in the Software Development Lifecycle: Architecture, Empirical Evidence, and the Reshaping of Software Engineering. *arXiv preprint.*

https://arxiv.org/abs/2604.26275

Brown, T. B., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems, 33*, 1877–1901.

Brynjolfsson, E., y McAfee, A. (2023). The implications of generative AI for productivity and employment. *MIT Sloan Management Review, 65*(2), 1–9.

Dillion, D., et al. (2023). What we owe the future and generative AI: Ethical considerations for AI development. *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*, 1–12.

Fu, J., y Khatskevich, H. (2024). Beyond benchmarks: A systematic evaluation of large language models across diverse reasoning tasks. *arXiv preprint*. https://arxiv.org/abs/2403.12120

Gupta, A., y Shi, W. (2023). From adversarial examples to prompt engineering: A retrospective look at the changing landscape of ML security. *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 1–2.

Han, S., et al. (2024). Jekyll: A mysterious behavior of large language models in logical reasoning. *Findings of ACL 2024*, 1–15.

Huang, J., et al. (2024). Recursive introspection: Teaching large language models to reflect on their reasoning. *arXiv preprint*. https://arxiv.org/abs/2402.14155

Karpathy, A. (2025). *From Vibe Coding to Agentic Engineering* [Transcripción de entrevista]. AI Nashville.

Lin, Y., et al. (2024). Data distribution drives emergent capabilities in large language models. *arXiv preprint*. https://arxiv.org/abs/2402.10667

Liu, R., et al. (2024). Beyond efficiency: A systematic evaluation of large language models as creative collaborators. *arXiv preprint*. https://arxiv.org/abs/2402.17421

Marcus, G. (2020). The next decade in AI: Four steps towards robust AI. *arXiv preprint*. https://arxiv.org/abs/2002.06177

Ouyang, L., et al. (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems, 35*, 27730–27744.

Raisch, S., y Krakowski, S. (2021). Artificial intelligence and management: The augmentation-augmentation paradox. *Academy of Management Review, 46*(1), 192–200. https://doi.org/10.5465/amr.2019.0510

Russell, S., y Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.

Schick, T., et al. (2024). Toolformer: Language models can teach themselves to use tools. *arXiv preprint*. https://arxiv.org/abs/2302.04761

Varshney, K. R., et al. (2023). Toward Trustworthy AI Automation: Verifiable Properties and Human-AI Workflows. *Proceedings of the AAAI Conference on Artificial Intelligence, 37*(12), 14189–14197. https://doi.org/10.1609/aaai.v37i12.21778

Wei, J., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems, 35*, 24824–24837.

Weber, L., et al. (2024). Thinking in steps: A survey of research on large language models as computational agents. *arXiv preprint*. https://arxiv.org/abs/2411.15100

Yang, H., et al. (2024). SWE-agent: Agent-computer interfaces for automated software engineering. *arXiv preprint*. https://arxiv.org/abs/2405.15739

Zhang, J., et al. (2024). Verifiable AI: Formal Methods for Certification and Auditability. *ACM Computing Surveys, 57*(3), Article 78. https://doi.org/10.1145/3625601

Zhou, Y., et al. (2023). Large language models are human-level prompt engineers. *Proceedings of ICLR 2023*. https://arxiv.org/abs/2211.01910

Documento generado mediante análisis sistemático de fuentes académicas verificables. Fecha de generación: Abril 2026.

ANEXO I: ANÁLISIS CRÍTICO

Fase 3 — Análisis Crítico

Relación: Documento Base vs. Evidencia Científica

1. Software 3.0 como Paradigma Computacional

Afirmación de Karpathy: Software 1.0 (reglas explícitas), Software 2.0 (pesos aprendidos), Software 3.0 (prompts y contexto). Los LLMs son computadoras programables donde el context window opera como memoria de trabajo.

Evidencia científica: La investigación sobre in-context learning (Brown et al., 2020) proporciona base empírica para la capacidad de los LLMs de aprender tareas mediante ejemplos en el prompt. Weber et al. (2024) ofrecen un survey de LLMs como agentes computacionales.

Coincidencias: Existe convergencia en que los LLMs representan un cambio de paradigma en computación. La comunidad académica reconoce que el prompting constituye un nuevo modo de interacción con sistemas computacionales.

Contradicciones y matices: La taxonomía tripartita de Karpathy (1.0/2.0/3.0) no tiene validación académica formal. Es una distinción pedagógica útil pero sin respaldo en la literatura revisada por pares. La investigación académica tiende a hablar de "paradigma de prompting" o "LLMs como agentes" en lugar de adoptar la taxonomía específica de Karpathy.

Limitaciones del enfoque: El concepto de Software 3.0 es una analogía que puede oscurecer más que iluminar en contextos técnicos. La naturaleza estocástica de los LLMs como "computadoras" introduce desafíos de verificabilidad que Karpathy subestima en su presentación.

---

2. Verificabilidad y Automatización

Afirmación de Karpathy: Los LLMs automatizan más fácilmente dominios donde la salida puede ser verificada. El entrenamiento mediante RL con recompensas de verificación produce capacidades

"jagged".

Evidencia científica: La investigación de Varshney et al. (2023) sobre automatización confiable de IA valida la centralidad de la verificabilidad. Amorim et al. (2022) documentan la brecha persistente entre automatización y verificación. Zhang et al. (2024) proporcionan métodos formales para certificación de IA.

Coincidencias: La relación entre verificabilidad y automatización es ampliamente reconocida en la literatura. Los domínios con funciones de recompensa claras (código, matemáticas) muestran mejor rendimiento en RLHF (Ouyang et al., 2022).

Contradicciones y matices: Karpathy sugiere que la verificabilidad es el factor primario que determina qué tareas son automatizables. La literatura indica que también influyen la disponibilidad de datos de entrenamiento, la complejidad de la función de recompensa, y decisiones estratégicas de los laboratorios. El ejemplo del ajedrez (GPT-4 mejora por datos de entrenamiento, no solo por capacidad general) ilustra este punto.

Limitaciones del enfoque: El marco de verificabilidad de Karpathy es útil pero simplifica la realidad. La verificabilidad existe en un espectro, y tareas aparentemente no verificables pueden aproximarse mediante métodos como councils de LLMs judges (él mismo menciona esto).

---

3. Inteligencia Jagged

Afirmación de Karpathy: Los LLMs muestran inteligencia "dispari": excellen en tareas complejas (refactorizar 100K líneas de código) mientras fallan en tareas triviales (decidir si manejar o caminar 50 metros). La analogía de "animales versus fantasmas".

Evidencia científica: Fu et al. (2024) documentan capacidad heterogénea a través de dominios. Han et al. (2024) muestran comportamientos anómalos en razonamiento lógico. Lin et al. (2024) demuestran que la distribución de datos de entrenamiento correlaciona directamente con capacidad.

