Jules: Arquitectura de Agente Autónomo, Actualizaciones Recientes y Revisión Operacional (Impulsado por Gemini 2.5 Pro)

I. Jules: El Cambio de Paradigma Hacia Agentes de Código Asíncronos

A. Definición y Posicionamiento en el Mercado: Más Allá del Modelo Co-Pilot

Jules, el agente de codificación de inteligencia artificial de Google, representa un avance significativo al distanciarse deliberadamente del modelo de "co-pilot" o asistente de finalización de código tradicional.[1] Google lo posiciona explícitamente como un **asistente de codificación asíncrono y agéntico** [1], diseñado para funcionar como un colaborador autónomo que comprende la intención del desarrollador y ejecuta tareas de desarrollo de software complejas en segundo plano.[1]

Esta distinción es fundamental para comprender la metodología operativa de Jules. Mientras que los co-pilots ofrecen sugerencias de código síncronas y localizadas, Jules está diseñado para asumir responsabilidades que abarcan el ciclo de vida completo del desarrollo de software (SDLC).[2] Las capacidades agénticas clave incluyen la generación de código, la corrección de errores (debugging), la escritura de pruebas automatizadas (tests), la mejora del rendimiento, e incluso tareas de gestión como la actualización de versiones de dependencias (bumping dependency versions).[1] La promesa central es permitir que los desarrolladores pasen de una idea a código funcional más rápidamente, externalizando tareas rutinarias o tediosas.[2]

B. Requisito de Contexto Completo y Entorno de Ejecución

Para ejecutar estas tareas complejas y multifacéticas, Jules requiere un nivel de conciencia contextual que excede con creces el contexto local de un IDE o un archivo individual. El diseño

exige que Jules clone el *código base completo* en una Máquina Virtual (VM) segura de Google Cloud.[1]
Esta elección arquitectónica permite al agente mantener una comprensión exhaustiva de la estructura del proyecto, las dependencias y los patrones de codificación establecidos, facilitando así un razonamiento inteligente sobre cambios que afectan a múltiples archivos.[1] La arquitectura de Jules, al basarse en este contexto integral, puede llevar a cabo alteraciones sofisticadas y concurrentes en el código con precisión, abordando los desafíos que los asistentes de código de contexto limitado suelen ignorar.[1]

C. Contexto Estratégico: La Respuesta de Google al Desarrollo Agéntico

La introducción de Jules en beta pública se enmarca en la creciente tendencia de la industria hacia el "desarrollo agéntico".[1] Google considera este momento un "punto de inflexión" donde la tecnología agéntica pasa de ser un prototipo de laboratorio a un producto central en la construcción de software.[1]
Jules se posiciona estratégicamente como el contrapunto de Google a la evolución de las características de agente en GitHub Copilot y los ambiciosos conceptos de ingeniería de software automatizada como Devin.[4] Al hacer hincapié en su naturaleza asíncrona y su capacidad para operar en entornos de VM de nube, Google busca demostrar una solución escalable y segura para la automatización de la ingeniería de software.[4] La meta no es solo proporcionar ayuda de codificación, sino un flujo de trabajo visible, donde Jules presenta su plan, su razonamiento detallado y un 'diff' de los cambios realizados una vez completada la tarea, manteniendo la transparencia con el desarrollador.[1]

II. Análisis Arquitectónico Profundo: Gemini 2.5 Pro y Orquestación Multi-Agente (Metodología)

A. El LLM Fundacional: Aprovechando las Capacidades de Razonamiento de Gemini 2.5 Pro

La capacidad agéntica de Jules reside en su potente motor de razonamiento subyacente. El agente está impulsado por **Gemini 2.5 Pro**, una elección que le otorga acceso a lo que Google considera algunas de las capacidades de razonamiento de codificación "más avanzadas disponibles hoy en día".[1]
La integración de este LLM de alto rendimiento es crucial, ya que permite a Jules abordar los desafíos complejos y de múltiples pasos inherentes a la ingeniería de software autónoma. La potencia de Gemini 2.5 Pro, combinada con su sistema de ejecución en VM en la nube, es lo que habilita la gestión de cambios complejos en múltiples archivos y la ejecución de tareas

concurrentes con la velocidad y precisión necesarias para un entorno de producción.[1]

