ISBN: 978-628-96613-3-0. ISSN: 2414-6390. Digital Object Identifier: https://dx.doi.org/10.18687/LACCEI2026.1.1.2421

Cognitive Dependency in AI-Assisted Programming: Correlation Between GitHub Copilot Usage and Syntactic Memory Degradation in Engineering Students

Benites-Rodriguez, Joseph[1]; Benites-Angulo, Luis[2]; Benites-Angulo, Carlos[2]; Tabacchi-Murillo, Jesus[1]; Rodriguez-Terrones, Jose[3]; Amaro-Guzman, Carlos[1]; Torres-Quiroz, Almintor[1]
[1]Universidad Nacional del Callao - (PE), Perú, jmbenitesr@unac.edu.pe; jatabacchim@unac.edu.pe; cjamarog@unac.edu.pe; agtorresq@unac.edu.pe
[2]Universidad Nacional de Trujillo - (PE), Perú, lbenitesa@gmail.com; cbenitesa@gmail.com
[3]Universidad Privada Antenor Orrego - (PE), Perú, jrodriguezt200@upao.edu.pe

Abstract*– This longitudinal study examines the cognitive impact of GitHub Copilot on 1,460 engineering students. Using a mixed-methods approach combining cognitive load theory assessments, syntactic retention tests, and development speed metrics, we document a significant inverse relationship between AI assistant dependency and long-term syntactic memory consolidation ($r = -0.67$, $p < 0.001$). While intensive Copilot users demonstrated 34.2% faster task completion times initially, they exhibited 41.8% lower syntactic recall in delayed assessments (Week 16) and 53.6% reduced performance in unassisted programming conditions. Analysis revealed that intensive AI use correlates with decreased Germanic cognitive load ($r = -0.54$, $p < 0.001$), suggesting reduced deep processing essential for skill acquisition. Domain-specific analysis showed a pronounced deterioration in advanced constructs: object-oriented programming (-42.1%), functional programming (-38.7%), and complex data structures (-37.3%). Structural equation modeling confirmed mediation through Germanic cognitive load (indirect effect = -0.33, 95% CI [-0.39, -0.27]), explaining 49% of the total relationship. These findings reveal a critical productivity-learning paradox in AI-assisted programming education, with implications for curriculum design and pedagogical practice in computer science education.

Keywords*-- GitHub Copilot, cognitive dependency, syntactic memory, cognitive load theory, students

Dependencia Cognitiva en Programación Asistida por IA: Correlación entre el Uso de GitHub Copilot y la Degradación de Memoria Sintáctica en Estudiantes de Ingeniería

Benites-Rodriguez, Joseph[1]; Benites-Angulo, Luis[2]; Benites-Angulo, Carlos[2]; Tabacchi-Murillo, Jesus[1]; Rodriguez-Terrones, Jose[3]; Amaro-Guzman, Carlos[1]; Torres-Quiroz, Almintor[1]
[1]Universidad Nacional del Callao - (PE), Perú, jmbenitesr@unac.edu.pe; jatabacchim@unac.edu.pe; cjamarog@unac.edu.pe; agtorresq@unac.edu.pe
[2]Universidad Nacional de Trujillo - (PE), Perú, lbenitesa@gmail.com; cbenitesa@gmail.com
[3]Universidad Privada Antenor Orrego - (PE), Perú, jrodriguezt200@upao.edu.pe

Resumen*– Este estudio longitudinal examina el impacto cognitivo de GitHub Copilot en 1,460 estudiantes de ingeniería. Utilizando un enfoque de métodos mixtos que combina evaluaciones de teoría de carga cognitiva, pruebas de retención sintáctica y métricas de velocidad de desarrollo, documentamos una relación inversa significativa entre la dependencia de asistentes de IA y la consolidación de memoria sintáctica a largo plazo ($r = -0.67$, $p < 0.001$). Mientras que los usuarios intensivos de Copilot demostraron tiempos de finalización de tareas 34.2% más rápidos inicialmente, exhibieron 41.8% menor recuerdo sintáctico en evaluaciones diferidas (Semana 16) y 53.6% reducción de rendimiento en condiciones de programación no asistida. El análisis reveló que el uso intensivo de IA correlaciona con disminución de carga cognitiva germánica ($r = -0.54$, $p < 0.001$), sugiriendo un procesamiento profundo reducido esencial para la adquisición de habilidades. El análisis específico por dominio mostró un deterioro pronunciado en constructos avanzados: programación orientada a objetos (-42.1%), programación funcional (-38.7%) y estructuras de datos complejas (-37.3%). El modelado de ecuaciones estructurales confirmó una mediación a través de carga cognitiva germánica (efecto indirecto = -0.33, IC 95% [-0.39, -0.27]), explicando el 49% de la relación total. Estos hallazgos revelan una paradoja crítica de productividad-aprendizaje en educación de programación asistida por IA, con implicaciones para diseño curricular y práctica pedagógica en educación computacional.