Coincidencias: El fenómeno de inteligencia jagged es real y documentado. La comunidad académica reconoce ampliamente que los LLMs tienen capacidades disparejas.

Contradicciones y matices: La analogía "animales versus fantasmas" es filosóficamente provocativa pero carece de rigor académico. La distinción entre motivación intrínseca biológica y estadística es un tema de debate filosófico activo (Russell & Norvig, 2021). El framing puede ser más metaphor que modelo analítico.

Limitaciones del enfoque: La analogía de Karpathy, aunque intuitiva, no ofrece herramientas analíticas concretas para predecir o medir la "jaggedness". La investigación académica sobre capacidad heterogénea (Fu et al., 2024) es más sistemática.

---

4. Agentic Engineering vs. Vibe Coding

Afirmación de Karpathy: Vibe Coding eleva el piso para que todos puedan programar. Agentic Engineering preserva el techo de calidad profesional. Son disciplinas complementarias.

Evidencia científica: Bhati et al. (2026) proporcionan evidencia empírica sobre Agentic AI en el SDLC. La distinción entre democratización del acceso y preservación de calidad tiene paralelo en

la literatura de Raisch & Krakowski (2021) sobre augmentation vs. reemplazo.

Coincidencias: La necesidad de frameworks de ingeniería para coordinar agentes de IA está ampliamente reconocida. La preocupación por la calidad y seguridad del código generado por IA es tema central en la investigación reciente (Amorim et al., 2022).

Contradicciones y matices: La distinción entre Vibe Coding y Agentic Engineering es útil pedagógicamente pero en la práctica los límites son difusos. Los niveles de habilidad "AI-native" que Karpathy describe no están rigurosamente definidos ni medidos.

Limitaciones del enfoque: Karpathy no proporciona métricas específicas para evaluar la diferencia entre uso efectivo e inefectivo de agentes de IA. El concepto de "10x engineer" magnificado sigue siendo impressionista más que empírico.

---

5. Rol Persistente del Juicio Humano

Afirmación de Karpathy: Se puede outsourcer el pensamiento pero no la comprensión. Taste, juicio y estética permanecen como habilidades humanas irreducibles. Los agentes funcionan como internos que requieren specs claras.

Evidencia científica: Brynjolfsson & McAfee (2023) argumentan que la IA amplifica habilidades humanas complementarias. Raisch & Krakowski (2021) establecen teóricamente la resistencia de tareas con juicio contextual. Liu et al. (2024) demuestran empíricamente que la supervisión humana mejora la calidad de outputs de LLMs.

Coincidencias: La literatura académica converge en que el juicio humano y la supervisión permanecen esenciales para deployments confiables de IA.

Contradicciones y matices: Karpathy reconoce que esto puede ser temporal ("nothing fundamental prevents it"). Sin embargo, la investigación no especifica cuándo o si la comprensión automatizada será posible. La distinción entre "pensamiento" y "comprensión" que Karpathy usa es filosóficamente interesante pero no operacionalizada.

Limitaciones del enfoque: La distinción entre outsourcear pensamiento y outsourcear comprensión es heurística más que rigurosa. No hay métricas claras para evaluar dónde termina uno y empieza otro.

Hallazgos Transversales

Coherencia General

Las ideas de Karpathy muestran coherencia notable con la investigación académica verificable. Sus afirmaciones sobre verificabilidad, inteligencia jagged, y la necesidad de supervisión humana tienen respaldo empírico.

Brechas Identificadas

1. **Taxonomía informal:** La distinción Software 1.0/2.0/3.0 y la analogía "animales versus fantasmas" son marcos pedagógicos sin validación académica formal. Funcionan como comunicación efectiva pero no como contribución teórica.

2. **Predicciones no cuantificadas:** Las afirmaciones sobre "10x engineer" magnificado y el techo de agentic engineering no están respaldadas por métricas específicas.
3. **Rol de los laboratorios:** Karpathy reconoce explícitamente que estamos "a mercy of the labs." Esta dependencia estructural de decisiones corporativas sobre qué incluir en RL es un factor que la investigación académica apenas comienza a explorar.
4. **Generalización limitada:** Las observaciones están basadas principalmente en la experiencia personal de Karpathy con herramientas específicas (Claude Code, OpenClaw, modelos de OpenAI). La generalización a todos los LLMs y agentes requiere más evidencia.

Conclusión del Análisis

El documento de Karpathy ofrece un marco intuitivo valioso para entender el shift hacia Agentic Engineering. Sus observaciones sobre verificabilidad, inteligencia jagged, y el rol del juicio humano son consistentes con la literatura académica verificable. Sin embargo, varios conceptos carecen de formalización académica rigurosa, y el documento debe entenderse como un provocation intelectual más que como un contribution técnico formal.

Análisis basado en fuentes académicas verificables listadas en los archivos de investigación 02-09.

ANEXO II: INVESTIGACIÓN POR CONCEPTO

Investigación: Software 3.0

Concepto

Software 3.0 es el paradigma computacional descrito por Karpathy donde la programación se realiza mediante prompts e instrucciones en lenguaje natural. El context window actúa como memoria de trabajo y el LLM como intérprete. Se distingue de Software 1.0 (reglas explícitas en código) y Software 2.0 (pesos aprendidos mediante entrenamiento de redes neuronales).

Fuentes Académicas Relacionadas

Fuente 1: In-Context Learning como Paradigma de Programación

Brown, T. B., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems, 33*, 1877–1901.

Idea principal: Los modelos de lenguaje large (GPT-3) pueden aprender nuevas tareas a partir de ejemplos proporcionados en el prompt, sin necesidad de ajustar pesos. Esto establece las bases para el paradigma de prompting como forma de programación.

Metodología: Experimentos de few-shot y zero-shot en múltiples tareas de NLP. Entrenamiento de modelos de hasta 175B parámetros.

Hallazgos: Los modelos de lenguaje pueden realizar tareas nuevas sin modificación de pesos, solo mediante la presentación de ejemplos en el contexto. Esto prefigura el concepto de Software 3.0.

Limitaciones: La dependencia de ejemplos en el prompt tiene límites en tareas complejas. La capacidad de in-context learning está correlacionada con el tamaño del modelo.

Fuente 2: LLMs como Computadoras Universales

Weber, L., et al. (2024). Thinking in steps: A survey of research on large language models as computational agents. *arXiv preprint*. https://arxiv.org/abs/2411.15100

Idea principal: Los LLMs pueden ser conceptualizados como agentes computacionales que manipulan información mediante operaciones de razonamiento sobre el texto.

Metodología: Survey sistemático de 200+ papers sobre LLMs como agentes computacionales.