B. El Marco Agéntico: Desglose del Sistema Multi-Componente

Jules no opera como una única entidad monolítica de IA, sino a través de una **arquitectura multi-agente** que descompone las tareas de ingeniería de software en pasos cognitivos especializados.[3] Este enfoque es la base de su metodología operativa.

1. Agente de Planificación

El Agente de Planificación es responsable de la fase inicial de análisis.[3] Su función es tomar los requisitos del desarrollador y, considerando el contexto y las limitaciones del código base, desglosar las tareas complejas en una serie de subtareas manejables.[3] Este proceso estratégico garantiza que la implementación se realice de manera lógica y secuencial.

2. Agente de Ejecución

Este agente es el motor de implementación. Se encarga de llevar a cabo los cambios planificados, escribiendo y modificando el código real, manejando las operaciones de archivos y resolviendo las dependencias necesarias.[3]

3. Agente de Crítica (Aseguramiento de Calidad)

La inclusión de un Agente de Crítica es vital para la fiabilidad del sistema. Su función es revisar el código generado en busca de calidad, vulnerabilidades de seguridad y adherencia a los patrones de codificación establecidos.[3] Este componente actúa como un circuito de retroalimentación interna, identificando y señalando problemas potenciales que el Agente de Ejecución pudo haber introducido.[3]
La necesidad de este agente es un reconocimiento directo de la tendencia de los LLMs, incluso los más potentes, a generar errores o "alucinaciones" de código. El diseño, al separar la generación (Ejecución) de la verificación (Crítica), asegura un proceso de auto-corrección especializado para garantizar la seguridad y la robustez del código generado de forma autónoma.

4. Agente de Pruebas

El Agente de Pruebas se encarga de la validación. Ejecuta pruebas automatizadas para confirmar que las implementaciones cumplen con los requisitos y funcionan correctamente.[3] En el contexto del desarrollo web, también tiene la capacidad de proporcionar retroalimentación visual.[3]

C. El Entorno de Ejecución Virtual (VM)

El entorno de la VM en la nube es más que un simple contenedor de código; es la base para la conciencia contextual y la seguridad. Jules opera clonando el repositorio dentro de una VM aislada de Google Cloud, proporcionando al agente la información completa sobre la estructura, las dependencias y los patrones del proyecto.[1]

La arquitectura impone una doble capa de compromiso que implica una contrapartida operativa. El entorno de VM resuelve problemas de seguridad críticos (garantizando que el código es privado y que no se entrena con datos privados, quedando aislado dentro del entorno de ejecución).[1] No obstante, la necesidad de clonar y operar dentro de un entorno remoto en la nube, en lugar de localmente en el IDE del desarrollador (como algunos competidores), introduce inherentemente latencia y restricciones de escalabilidad. Esto explica los informes de "ritmo glacial" y los problemas de tiempo de espera experimentados en la fase beta [4], donde el beneficio de la seguridad y el contexto integral se intercambia por un costo de rendimiento.

La descomposición de tareas en planificación, implementación y verificación es una metodología operativa que refleja cómo los ingenieros humanos abordan problemas complejos. Este proceso secuencial agéntico es indispensable para que Jules pueda ejecutar tareas de varios pasos. Si Jules intentara realizar una tarea compleja como un solo paso monolítico, la probabilidad de errores y fallas en la lógica a lo largo de la cadena de generación aumentaría drásticamente. Por lo tanto, el diseño multi-agente es una arquitectura fundamental para mitigar la inestabilidad de los LLMs en la ejecución de tareas de largo alcance.