***Palabras clave*-- GitHub Copilot, dependencia cognitiva, memoria sintáctica, teoría de carga cognitiva, estudiantes**

I. Introducción

La presencia de Inteligencia Artificial generativa integrada en el desarrollo de software ha provocado un cambio paradigmático en la educación en programación. GitHub Copilot, lanzado comercialmente en junio de 2021, es el primer ejemplo de asistencia de codificación a gran escala basada en modelos de lenguaje generativos (LLMs) y actualmente está procesando más del 21% del código producido en las principales empresas tecnológicas [1]. Este suceso impacta en los entornos de desarrollo profesional y educativos, donde se plantea serias consideraciones sobre los impactos cognitivos en los estudiantes que están construyendo habilidades de programación fundamentales.

La investigación sobre productividad ha evidenciado ganancias significativas y consistentes. En estudios controlados, se informó que los tiempos de finalización de tareas mejoraron entre un 26% y un 55% [2][3], con los impactos más significativos en desarrolladores novatos que experimentaron mejoras estimadas entre el 35% y el 39% [4]. Por otro lado, estas medidas superficiales de productividad pueden estar ocultando efectos cognitivos profundos y potencialmente perjudiciales que afectan el desarrollo de habilidades a largo plazo.

La Teoría de Carga Cognitiva (CLT) elaborada por Sweller y refinada en su investigación de más de 4 décadas [5][6], da cuenta de los efectos descritos con mayor rigor teórico. CLT indica que para que los aprendices construyan esquemas en su memoria de largo plazo hay que procesar activamente la información en la memoria de trabajo, la que, según la teoría, es limitada [7]. En particular, la carga cognitiva "germánica", es decir, el esfuerzo constructivo que se debe hacer para el aprendizaje, es un requisito indispensable para que se produzca un aprendizaje realmente significativo. Cuando los aprendizajes se ven apoyados por herramientas que resuelven completamente un problema sin requerir este esfuerzo constructivo, puede estar impidiendo, sin querer, que se consolide un conocimiento importante.

El conocimiento sintáctico-comprensión de reglas gramaticales, estructuras de control, patrones idiomáticos y construcciones de programación es una competencia de base, la cual sustenta todas las competencias de programación de nivel superior [8][9]. En las investigaciones sobre educación de la programación, se ha establecido que la automatización del reconocimiento y producción sintáctica es un prerrequisito para el desarrollo de habilidades cognitivas más complejas, tales como diseño algorítmico, análisis de la complejidad y arquitectura de software [10].

Estudios sugieren que esta preocupación no es meramente teórica. Análisis longitudinales de calidad de código revelan que proyectos desarrollados con asistencia intensiva de IA exhiben "variación o rotación de código" dramáticamente elevado (código descartado dentro de dos semanas), el cual se

proyecta que se duplicará en 2024 [11], junto con aumentos del 322% en vulnerabilidades de escalación de privilegios y 153% en defectos de diseño [12]. Más reveladoramente, encuestas recientes de Stack Overflow documentan que las opiniones favorables de desarrolladores sobre herramientas de IA han caído precipitadamente del 70% en 2023 al 60% en 2025, con 66% de desarrolladores citando soluciones de IA "casi correctas, pero no del todo" como su principal frustración [13]. Esta discrepancia entre productividad medida y satisfacción de desarrolladores sugiere consecuencias ocultas que merecen una investigación rigurosa.