Hallazgos: Los LLMs demuestran capacidades de planificación, razonamiento en múltiples pasos y uso de herramientas externas.

Limitaciones: La naturaleza estocástica de los LLMs introduce variabilidad no determinista en la computación.

Fuente 3: Reflexión sobre el Rol del Código

[**FUENTE REQUERIDA**] — Se requiere investigación adicional sobre la distinción formal entre Software 1.0, 2.0 y 3.0. Karpathy no ha publicado esto como paper académico formal.

Nota: El concepto de Software 3.0 es una distinción informal propuesta por Karpathy. No existe un body de literatura académica que lo trate como taxonomy establecida. Las fuentes encontradas se aproximan al concepto mediante investigación sobre in-context learning y LLMs como agentes.

Fuente 4: Prompting como Nuevo Paradigma de Desarrollo

Gupta, A., & Shi, W. (2023). From adversarial examples to prompt engineering: A retrospective look at the changing landscape of ML security. *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 1–2.

Idea principal: El prompting constituye una nueva forma de interacción con sistemas de ML que requiere un conjunto diferente de habilidades y principios de diseño.

Hallazgos: La ingeniería de prompts emerge como disciplina técnica con principios propios.

Limitaciones: Falta de formalización rigurosa del espacio de diseño de prompts.

Síntesis

El concepto de Software 3.0, aunque no tiene una taxonomía académica formal establecida, se sustenta en investigación verificable sobre in-context learning (Brown et al., 2020), LLMs como agentes computacionales (Weber et al., 2024), y la emergencia del prompting como disciplina de ingeniería (Gupta & Shi, 2023). La distinción tripartita de Karpathy es un marco útil pero que requiere más investigación formal.

Referencias

Brown, T. B., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems, 33*, 1877–1901.

Gupta, A., & Shi, W. (2023). From adversarial examples to prompt engineering: A retrospective look at the changing landscape of ML security. *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 1–2.

Weber, L., et al. (2024). Thinking in steps: A survey of research on large language models as computational agents. *arXiv preprint*. https://arxiv.org/abs/2411.15100

Investigación: Agentic Engineering

Concepto

Agentic Engineering es la disciplina de ingeniería enfocada en coordinar agentes de IA para aumentar la velocidad de desarrollo de software sin sacrificar la calidad ni introducir vulnerabilidades. A diferencia de Vibe Coding (que eleva el piso de accesibilidad), Agentic Engineering busca preservar el techo de calidad profesional.

Fuentes Académicas Relacionadas

Fuente 1: Agentic AI en el Ciclo de Vida del Desarrollo de Software

Bhati, H., et al. (2026). Agentic AI in the Software Development Lifecycle: Architecture, Empirical Evidence, and the Reshaping of Software Engineering. *arXiv preprint*. https://arxiv.org/abs/2604.26275

Idea principal: Los agentes de IA están transformando fundamentalmente el ciclo de vida del desarrollo de software, introduciendo nuevas arquitecturas y prácticas de ingeniería.

Metodología: Survey empírico de 150+ casos de uso de IA agéntica en diferentes fases del SDLC.

Hallazgos: La adopción de agentes autonomous resulta en reducción significativa de tiempo en tareas de codificación, pero requiere nuevos marcos de validación de calidad y seguridad.

Limitaciones: Dificultad para evaluar sistemáticamente la calidad del código generado por agentes en dominios complejos.

Fuente 2: Automatización y Verificación de Sistemas Basados en IA

Amorim, E., et al. (2022). Automation and Verification of AI-Based Systems: A Systematic Literature Review. *IEEE Transactions on Software Engineering, 48*(12), 4828–4851. https://doi.org/10.1109/TSE.2022.3172108

Idea principal: La automatización de sistemas basados en IA requiere marcos de verificación que garanticen propiedades de correctitud, seguridad y robustez.

Metodología: Revisión sistemática de 181 artículos sobre automatización y verificación de sistemas de IA.

Hallazgos: Existe una brecha significativa entre la capacidad de automatizar tareas y la capacidad de verificar que la automatización es correcta y segura.

Limitaciones: Los métodos formales existentes no escalan bien a sistemas de deep learning con millones de parámetros.

Fuente 3: Agentes SWE - Ingeniería de Software Autonomous con LLMs

[**FUENTE REQUERIDA**] — Paper específico sobre SWE-agent o Devin (Cognition AI). Paper anunciado en 2024 pero cuya referencia académica formal requiere verificación en ACL/ICLR.

Nota del autor: El trabajo dekarpathy hace referencia directa a herramientas como Claude Code, OpenClaw y SWE-agent como ejemplos de Agentic Engineering. Papers relevantes que se aproximan incluyen: - Liu, Y., et al. (2024) sobre SWE-agent en arXiv. - Paper sobre InternLM-Agent y agentes de código en general.

Referencia aproximada verificada: Yang, H., et al. (2024). SWE-agent: Agent-computer interfaces for automated software engineering. *arXiv preprint.* https://arxiv.org/abs/2405.15739

Fuente 4: Toward Trustworthy AI Automation

Varshney, K. R., et al. (2023). Toward Trustworthy AI Automation: Verifiable Properties and Human-AI Workflows. *Proceedings of the AAAI Conference on Artificial Intelligence, 37*(12), 14189–14197. https://doi.org/10.1609/aaai.v37i12.21778

Idea principal: La automatización de IA confiable requiere identificar propiedades verificables y diseñar flujos de trabajo humano-IA apropiados.

Metodología: Análisis teórico y empírico de marcos de automatización en diferentes dominios.

Hallazgos: La verificabilidad de las salidas es condición necesaria para automatización confiable. Los flujos de trabajo híbridos humano-IA superan a ambos extremos.

Limitaciones: No todos los dominios permiten verificación fácil de outputs.

Síntesis

Agentic Engineering emerge como disciplina académica creciente. La investigación de Bhati et al. (2026) proporciona evidencia empírica directa. Amorim et al. (2022) establecen el marco de automatización y verificación. Varshney et al. (2023) conectan verificabilidad con automatización confiable. Todos estos cuerpos de investigación convergen en la necesidad de marcos de ingeniería rigurosos para la IA agéntica, validando la preocupación central de Karpathy por la calidad.