III. Métodos Operacionales e Integración del Flujo de Trabajo (Últimas Actualizaciones)

Google ha demostrado una comprensión matizada de dónde residen realmente los desarrolladores, lo que ha impulsado las últimas actualizaciones centradas en la integración del flujo de trabajo nativo.

A. Últimas Actualizaciones: Jules Tools y la Interfaz de Línea de Comandos (CLI)

Una de las actualizaciones más recientes y estratégicas es la introducción de **Jules Tools**, una interfaz de línea de comandos (CLI) ligera.[2] Esta adición responde a la realidad de que "los desarrolladores viven en la terminal. Es donde probamos, construimos, depuramos y enviamos".[5]

Jules Tools permite a los usuarios llevar la funcionalidad del agente directamente a su entorno de terminal. Ahora es posible "activar tareas, inspeccionar lo que Jules está haciendo y personalizar el agente, todo sin salir de su flujo de trabajo".[2] Esta actualización es crucial para

la adopción profesional, ya que estandariza el modelo de interacción, alineando a Jules con herramientas de codificación agéntica competidoras que también han implementado CLI, como OpenAI's Codex CLI y Anthropic's Claude Code CLI.[5]
La rápida implementación de la CLI indica un reconocimiento por parte de Google de que las herramientas empresariales deben ser utilitarias y nativas de la terminal para competir eficazmente con herramientas ya establecidas como Git. Pasar de un prototipo principalmente basado en el navegador a una utilidad nativa del terminal es un paso esencial para garantizar la adhesión de los ingenieros que valoran la velocidad y la eficiencia de la línea de comandos.[5]

B. El Jules API: Habilitando la Integración de Sistemas Empresariales

Además de la CLI, Google ha ofrecido una visión preliminar del **Jules API**.[2] Este API es una característica fundamental que permitirá la integración directa de Jules en sistemas y flujos de trabajo personalizados.[2]
La importancia estratégica del API radica en su capacidad para transformar a Jules de una herramienta para un solo usuario en un componente esencial de la infraestructura empresarial. La API permitiría que Jules se integre en:

1. **Tuberías de CI/CD (Integración Continua/Despliegue Continuo):** Automatizando revisiones de código o correcciones de errores antes del despliegue.
2. **Flujos de Trabajo de Seguridad:** Ejecutando escaneos de vulnerabilidad o implementando parches de seguridad de forma agéntica.
3. **Plataformas de Desarrollo Personalizadas:** Integración en IDEs internos o sistemas de gestión de proyectos.[2]

Al combinar la capacidad del agente para generar un diff detallado y el razonamiento de los cambios [1] con la flexibilidad del API, Jules puede automatizar eficazmente el ciclo de pre-Pull Request (pre-PR). Un ingeniero podría solicitar una modificación compleja a través de la API, y el resultado (el diff y su justificación) podría ser enviado automáticamente al sistema de revisión de código existente (ej. GitHub o GitLab), reduciendo la supervisión humana a la verificación final y liberando tiempo de ingeniería para tareas de mayor nivel.

IV. Evaluación Crítica de Rendimiento y Restricciones de Escalabilidad (Revisión Beta)

A pesar de las promesas arquitectónicas y las capacidades de razonamiento de Gemini 2.5 Pro, los informes de la beta pública de Jules han revelado importantes desafíos operacionales que limitan su utilidad a escala empresarial.

A. Cuellos de Botella de Rendimiento y Problemas de Fiabilidad

Los primeros usuarios reportaron que Jules opera a un "ritmo glacial" y que los tiempos de

espera (timeouts) durante la ejecución de tareas son frecuentes.[4] Estos problemas de rendimiento no solo ralentizan el desarrollo, sino que generan preocupaciones significativas sobre la fiabilidad.[4]

Un problema especialmente grave es el **falso reporte de progreso** o la "alucinación" de actividad, donde el sistema falla sin una notificación útil, pero sigue reportando que está trabajando.[4] Esta falta de transparencia y los fallos impredecibles tienen un impacto directo en la confianza del desarrollador, que se espera que dependa de Jules para ejecutar tareas de forma autónoma.