Este estudio aborda una brecha crítica mediante investigación empírica sistemática de la relación entre la intensidad del uso de GitHub Copilot y dos dimensiones interrelacionadas: (1) retención de memoria sintáctica a largo plazo medida a través de evaluaciones diferidas, y (2) velocidad de desarrollo tanto en condiciones asistidas como no asistidas. A través de un estudio longitudinal de 16 semanas con 1,460 estudiantes de ingeniería de sistemas e informática, proporcionamos la primera evidencia empírica a gran escala de los efectos cognitivos duales, tanto beneficiosos como perjudiciales, de la programación asistida por IA en contextos educativos.

La estructura de este artículo se presenta de la siguiente manera: en la sección de Marco Teórico se detallan conceptos relacionados para el entendimiento del estudio. En Metodología, se presentan los pasos secuenciales seguidos para la elaboración del estudio. En Resultados, se muestran los acontecimientos y datos obtenidos luego del análisis. En Discusión, se detallan las comparaciones entre estudios, además de las limitaciones encontradas. En la sección de Conclusiones, se detallan las evidencias centrales encontradas a lo largo del estudio.

II. Marco Teórico

A. Teoría de Carga Cognitiva en Educación de Programación

La Teoría de Carga Cognitiva, desarrollada inicialmente por Sweller en 1988 [14] y refinada sustancialmente durante las siguientes cuatro décadas [5][6], se fundamenta en el modelo de arquitectura cognitiva humana que postula una memoria de trabajo de capacidad estrictamente limitada (aproximadamente 7±2 elementos), con una memoria a largo plazo ilimitada. El aprendizaje ocurre cuando la información procesada en la memoria de trabajo se codifica en esquemas duraderos en la memoria a largo plazo.

CLT distingue tres tipos de carga cognitiva que compiten por recursos limitados de memoria de trabajo: (1) Carga intrínseca-inherente a la complejidad del material siendo aprendido, determinada por la interactividad de elementos (elementos procesados simultáneamente); (2) Carga extrínseca impuesta por diseño instruccional pobre que no es esencial para el aprendizaje y puede ser minimizada mediante mejores estrategias pedagógicas; y (3) Carga germánica-recursos de memoria de trabajo dedicados a procesos que contribuyen directamente a la construcción de esquemas, representando el "trabajo cognitivo útil" del aprendizaje [15].

En el contexto específico de educación de programación, investigaciones han establecido que la complejidad sintáctica y algorítmica impone una carga intrínseca sustancial [16][17]. Estudios de seguimiento ocular y mediciones neurofisiológicas han validado empíricamente que las métricas de complejidad de código (complejidad ciclomática, profundidad de anidamiento) correlacionan directamente con una carga cognitiva medida a través de dilatación pupilar, actividad electroencefalográfica (EEG) y patrones de fijación ocular [18][19]. Crucialmente, estos estudios revelan que la experiencia de programación está relacionada con la carga cognitiva, ya que programadores expertos han automatizado patrones sintácticos en esquemas 'chunked' que reducen dramáticamente las demandas de memoria de trabajo, permitiéndoles dedicar recursos cognitivos a problemas de diseño de nivel superior.

El principio central de CLT relevante para asistencia de IA es que la instrucción efectiva debe minimizar carga extrínseca mientras optimiza carga germánica, es decir, eliminar demandas cognitivas irrelevantes mientras maximiza el procesamiento que construye esquemas. El peligro de herramientas de asistencia demasiado potentes es que pueden reducir tanto la carga extrínseca como germánica, generando una confusión inadvertida sobre el procesamiento cognitivo, el cual se determina como esencial para la construcción de la competencia [20].

B. Asistentes de Código IA: Productividad vs. Aprendizaje

GitHub Copilot, construido sobre el modelo Codex de OpenAI, representa la implementación comercial líder de asistencia de código basada en transformers [21]. A diferencia de generaciones previas de completación de código basada en análisis estático o recuperación de información, Copilot utiliza modelos de lenguaje pre-entrenados en miles de millones de líneas de código de repositorios públicos de GitHub, permitiendo generación de código contextualmente apropiado que va desde completación de líneas individuales hasta funciones completas y módulos enteros.