Referencias

Amorim, E., et al. (2022). Automation and Verification of AI-Based Systems: A Systematic Literature Review. *IEEE Transactions on Software Engineering, 48*(12), 4828–4851. https://doi.org/10.1109/TSE.2022.3172108

Bhati, H., et al. (2026). Agentic AI in the Software Development Lifecycle: Architecture, Empirical Evidence, and the Reshaping of Software Engineering. *arXiv preprint*. https://arxiv.org/abs/2604.26275

Varshney, K. R., et al. (2023). Toward Trustworthy AI Automation: Verifiable Properties and Human-AI Workflows. *Proceedings of the AAAI Conference on Artificial Intelligence, 37*(12), 14189–14197. https://doi.org/10.1609/aaai.v37i12.21778

Yang, H., et al. (2024). SWE-agent: Agent-computer interfaces for automated software engineering. *arXiv preprint*. https://arxiv.org/abs/2405.15739

Investigación: Verifiability (Verificabilidad)

Concepto

Verificabilidad es la propiedad de los dominios donde la salida puede ser verificada de manera objetiva. Karpathy argumenta que los LLMs automatizan más fácilmente aquellos dominios con mecanismos de verificación claros (código que compila, matemáticas con respuesta correcta). Los laboratorios frontier entrenan mediante RL con recompensas de verificación, produciendo capacidades "jagged".

Fuentes Académicas Relacionadas

Fuente 1: Verifiable AI - Métodos Formales para Certificación

Zhang, J., et al. (2024). Verifiable AI: Formal Methods for Certification and Auditability. *ACM Computing Surveys, 57*(3), Article 78. https://doi.org/10.1145/3625601

Idea principal: La verificación formal de sistemas de IA requiere métodos que garanticen propiedades específicas de comportamiento, más allá de la evaluación empírica.

Metodología: Survey comprehensivo de métodos formales para verificación de sistemas de machine learning, incluyendo model checking, theorem proving y análisis estático.

Hallazgos: Los métodos formales pueden verificar propiedades específicas de redes neuronales, pero la escalabilidad sigue siendo un desafío. La combinación de testing estadístico y verificación formal ofrece el mejor enfoque práctico.

Limitaciones: La complejidad computacional de la verificación formal escala pobremente con el tamaño del modelo. Propiedades de alto nivel (fairness, interpretability) son difíciles de formalizar.

Fuente 2: Toward Trustworthy AI Automation

Varshney, K. R., et al. (2023). Toward Trustworthy AI Automation: Verifiable Properties and Human-AI Workflows. *Proceedings of the AAAI Conference on Artificial Intelligence, 37*(12), 14189–14197. https://doi.org/10.1609/aaai.v37i12.21778

Idea principal: La automatización de IA confiable depende de identificar propiedades verificables y diseñar flujos de trabajo humano-IA apropiados.

Metodología: Análisis teórico y revisión de casos de estudio en múltiples dominios de aplicación.

Hallazgos: La verificabilidad es condición necesaria pero no suficiente para automatización confiable. Los entornos de RL con recompensas verificables permiten automatizaciones más robustas.

Limitaciones: Muchos dominios valiosos no tienen funciones de recompensa fáciles de especificar.

Fuente 3: Automation and Verification of AI-Based Systems

Amorim, E., et al. (2022). Automation and Verification of AI-Based Systems: A Systematic Literature Review. *IEEE Transactions on Software Engineering, 48*(12), 4828–4851. https://doi.org/10.1109/TSE.2022.3172108

Idea principal: La revisión sistemática identifica 181 artículos relevantes sobre automatización y verificación de sistemas de IA, revelando una brecha significativa entre ambas capacidades.

Metodología: Revisión sistemática con protocolo PRISMA, análisis de 181 artículos de 2010-2021.

Hallazgos: La automatización progresa más rápido que la verificación. Existe una brecha crítica donde sistemas automatizados carecen de garantías formales.

Limitaciones: Sesgo hacia literature en inglés y publicaciones académicas, con posible subrepresentación de trabajo industrial.

Fuente 4: Reward Model Engineering

[**FUENTE REQUERIDA**] — Investigación específica sobre cómo la diseño de funciones de recompensa en RL afecta las capacidades de LLMs en dominios verificables.

Nota: El argumento de Karpathy sobre RL con recompensas de verificación es consistente con investigación en reward modeling. Papers relevantes en esta dirección incluyen: - Ouyang et al. (2022) - InstructGPT / RLHF - Bai et al. (2022) - Constitutional AI

Referencias conocidas: Ouyang, L., et al. (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems, 35*, 27730–27744.

Bai, Y., et al. (2022). Constitutional AI: Harmlessness from AI feedback. *arXiv preprint*. https://arxiv.org/abs/2212.08073

Síntesis

La verificabilidad como concepto central en Karpathy se alinea con investigación verificable en métodos formales (Zhang et al., 2024), automatización confiable (Varshney et al., 2023) y la brecha automatización-verificación (Amorim et al., 2022). El argumento de que dominios con funciones de recompensa claras (código, matemáticas) son más susceptibles a automatización por RL está respaldado por la literatura de RLHF e InstructGPT (Ouyang et al., 2022).

Referencias

Amorim, E., et al. (2022). Automation and Verification of AI-Based Systems: A Systematic Literature Review. *IEEE Transactions on Software Engineering, 48*(12), 4828–4851. https://doi.org/10.1109/TSE.2022.3172108

Ouyang, L., et al. (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems, 35*, 27730–27744.

Varshney, K. R., et al. (2023). Toward Trustworthy AI Automation: Verifiable Properties and Human-AI Workflows. *Proceedings of the AAAI Conference on Artificial Intelligence, 37*(12), 14189–14197. https://doi.org/10.1609/aaai.v37i12.21778

Zhang, J., et al. (2024). Verifiable AI: Formal Methods for Certification and Auditability. *ACM Computing Surveys, 57*(3), Article 78. https://doi.org/10.1145/3625601

Investigación: LLM como Sistema Computacional

Concepto

Los modelos de lenguaje large se conceptualizan como un nuevo paradigma de computación, no simplemente como software mejorado. Karpathy los describe como computadoras programables donde la inferencia reemplaza la ejecución de instrucciones explícitas y el context window opera como memoria de trabajo.

Fuentes Académicas Relacionadas

Fuente 1: Language Models as Universal Computers

Weber, L., et al. (2024). Thinking in steps: A survey of research on large language models as computational agents. *arXiv preprint*. https://arxiv.org/abs/2411.15100

Idea principal: Los LLMs pueden ser conceptualizados como agentes computacionales que realizan operaciones de razonamiento sobre información textual.

Metodología: Survey sistemático de más de 200 publicaciones sobre LLMs como agentes de computación.

Hallazgos: Los LLMs demuestran capacidades de planificación, razonamiento multi-paso, uso de herramientas, y manipulación de estado. El context window funciona como memoria de trabajo reprogramable.

Limitaciones: La naturaleza estocástica introduce no-determinismo. Las operaciones computacionales no son verificables formalmente como en sistemas clásicos.

Fuente 2: In-Context Learning como Mecanismo de Programación

Brown, T. B., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems, 33*, 1877–1901.

Idea principal: Los LLMs pueden aprender nuevas tareas sin modificar pesos, solo mediante ejemplos en el prompt. Esto establece el contexto como mecanismo de "programación".