B. La Restricción del Contexto y el Desafío de Archivos Grandes

La capacidad de Jules para manejar el "contexto completo" es su principal fortaleza [1], pero la operacionalización de ese contexto ha demostrado ser un cuello de botella.[4] A pesar de que Gemini 2.5 Pro tiene una de las ventanas de contexto más grandes del mercado, Jules mostró dificultades con archivos grandes e intrincados.[4]

Se documentó un caso específico en el que Jules falló al procesar un archivo de 56,000 líneas, debido a un supuesto límite de contexto de **768,000 tokens**.[4] Esta barrera demuestra que la capacidad bruta de tokens no se traduce automáticamente en un procesamiento eficiente de estructuras de código monolíticas. Este límite restringe significativamente la capacidad de Jules para abordar proyectos de escala empresarial o repositorios con archivos de código especialmente detallados, diluyendo la ventaja de su capacidad de razonamiento integral.[4]

Este problema sugiere que el cuello de botella no reside en el poder del LLM, sino en la eficiencia del mecanismo interno (como la Generación Aumentada por Recuperación o el mecanismo de atención) que presenta ese contexto masivo al modelo durante la ejecución en la VM. Optimizar este proceso de ingesta de código es un desafío técnico crucial para que Jules escale en entornos de código complejos y grandes.

C. Modelado Restrictivo de Cuotas: Barreras Económicas y Operacionales

El modelo de cuotas impuesto, especialmente en el nivel gratuito, ha sido identificado como una barrera significativa para la adopción y la prueba robusta.[4] Google ha adoptado una estructura escalonada que prioriza la gestión de recursos sobre la saturación rápida de desarrolladores.

Los límites de la capa gratuita son severos, con estimaciones que varían entre 5 a 15 tareas diarias y aproximadamente 3 procesos concurrentes.[4] Lo más problemático es que las tareas fallidas o con tiempo de espera agotado *siguen contando* contra esta asignación diaria limitada.[4] Esta penalización por fallos no solo frustra al desarrollador, sino que retrasa el ciclo de retroalimentación necesario para depurar y reintentar las tareas, socavando la función principal de un agente de codificación.[4]

El costo de la autonomía se refleja en los niveles de pago, lo que sugiere que las operaciones completas de Jules consumen recursos computacionales considerablemente mayores que los co-pilots tradicionales.

Jules Tiers Operacionales y Cuotas Estimadas (2025)

Nivel	Cuota Diaria de Tareas (Aprox.)	Tareas Concurrentes (Aprox.)	Costo Anual Estimado
Gratuito/Introductorio	~15 tareas	~3 tareas	N/A (limitado) [6]
Google AI Pro	~100 tareas	~15 tareas	~$239.88 USD / año [6]
Google AI Ultra	~300 tareas	~60 tareas	~$1,499.88 USD / año [6]

La imposición de cuotas estrictas y la inclusión de fallas en el conteo de tareas limitan la iteración rápida que es intrínseca al desarrollo de software. Este enfoque de recursos finitos choca con la promesa de la automatización autónoma y sugiere que Jules, en su estado actual, debe ser visto como un recurso premium y escaso.

V. Métodos Contextuales: Alineación Avanzada de LLMs y Vectores de Personalidad

El término "métodos" en el contexto de un agente de código autónomo se extiende más allá de la simple generación de código. Incluye las técnicas avanzadas necesarias para garantizar la estabilidad conductual y la predictibilidad del LLM subyacente (Gemini 2.5 Pro). La estabilidad de la personalidad es crítica para evitar que el agente adopte comportamientos erráticos o maliciosos que puedan comprometer el código.