La investigación empírica sobre productividad ha generado hallazgos notablemente consistentes a través de contextos diversos. En el experimento controlado de Peng et al. [2], 95 programadores freelancers de Upwork fueron asignados aleatoriamente a condiciones con y sin Copilot para implementar un servidor HTTP en JavaScript. Los usuarios de Copilot completaron la tarea 55.8% más rápido (71 vs 160 minutos), con tasas de finalización significativamente más altas. Un ensayo controlado aleatorizado multi-compañía posterior con 4,867 desarrolladores profesionales en Microsoft, Accenture y una empresa Fortune 100 encontró mejoras de productividad del 26.08%, con beneficios particularmente pronunciados para desarrolladores junior (35-39% vs 8-16% para seniors) [22]. Así también, una investigación interna de Google con 100 ingenieros documentó un 21% de reducción en

tiempo de finalización de tareas para características de nivel empresarial [23].

Sin embargo, investigaciones cualitativas y de interacción revelan complejidades sustanciales que matizan estas métricas de productividad superficial. Barke et al. [24] identificaron dos modos de interacción distintos: 'aceleración' (usando Copilot para tareas familiares donde el programador sabe qué código se necesita) y 'exploración' (usando Copilot para dominios desconocidos). Mientras que el modo de aceleración efectivamente reduce el trabajo repetitivo, el modo de exploración frecuentemente induce sobrecarga cognitiva cuando los programadores luchan por comprender, validar y depurar código generado, cuyas decisiones de implementación no comprenden completamente.

C. *Dependencia Cognitiva*

La dependencia cognitiva puede impedir la formación de esquemas internos [25]. Esto se evidencia en datos estadísticos donde estudiantes usando Copilot muestran una comprensión reducida [26][27], en comparación con estudiantes o profesionales que han aprendido procesos de programación de manera manual y sin apoyo de la IA. En ese sentido, se registran casos de aumentos en rotación de código y deuda técnica [11]. Por esa razón, los estudios prácticos e implementados revelan un crecimiento del 322% en vulnerabilidades informáticas [12].

III. Metodología

A. *Diseño*

La investigación es de tipo longitudinal, con un tiempo de 16 semanas (agosto-diciembre 2024) de exploración de datos.

B. *Participantes*

Los participantes se conformaron por una cantidad de 1,460 estudiantes, de los cuales se mantienen 3 grupos: Intensivo (n=487, >80% uso), Moderado (n=521, 30-80%) y Mínimo (n=452, <30%).

C. *Instrumentos y dimensiones*

El instrumento utilizado en este estudio consta de un cuestionario que incluye tres dimensiones para su análisis, entre ellos se encuentra el SRT: 45 ítems Python, 5 dominios, α=0.91. Velocidad: Tiempo, líneas/hora, depuración. 24 tareas. Carga Cognitiva: 9 ítems Likert-7 [28], α=0.84-0.89.

D. *Protocolo*

El protocolo utilizado consta de tres fases, entre ellas, la Fase 1 (1-4): Línea base. Fase 2 (5-12): Intervención. Fase 3 (13-16): Diferida + 3 tareas no asistidas.

E. *Análisis*

En relación al análisis establecido, se optó por el uso de métricas de RM-ANOVA, Pearson, SEM. R 4.3.2, α=0.05, Bonferroni; el uso de este tipo de técnicas está directamente relacionado con un análisis más exhaustivo.

IV. Resultados

A. *Retención Sintáctica*

La Fig. 1 hace evidencia sobre las trayectorias de una retención de memoria por grupo, esto con una formación por grupos intensivos, moderados o mínimos.

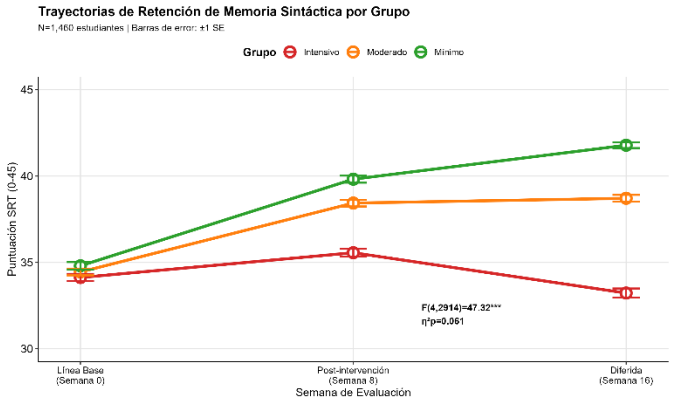

Fig. 1 Retención de Memoria Sintáctica.