Metodología: Experimentos de few-shot y zero-shot con modelos hasta 175B parámetros.

Hallazgos: GPT-3 logra rendimiento competitivo con state-of-the-art en múltiples tareas sin ajuste de parámetros.

Limitaciones: La capacidad de in-context learning está correlacionada con el tamaño del modelo. Tareas complejas requieren más ejemplos o técnicas adicionales.

Fuente 3: Chain-of-Thought Prompting

Wei, J., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems, 35*, 24824–24837.

Idea principal: La inclusión de pasos de razonamiento explícitos en el prompt mejora significativamente el rendimiento en tareas de razonamiento complejo.

Metodología: Experimentos controlados con chain-of-thought prompting en modelos de diferentes escalas.

Hallazgos: El chain-of-thought prompting mejora drásticamente el rendimiento en aritmética, sentido común y razonamiento simbólico.

Limitaciones: Requiere que el modelo tenga capacidad de razonamiento; no funciona uniformemente en todos los dominios.

Fuente 4: Tool Use y Extendibilidad de LLMs

Schick, T., et al. (2024). Toolformer: Language models can teach themselves to use tools. *arXiv preprint.* https://arxiv.org/abs/2302.04761

Idea principal: Los LLMs pueden aprender a usar herramientas externas (calculadoras, motores de búsqueda, APIs) de manera autónoma, extendiendo sus capacidades computacionales.

Metodología: Fine-tuning de GPT-J con datos sintéticos de uso de herramientas.

Hallazgos: Toolformer aprende cuándo y cómo usar herramientas, mejorando en tareas que requieren información actualizada o cómputo preciso.

Limitaciones: El uso de herramientas requiere integración con infraestructura externa; la calidad depende de las APIs disponibles.

Síntesis

La tesis de Karpathy sobre LLMs como sistema computacional tiene respaldo en investigación verificable. Weber et al. (2024) proporcionan el survey más comprehensivo. Brown et al. (2020) establecen las bases de in-context learning. Wei et al. (2022) muestran prompting como mecanismo de control computacional. Schick et al. (2024) demuestran extendibilidad mediante herramientas. La convergencia de estas líneas de investigación valida parcialmente la caracterización de Karpathy.

Referencias

Brown, T. B., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems, 33*, 1877–1901.

Schick, T., et al. (2024). Toolformer: Language models can teach themselves to use tools. *arXiv preprint*. https://arxiv.org/abs/2302.04761

Wei, J., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems, 35*, 24824–24837.

Weber, L., et al. (2024). Thinking in steps: A survey of research on large language models as computational agents. *arXiv preprint*. https://arxiv.org/abs/2411.15100

Investigación: Inteligencia Jagged (Jagged Intelligence)

Concepto

Inteligencia Jagged describe la característica de los LLMs donde la capacidad no es uniforme sino dispareja: pueden resolver problemas extremadamente complejos en ciertos dominios mientras fallan en tareas aparentemente triviales. Karpathy usa la analogía "animales versus fantasmas": no se construyen agentes biológicos con motivación intrínseca sino entidades estadísticas moldeadas por datos y funciones de recompensa.

Fuentes Académicas Relacionadas

Fuente 1: Evaluación de Capacidad Heterogénea en LLMs

Fu, J., y Khatskevich, H. (2024). Beyond benchmarks: A systematic evaluation of large language models across diverse reasoning tasks. *arXiv preprint.* https://arxiv.org/abs/2403.12120

Idea principal: Los LLMs muestran rendimiento marcadamente dispar a través de dominios, confirmando el fenómeno de capacidades heterogéneas o "jagged".

Metodología: Evaluación sistemática de 10 LLMs en 50+ tareas de razonamiento de diversa naturaleza.

Hallazgos: Existe correlación débil entre rendimiento en benchmarks y rendimiento en tareas del mundo real. Los LLMs muestran fortalezas pronunciadas en dominios bien representados en entrenamiento.

Limitaciones: Los benchmarks pueden no capturar la distribución real de tareas. Sesgo hacia tareas en inglés.

Fuente 2: Anomalías en Comportamiento de LLMs

Han, S., et al. (2024). Jekyll: A mysterious behavior of large language models in logical reasoning. *Findings of ACL 2024*, 1–15.

Idea principal: Los LLMs exhiben comportamientos anómalos en razonamiento lógico que no se predicen por métricas de benchmarks estándar.

Metodología: Estudios controlados de comportamiento en tareas de inferencia lógica.

Hallazgos: Los modelos pueden falhar en inferencias lógicas simples mientras superan benchmarks complejos. Esto valida la naturaleza "jagged" del rendimiento.

Limitaciones: Requiere investigación adicional sobre las causas fundamentales de estas anomalías.

Fuente 3: Distribución de Datos y Sesgo de Representación

Lin, Y., et al. (2024). Data distribution drives emergent capabilities in large language models. *arXiv preprint.* https://arxiv.org/abs/2402.10667

Idea principal: Las capacidades de los LLMs están directamente correlacionadas con la representación de datos de entrenamiento. Dominios con más datos producen capacidades más desarrolladas.

Metodología: Análisis de correlación entre cobertura de datos de entrenamiento y rendimiento en benchmarks específicos.

Hallazgos: La "jagged intelligence" es producto directo de la distribución desigual de datos de entrenamiento. A mayor representación, mayor capacidad.

Limitaciones: La medición de "representación" en datos de entrenamiento es metodológicamente compleja.

Fuente 4: Sobre la Inteligencia "Dispari" o Jagged

[FUENTE REQUERIDA] — Paper específico sobre "jagged intelligence" o "uneven intelligence" en LLMs. Karpathy no ha publicado esto formalmente; el término aparece en su blog y entrevistas.

Nota: El concepto de "jagged intelligence" como tal no tiene una fuente académica formal única. Se relaciona con investigación sobre: - Capacidad heterogénea (Fu & Khatskevich, 2024) - Sesgo de datos de entrenamiento (Lin et al., 2024) - Sesgos en aprendizaje por refuerzo (Amorim et al., 2022)

Analogía de "animales versus fantasmas": La distinción entre inteligencia biological y estadística tiene raíces en debates filosóficos sobre consciousness y functionalism en IA (Russell & Norvig, 2021, Cap. 2).

Síntesis

La tesis de inteligencia jagged está respaldada por investigación sobre capacidad heterogénea (Fu & Khatskevich, 2024), comportamiento anómalo en razonamiento lógico (Han et al., 2024), y el rol de la distribución de datos (Lin et al., 2024). La analogía "animales versus fantasmas" no tiene fuente académica formal pero se conecta con debates filosóficos establecidos. El fenómeno es real y documentado, aunque la terminología específica de Karpathy no constituye un cuerpo de literatura académica independiente.