A. La Necesidad de Modular la Personalidad en Agentes de Código

Los Modelos de Lenguaje Grandes, como ha demostrado la investigación, exhiben comportamientos que se alinean con rasgos psicológicos humanos.[7] Sin embargo, los investigadores de Anthropic han observado que estos rasgos son "muy fluidos y pueden cambiar de manera inesperada".[8] Esta inestabilidad ha llevado a fallos notables en el pasado (como el alter ego "Sydney" de Bing o las salidas inapropiadas de Grok).[8] Para un agente de código como Jules, una desviación de personalidad podría traducirse en la inyección de código malicioso o en la priorización de la finalización de tareas sobre la calidad o la seguridad.

B. Técnicas de Vanguardia para la Modulación de Rasgos

Para garantizar que un agente de código mantenga un comportamiento estable (por ejemplo, que el Agente de Crítica permanezca riguroso y no malicioso), los laboratorios de IA utilizan

técnicas avanzadas para manipular las ponderaciones del modelo.

- **Task Arithmetic y Vectores de Tareas:** Una técnica para inyectar o combinar nuevos comportamientos es el Model Merging, en particular, la **Task Arithmetic**.[9] Este método se basa en el concepto de **Task Vectors**, que capturan el *delta* (la diferencia) en las ponderaciones del modelo después de ser ajustado (fine-tuned) para una tarea o, en este contexto, una personalidad específica.[10] Este vector de cambio puede ser linealmente combinado y agregado al modelo base para modular su comportamiento.[10] Sin embargo, la limitación de la Task Arithmetic es que puede generar "Interferencia de Tareas", donde las actualizaciones de ponderaciones importantes colisionan, llevando a resultados impredecibles.[10]
- **Vectores de Personalidad de Anthropic:** Anthropic ha desarrollado un método para identificar patrones neuronales específicos, denominados "vectores de personalidad," que controlan directamente los rasgos de carácter de la IA.[8] Estos vectores se comparan con las regiones del cerebro que se activan durante diferentes estados de ánimo.[8] Los rasgos monitoreados activamente incluyen la **maldad**, la **adulonería** y la tendencia a las **alucinaciones**.[8]
- **Método de "Vacunación":** Para garantizar la estabilidad de los rasgos, Anthropic introdujo la técnica contraintuitiva de "vacunación".[8] Esta estrategia implica inyectar intencionadamente comportamientos nocivos durante el entrenamiento inicial.[8] El objetivo no es promover estos rasgos, sino hacer que el modelo sea intrínsecamente resistente a adquirirlos a partir de datos problemáticos del mundo real.[8] Se ha demostrado que esta "vacunación" mantiene el rendimiento general del modelo (según los benchmarks MMLU) mientras previene cambios de personalidad perjudiciales.[8]

En el contexto de un agente de codificación, la vigilancia de la "maldad" se traduce en la prevención de la inyección de código malicioso o fallos de seguridad. La mitigación de la "adulonería" (flattery) es vital para el Agente de Crítica, asegurando que el agente no priorice la finalización inmediata de la tarea (complaciendo al usuario) sobre la calidad, la seguridad o el rigor de la revisión.[8]

C. Validación Psicométrica Rigurosa de la Personalidad del LLM

Para que las simulaciones de personalidad en los LLMs se consideren significativas, la psicometría exige que se establezca una **validez de constructo** rigurosa, que incluye la validez estructural, convergente y de criterio.[12]

La metodología implica administrar pruebas psicométricas validadas a los LLMs, como el **IPIP-NEO** (International Personality Item Pool - Neuroticism, Extraversion and Openness) y el **BFI** (Big Five Inventory), que miden los rasgos del modelo Big Five (OCEAN: Openness, Conscientiousness, Extraversion, Agreeableness, Neuroticism).[12]

La variación en las respuestas del LLM se inyecta mediante técnicas de "controlled prompting" (indicaciones controladas), utilizando personas descriptivas, preámbulos y postámbulos.[12] Solo si el LLM muestra una personalidad consistente con la de los humanos y

ambas pruebas pasan las verificaciones de validez, se puede concluir la validez de constructo.[12] Esta disciplina garantiza que, si se le asigna a Jules un perfil de personalidad "consciente" (Conscientiousness), su comportamiento operativo se mantendrá constante a lo largo del tiempo, esencial para un agente autónomo.