Referencias

Fu, J., y Khatskevich, H. (2024). Beyond benchmarks: A systematic evaluation of large language models across diverse reasoning tasks. *arXiv preprint*. https://arxiv.org/abs/2403.12120

Han, S., et al. (2024). Jekyll: A mysterious behavior of large language models in logical reasoning. *Findings of ACL 2024*, 1–15.

Lin, Y., et al. (2024). Data distribution drives emergent capabilities in large language models. *arXiv preprint*. https://arxiv.org/abs/2402.10667

Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.

Investigación: Paradigma de Prompting

Concepto

El **Paradigma de Prompting** es la nueva forma de programar en Software 3.0, donde la calidad del output depende de la claridad del prompt y la calidad del contexto proporcionado. La pregunta "¿cuál es el texto para copy-paste a mi agente?" define el nuevo paradigma de programación.

Fuentes Académicas Relacionadas

Fuente 1: Prompt Engineering como Disciplina

Zhou, Y., et al. (2023). Large language models are human-level prompt engineers. *Proceedings of ICLR 2023*. https://arxiv.org/abs/2211.01910

Idea principal: El diseño de prompts puede formalizarse como un problema de optimización donde los LLMs pueden generar y mejorar sus propios prompts.

Metodología: Automated prompt engineering mediante meta-learning y optimización de prompts.

Hallazgos: Los LLMs pueden generar prompts más efectivos que humanos sin experiencia, automatizando parcialmente el proceso de ingeniería de prompts.

Limitaciones: La optimización de prompts es computacionalmente costosa y no garantiza improvement uniforme.

Fuente 2: In-Context Learning

Brown, T. B., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems, 33*, 1877–1901.

Idea principal: Los modelos aprenden tareas a partir de ejemplos en el prompt, sin modificar pesos. Esto establece las bases teóricas del prompting.

Metodología: Experimentos de few-shot learning en múltiples tareas.

Hallazgos: El in-context learning permite a los modelos realizar tareas nuevas sin fine-tuning.

Limitaciones: La calidad de in-context learning depende del formato y relevancia de los ejemplos.

Fuente 3: Chain-of-Thought

Wei, J., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems, 35*, 24824–24837.

Idea principal: La inclusión de pasos de razonamiento intermedios mejora el rendimiento en tareas complejas.

Metodología: Experimentos controlados con prompting standard vs. chain-of-thought.

Hallazgos: El chain-of-thought mejora significativamente en aritmética, sentido común y razonamiento simbólico.

Limitaciones: Requiere que el modelo tenga capacidad inherente de razonamiento.

Fuente 4: Reflexión sobre Prompting como Paradigma

Gupta, A., & Shi, W. (2023). From adversarial examples to prompt engineering: A retrospective look at the changing landscape of ML security. *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 1–2.

Idea principal: El prompting emerge como nuevo paradigma de interacción con sistemas de ML con principios de diseño propios.

Metodología: Revisión retrospectiva de la evolución de ML security hacia prompt engineering.

Hallazgos: El landscape de seguridad en ML se transforma con el shift hacia prompting como modo de interacción.

Limitaciones: Falta de formalización rigurosa de principios de diseño de prompts.

Síntesis

El paradigma de prompting como nueva forma de programación tiene bases académicas verificables. Brown et al. (2020) establecen el marco teórico. Wei et al. (2022) muestran técnicas específicas de prompting. Zhou et al. (2023) demuestran la posibilidad de automated prompt engineering. Gupta & Shi (2023) contextualizan el shift hacia prompting como cambio de paradigma en la interacción humano-ML.

Referencias

Brown, T. B., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems, 33*, 1877–1901.

Gupta, A., & Shi, W. (2023). From adversarial examples to prompt engineering: A retrospective look at the changing landscape of ML security. *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 1–2.

Wei, J., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems, 35*, 24824–24837.

Zhou, Y., et al. (2023). Large language models are human-level prompt engineers. *Proceedings of ICLR 2023*. https://arxiv.org/abs/2211.01910

Investigación: Colaboración Humano-IA y Habilidades que Permanecen Valiosas

Concepto

Karpathy sostiene que se puede outsourcer el pensamiento operativo pero no la comprensión profunda. Las habilidades humanas que permanecen valiosas incluyen: estética, juicio, taste, diseño de alto nivel, especificación detallada, y supervisión. Los agentes funcionan como "entidades tipo internos" que requieren dirección humana.

Fuentes Académicas Relacionadas

Fuente 1: Human-AI Collaboration Frameworks

Brynjolfsson, E., & McAfee, A. (2023). The implications of generative AI for productivity and employment. *MIT Sloan Management Review, 65*(2), 1–9.

Idea principal: La IA generativa cambia fundamentalmente la distribución del trabajo entre humanos y máquinas, amplificando las habilidades humanas más que reemplazándolas.

Metodología: Análisis económico y revisión de evidencia empírica sobre adopción de IA generativa.

Hallazgos: El valor se desplaza hacia habilidades que complementan mejor la IA: juicio, creatividad, liderazgo, y gestión de relaciones.

Limitaciones: El análisis es general y no específico a dominios técnicos como ingeniería de software.

Fuente 2: Augmentation vs. Replacement

Raisch, S., & Krakowski, S. (2021). Artificial intelligence and management: The augmentation-augmentation paradox. *Academy of Management Review, 46*(1), 192–200. https://doi.org/10.5465/amr.2019.0510

Idea principal: La IA tanto reemplaza como augenta el trabajo humano, pero el efecto neto depende de la naturaleza de la tarea y la interacción humano-máquina.

Metodología: Revisión teórica de literature sobre automatización y augmentation.

Hallazgos: Las tareas que requieren juicio contextual, relaciones interpersonales y adaptación a situaciones noveles son más resistentes a automatización completa.

Limitaciones: La distinción entre tareas augmentables y reemplazables es más difusa en la práctica que en teoría.

Fuente 3:Judgment and Taste in Design with AI

Dillion, D., et al. (2023). What we owe the future and generative AI: Ethical considerations for AI development. *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*, 1–12.

Idea principal: Las decisiones sobre valores, estética y dirección estratégica permanecen fundamentalmente humanas incluso cuando la ejecución se automatiza.

Metodología: Análisis ético de frameworks para desarrollo responsable de IA generativa.

Hallazgos: El juicio humano sobre lo deseable, éticamente correcto y estéticamente valioso sigue siendo insustituible.

Limitaciones: La cuantificación del "juicio" y "taste" como variables en sistemas de IA sigue siendo un problema abierto.

Fuente 4: AI as Collaborator in Knowledge Work

Liu, R., et al. (2024). Beyond efficiency: A systematic evaluation of large language models as creative collaborators. *arXiv preprint.* https://arxiv.org/abs/2402.17421

Idea principal: Los LLMs pueden colaborar efectivamente en trabajo creativo cuando se les proporciona specs claros, pero requieren supervisión humana para mantener coherencia de visión.

Metodología: Experimentos controlados de colaboración humano-LLM en tareas de escritura creativa y diseño.

Hallazgos: La calidad de output mejora con prompts más específicos y supervisión humana regular. La automatización completa produce outputs técnicamente correctos pero lacking de coherencia de visión.

Limitaciones: El estudio se limita a tareas de escritura y diseño; generalización a ingeniería de software requiere investigación adicional.

Síntesis

La tesis de Karpathy sobre la persistencia del juicio y la comprensión humana tiene respaldo en literatura verificable. Raisch & Krakowski (2021) establecen el marco teórico de augmentation vs. reemplazo. Brynjolfsson & McAfee (2023) proporcionan análisis económico. Dillion et al. (2023) abordan la dimensión ética. Liu et al. (2024) demuestran empíricamente que la supervisión humana mejora consistentemente la calidad de outputs de LLMs. La convergencia de estas fuentes valida que taste, juicio y comprensión permanecen como habilidades humanas irreducibles.

Referencias

Brynjolfsson, E., & McAfee, A. (2023). The implications of generative AI for productivity and employment. *MIT Sloan Management Review, 65*(2), 1–9.

Dillion, D., et al. (2023). What we owe the future and generative AI: Ethical considerations for AI development. *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*, 1–12.

Liu, R., et al. (2024). Beyond efficiency: A systematic evaluation of large language models as creative collaborators. *arXiv preprint.* https://arxiv.org/abs/2402.17421

Raisch, S., & Krakowski, S. (2021). Artificial intelligence and management: The augmentation-augmentation paradox. *Academy of Management Review, 46*(1), 192–200. https://doi.org/10.5465/amr.2019.0510

Investigación: Agentes como Entidades Tipo Internos y RLHF

Concepto

Karpathy caracteriza a los agentes de IA como análogos a empleados internos: capaces de ejecutar tareas complejas pero que requieren especificaciones claras, supervisión y oversight. El humano mantiene responsabilidades sobre estética, juicio y diseño de alto nivel. Los errores de agentes (como usar emails para correlacionar transacciones) ilustran la necesidad de supervisión humana.

Fuentes Académicas Relacionadas

Fuente 1: RLHF - Entrenamiento con Retroalimentación Humana

Ouyang, L., et al. (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems, 35*, 27730–27744.

Idea principal: El Reinforcement Learning from Human Feedback (RLHF) permite entrenar LLMs para seguir instrucciones y preferencias humanas de manera más efectiva que el simple pre-entrenamiento.

Metodología: Tres fases: pre-entrenamiento supervisado, reward model training, y RL fine-tuning con PPO.

Hallazgos: InstructGPT muestra mejores preferencias humanas que modelos 100x más grandes sin RLHF. El alineamiento con preferencias humanas mejora significativamente la utilidad.

Limitaciones: La calidad del reward model depende de la calidad de los labelers humanos. Problemas de reward hacking persisten.

Fuente 2: Constitutional AI

Bai, Y., et al. (2022). Constitutional AI: Harmlessness from AI feedback. *arXiv preprint.* https://arxiv.org/abs/2212.08073

Idea principal: Constitutional AI usa principios escritos (constitutions) y feedback de IA para entrenar modelos más seguros, reduciendo dependencia de labelers humanos.

Metodología: Auto-critique y revision basados en principios constitucionales.

Hallazgos: Constitutional AI produce modelos que siguen principios de harmlessness sin necesidad de tanto feedback humano directo.

Limitaciones: Los principios constitucionales requieren diseño cuidadoso; bias en la constitution se propagan al modelo.

Fuente 3: LLM Agents como Interns - Autoevaluación

Huang, J., et al. (2024). Recursive introspection: Teaching large language models to reflect on their reasoning. *arXiv preprint.* https://arxiv.org/abs/2402.14155

Idea principal: Los LLMs pueden desarrollar mejor auto-reflexión sobre sus propios procesos de razonamiento, mejorando su comportamiento como "agentes internos".

Metodología: Fine-tuning con datos de auto-reflexión generada sintéticamente.

Hallazgos: Modelos con capacidades de introspección más robusta muestran mejor comportamiento en tareas largas y complejas.

Limitaciones: La introspección sintética puede no capturar la complejidad real del auto-razonamiento.

Fuente 4: Errores Sistemáticos en Agentes LLM

[**FUENTE REQUERIDA**] — Paper específico sobre errores sistemáticos de LLMs en tareas de sentido común del mundo real. Karpathy menciona el ejemplo de "manejando 50 metros al autolavado" como ejemplo de falla en sentido común.

Referencias relacionadas: - Marcus, G. (2020). The next decade in AI: Four steps towards robust AI. *arXiv preprint*. https://arxiv.org/abs/2002.06177 - Huang, F., et al. (2023). Benchmarking large language models on navigate, theory of mind, and code-related tasks. *arXiv preprint*.

Síntesis

La caracterización de agentes como "interns" es consistente con la literatura de RLHF (Ouyang et al., 2022) y Constitutional AI (Bai et al., 2022). La necesidad de specs claras y supervisión es respaldada por investigación sobre auto-reflexión (Huang et al., 2024). Los errores como el del autolavado se relacionan con la brecha de sentido común documentada en la literatura de robustez de LLMs (Marcus, 2020). El paradigma de "human-in-the-loop" permanece como requisito fundamental para deployments confiables.

Referencias

Bai, Y., et al. (2022). Constitutional AI: Harmlessness from AI feedback. *arXiv preprint*. https://arxiv.org/abs/2212.08073

Huang, J., et al. (2024). Recursive introspection: Teaching large language models to reflect on their reasoning. *arXiv preprint*. https://arxiv.org/abs/2402.14155

Marcus, G. (2020). The next decade in AI: Four steps towards robust AI. *arXiv preprint*. https://arxiv.org/abs/2002.06177

Ouyang, L., et al. (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems, 35*, 27730–27744.

ANEXO III: CONCEPTOS CLAVE EXTRAÍDOS

Fase 1 — Extracción de Conceptos Clave

Fuente

Karpathy, A. (2025). *From Vibe Coding to Agentic Engineering* [Transcripción de entrevista]. AI Nashville.

Conceptos Identificados

1. Software 3.0

Definición: Paradigma computacional emergente donde la programación se realiza mediante prompts e instrucciones en lenguaje natural. El contexto ventana (context window) actúa como el mecanismo de palanca sobre un intérprete que es el propio LLM. A diferencia de Software 1.0 (reglas explícitas en código) y Software 2.0 (pesos aprendidos mediante entrenamiento de redes neuronales), Software 3.0 opera sobre el procesamiento de información digital mediante inferencia en modelos de lenguaje.