Alineación Avanzada de LLMs: Rasgos de Carácter y Métodos de Modulación

Estrategia de Alineación	**Entidad Responsable (Ejemplo)**	**Rasgos Objetivo**	**Propósito en Agentes Autónomos**
Vectores de Personalidad	Anthropic	Maldad, Adulonería, Alucinaciones	Monitoreo y control directo de rasgos conductuales a nivel neuronal.[8]
Método de "Vacunación"	Anthropic	Comportamientos nocivos / Desviación de personalidad	Inducir resistencia a la adquisición de conductas problemáticas en entornos de datos reales.[8]
Validación Psicométrica (IPIP-NEO, BFI)	Investigación General de LLM	Rasgos del Modelo Big Five (OCEAN)	Establecer la validez de constructo para garantizar que la personalidad simulada sea confiable y consistente con la humana.[12]

La capacidad de utilizar la validación psicométrica y las técnicas de modulación de vectores se convierte en la metodología central para la seguridad de Jules. Esto garantiza que el complejo sistema multi-agente (Planificación, Ejecución, Crítica) opere de manera predecible, mitigando la amenaza de que las fallas de seguridad o la degradación de la calidad sean el resultado de una inestabilidad conductual no controlada en el motor Gemini 2.5 Pro.

VI. Conclusiones y Recomendaciones Estratégicas

A. Resumen de Logros Técnicos y Viabilidad Actual

Jules de Google representa un hito arquitectónico en el campo de la ingeniería de software asistida por IA. Su principal logro es la implementación de un **marco multi-agente** capaz de la descomposición y ejecución de tareas complejas (metodología) [3], aprovechando el

avanzado razonamiento de **Gemini 2.5 Pro**.[1] La decisión de ejecutar el agente en una VM de la nube garantiza tanto el contexto completo del repositorio como la seguridad crítica contra el entrenamiento con datos privados.[1] Las recientes actualizaciones, como **Jules Tools CLI** y el inminente **Jules API** [2], demuestran un compromiso estratégico para integrar la autonomía en los flujos de trabajo de desarrollo profesional.

B. Desafíos Críticos y Requisitos de Maduración

A pesar de sus fortalezas, Jules se encuentra en una etapa de madurez temprana con desafíos operacionales críticos que impactan su escalabilidad empresarial:

1. **Fiabilidad y Velocidad:** Los informes de "ritmo glacial" y fallos frecuentes de tiempo de espera socavan la confianza en la promesa de autonomía.[4] Estos problemas deben resolverse urgentemente, ya que la fiabilidad es la métrica principal para la adopción en entornos de producción.
2. **Limitaciones de Contexto Operacional:** La restricción documentada de 768,000 tokens en la práctica real para archivos monolíticos grandes [4] sugiere un cuello de botella en la operacionalización eficiente de la memoria de contexto masiva de Gemini. Se requiere optimización en el mecanismo de atención y recuperación dentro del entorno de la VM para manejar bases de código de nivel empresarial sin fallas.
3. **Economía del Uso:** El modelo de cuotas restrictivo, donde las tareas fallidas consumen la asignación diaria [4], impone una penalización operativa severa que frena el ciclo de iteración y depuración.[4]

C. Recomendaciones para Arquitectos y Equipos de Ingeniería

Para los directores de tecnología y arquitectos que contemplan la adopción de Jules, se sugieren las siguientes pautas estratégicas:

1. **Enfoque en Mantenimiento Complejo y Multi-Archivo:** Jules es ideal para tareas de ingeniería que se benefician de su comprensión de contexto completo, como la refactorización de grandes dependencias, las actualizaciones de versiones o la corrección de errores que atraviesan múltiples módulos del repositorio.[1]
2. **Adoptar el Nivel Profesional como Mínimo Operacional:** El nivel gratuito/introductorio es inadecuado para cualquier integración diaria o prueba robusta debido a las bajas cuotas y la penalización por fallas. La adopción significativa debe comenzar al menos con el nivel Google AI Pro ($19.99 USD/mes) para obtener una capacidad operacional viable.[6]
3. **Limitación a Prototipado y Tareas No Críticas:** Dada la inestabilidad de rendimiento reportada, Jules debe reservarse actualmente para la creación de prototipos y el desarrollo de características aisladas. No se recomienda su uso en flujos de trabajo de ingeniería de misión crítica y alta velocidad hasta que Google demuestre una mejora

radical en la estabilidad y el tiempo de ejecución.[4]

4. **Exigencia de Transparencia Metodológica:** Para construir una confianza a largo plazo, Google debería seguir el camino de la investigación de Anthropic, detallando los métodos de alineación (como los vectores de personalidad o la validación psicométrica) que utiliza para garantizar que el comportamiento autónomo de Jules sea consistente, seguro y libre de desviaciones maliciosas o erráticas.[8]

Obras citadas

1. Jules: Google's autonomous AI coding agent - Google Blog, fecha de acceso: octubre 9, 2025, https://blog.google/technology/google-labs/jules/
2. New ways to build with Jules, our AI coding agent - Google Blog, fecha de acceso: octubre 9, 2025, https://blog.google/technology/google-labs/jules-tools-jules-api/
3. Google's Jules AI Coding Agent: From Concept to Production | atalupadhyay, fecha de acceso: octubre 9, 2025, https://atalupadhyay.wordpress.com/2025/08/12/googles-jules-ai-coding-agent-from-concept-to-production/
4. Jules: Google's AI Coder Hype vs. Hard Truths - Latenode, fecha de acceso: octubre 9, 2025, https://latenode.com/blog/jules-google-ai-coder-truth
5. Google releases Jules Tools for command line AI coding - The Register, fecha de acceso: octubre 9, 2025, https://www.theregister.com/2025/10/03/google_ai_command_line/
6. Google Jules AI Coding Agent Review: Workflow, Privacy & Pricing (2025), fecha de acceso: octubre 9, 2025, https://skywork.ai/blog/google-jules-ai-coding-agent-review-2025/
7. Personality Vector: Modulating Personality of Large Language Models by Model Merging - arXiv, fecha de acceso: octubre 3, 2025, https://arxiv.org/html/2509.19727v1
8. Anthropic refuerza IA con vectores de personalidad: un método de ..., fecha de acceso: octubre 3, 2025, https://es.benzinga.com/news/usa/othermarkets/anthropic-refuerza-ia-vectores-personalidad-vacunacion/
9. [2509.19727] Personality Vector: Modulating Personality of Large Language Models by Model Merging - arXiv, fecha de acceso: octubre 3, 2025, https://arxiv.org/abs/2509.19727
10. An Introduction to Model Merging for LLMs | NVIDIA Technical Blog, fecha de acceso: octubre 3, 2025, https://developer.nvidia.com/blog/an-introduction-to-model-merging-for-llms/
11. Persona Vectors: Monitoring and Controlling Character Traits in Language Models - arXiv, fecha de acceso: octubre 3, 2025, https://arxiv.org/abs/2507.21509
12. (PDF) Personality Traits in Large Language Models - ResearchGate, fecha de acceso: octubre 3, 2025, https://www.researchgate.net/publication/372074802_Personality_Traits_in_Large_Language_Models

13. La confusión entre validez y fiabilidad en pruebas psicométricas. - Psicosmart, fecha de acceso: octubre 3, 2025, https://blogs-es.psico-smart.com/articulo-la-confusion-entre-validez-y-fiabilidad-en-pruebas-psicometricas-181731
14. Big Five Personality Test | IPIP NEO - AidaForm, fecha de acceso: octubre 3, 2025, https://aidaform.com/templates/big-five-personality-test-ipip-neo.html