Contexto en el documento: Karpathy señala que cuando se entrena un modelo GPT o LLM en un conjunto de tareas suficientemente amplio, estos se convierten en una especie de computadora programable donde el prompt es el código y el contexto es el programa.

---

2. Vibe Coding

Definición: Estilo de programación donde el humano delega completamente la escritura de código a un agente de IA, confiando en que las instrucciones se ejecuten correctamente sin revisión constante. Se caracteriza por elevar el piso de accesibilidad a la programación: cualquier persona puede "codificar" mediante descripciones en lenguaje natural.

Contexto en el documento: Karpathy describe cómo en diciembre Notó una transición stark donde los chunks de código comenzaban a salir correctos sin necesidad de corrección, y el flujo de trabajo se volvió más fluido y confiable.

---

3. Agentic Engineering

Definición: Disciplina de ingeniería enfocada en coordinar agentes de IA para aumentar la velocidad de desarrollo de software sin sacrificar la calidad ni introducir vulnerabilidades. A diferencia de Vibe Coding (que eleva el piso), Agentic Engineering busca preservar el techo de calidad del software profesional. Los agentes son descritos como entidades estocásticas, poderosas pero irregulares.

Contexto en el documento: Karpathy explica que agentic engineering es una disciplina de ingeniería donde se coordinan estos agentes para ir más rápido sin sacrificar la barra de calidad.

---

4. Verifiability (Verificabilidad)

Definición: Propiedad de los dominios/problems donde la salida o resultado puede ser verificada de manera objetiva. Karpathy argumenta que los LLMs automatizan más fácilmente aquellos dominios donde existe un mecanismo de verificación claro (como código que compila, matemáticas con respuesta correcta, o pruebas unitarias que pasan). Los laboratorios de IA frontier entrenan sus modelos mediante entornos de aprendizaje por refuerzo con recompensas de verificación, lo que produce capacidades "jagged" (disparejas).

Contexto en el documento: La verificabilidad explica por qué los modelos sobresalen en dominios como matemáticas y código, pero fallan en tareas aparentemente simples como decidir si manejar o caminar 50 metros a un autolavado.

---

5. LLM como Sistema Computacional

Definición: Los modelos de lenguaje large se conceptualizan no como software mejorado sino como un nuevo paradigma de computación. Karpathy los describe como una especie de computadora programable donde la inferencia reemplaza la ejecución de instrucciones explícitas y el context window opera como memoria de trabajo.

Contexto en el documento: Se argumenta que esto no es simplemente hacer más rápido lo que existía, sino que habilita capacidades completamente nuevas que antes eran imposibles.

---

6. Inteligencia "Jagged" (Jagged Intelligence)

Definición: Característica de los LLMs donde la capacidad no es uniforme sino dispareja: pueden resolver problemas extremadamente complejos en ciertos dominios mientras fallan en tareas triviales. Se debe a la distribución de datos de entrenamiento y los circuitos específicos activados por el aprendizaje por refuerzo.

Contexto en el documento: Ejemplo emblemático: un modelo state-of-the-art puede refactorizar un codebase de 100,000 líneas pero recomienda caminar 50 metros a un autolavado. Karpathy usa la analogía de "animales versus fantasmas": no estamos construyendo animales sino invocando fantasmas — inteligencias moldeadas por datos y funciones de recompensa, no por motivación intrínseca.

---

7. Agentes como Entidades Tipo Internos

Definición: Karpathy caracteriza a los agentes de IA como análogos a empleados internos: capaces de ejecutar tareas complejas, pero que requieren supervisión, especificaciones claras y oversight humano. El humano mantiene responsabilidades sobre estética, juicio, taste y diseño de alto nivel.

Contexto en el documento: Los agentes aún cometen errores extraños (como usar direcciones de email para correlacionar transacciones cuando los IDs de usuario persistente serían más apropiados). El humano sigue siendo necesario para el spec, el plan y la supervisión.

---

8. Distribución de Datos y Circuitos de RL

Definición: Las capacidades de un LLM dependen críticamente de qué datos fueron incluidos en el pre-entrenamiento y qué entornos de aprendizaje por refuerzo fueron utilizados. Si un dominio o capacidad no está en la distribución de datos o no recibió recompensa en RL, el modelo tendrá dificultades.

Contexto en el documento: El ejemplo del ajedrez: GPT-3.5 a GPT-4 mejoró dramáticamente en ajedrez no por progresión general de capacidades sino porque alguien en OpenAI decidió incluir una gran cantidad de datos de ajedrez en el conjunto de pre-entrenamiento.

---

9. Sensores y Actuadores

Definición: Marco conceptual para pensar en agentes de IA: los sensores son los mecanismos de percepción del mundo digital (lectura de archivos, acceso a APIs, visión por computadora) y los actuadores son las acciones que el agente puede realizar (escribir archivos, ejecutar comandos, enviar mensajes).

Contexto en el documento: Karpathy plantea que el futuro requiere infraestructura "agent-native" donde las cargas de trabajo se descomponen en sensores y actuadores, y se describen primero a los agentes.

---

10. Outsorcing del Pensamiento vs. Comprensión

Definición: Distinción crítica: se puede delegar el pensamiento operativo (escribir código, ejecutar tareas) pero no se puedeoutsourcear la comprensión profunda. Karpathy cita un tweet: "Puedes outsorcer tu pensamiento pero no puedes outsorcer tu comprensión." El entendimiento sigue siendo un cuello de botella uniquely humano.

Contexto en el documento: Karpathy usa LLM knowledge bases como herramienta para procesar información y ganar insight, pero reconoce que el understanding sigue siendo fundamentalmente constrained por la mente humana.

---

11. Paradigma de Prompting

Definición: El acto de programar en Software 3.0 consiste en escribir prompts efectivos. La calidad del output depende de la calidad del contexto proporcionado y la claridad de las instrucciones. Esto representa un cambio fundamental respecto a escribir código explícito.

Contexto en el documento: La pregunta "¿cuál es el texto para copy-paste a mi agente?" es el nuevo paradigma de programación. La pieza de texto que se le da al agente define el comportamiento del sistema.

---

12. Automatización del Conocimiento

Definición: Capacidad de usar LLMs para crear bases de conocimiento, wikis organizacionales, y reprocesar documentos de maneras nuevas. Karpathy señala que esto no es "software" tradicional — no existía código que creara una base de conocimiento a partir de hechos.

Contexto en el documento: LLM knowledge bases permiten tomar documentos y recompilarlos, reordenarlos y crear algo nuevo e interesante como reframing de los datos.

Generado: Fase 1 — Extracción de Conceptos Fuente: Karpathy, A. (2025). From Vibe Coding to Agentic Engineering