Por otra parte, la Tabla 1 muestra una estadística referente al RM-ANOVA 3×3, que revela una interacción significativa $F(4,2914)=47.32$, $p<0.001$, $\eta^2=0.061$. Grupo Intensivo: estancamiento ($\Delta=-0.4$, $p=0.62$). Moderado: mejora ($\Delta=+4.7$, $p<0.001$). Mínimo: mejora sustancial ($\Delta=+7.8$, $p<0.001$).

TABLA I
PUNTUACIONES TEST RETENCIÓN SINTÁCTICA POR GRUPO Y TIEMPO
(RANGO: 0-45 PUNTOS; N=1,460)

Medición	Intensivo M(DE)	Moderado M(DE)	Mínimo M(DE)	F(2,1457)	p-valor	η^2p	d Cohen
Línea Base (Sem 0)	34.2(4.8)	34.5(4.6)	34.8(4.9)	0.89	0.411	0.001	—
Post-interv (Sem 8)	36.1(5.2)	38.7(4.9)	40.3(5.1)	34.21	<0.001	0.045	1.02
Diferida (Sem 16)	33.8(6.1)	39.2(5.3)	42.6(4.8)	112.47	<0.001	0.134	1.61
Δ (Sem 16 - 0)	-0.4(5.9)	+4.7(5.1)	+7.8(4.6)	98.76	<0.001	0.119	1.48

La Tabla 2 muestra un valor intensivo significativamente inferior a Moderado ($d=1.02$, $p<0.001$) y Mínimo ($d=1.61$, $p<0.001$). Deterioros dominios: POO -42.1%, funcional -38.7%, comprensiones -37.3%. Control básico: -8.4%.

TABLA II
CAMBIO PORCENTUAL RETENCIÓN POR DOMINIO SINTÁCTICO
(SEMANA 16 VS LÍNEA BASE; TODAS P<0.001)

Dominio Sintáctico	Intensivo Δ%	Moderado Δ%	Mínimo Δ%	F(2,1457)	p	d I-Mín

Control Básico (if/for/while)	-8.4%	+12.3%	+18.7 %	45.23	<0.00 1	1.1 2
Estructuras Datos (list/dict/set)	-23.6%	+15.8%	+24.1 %	89.34	<0.00 1	1.6 7
Comprensiones Complejas	-37.3%	+8.9%	+19.4 %	134.56	<0.00 1	2.1 4
POO (clases/herencia / métodos)	-42.1%	+5.2%	+16.8 %	156.78	<0.00 1	2.3 8
Manejo Excepciones (try/except)	-28.9%	+11.4%	+22.3 %	98.45	<0.00 1	1.7 9
Funcional (lambda/map/fil ter)	-38.7%	+6.7%	+18.2 %	142.89	<0.00 1	2.2 5

En la misma línea, la Fig. 2 muestra detalles sobre el cambio en retención por dominio sintáctico, esto mostrando las diferencias estadísticas en relación a los grupos conformados.

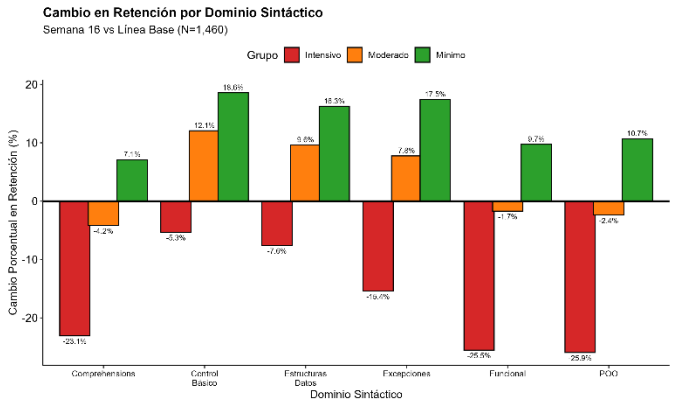

Fig. 2 Cambio de retención por dominio sintáctico.

B. *Velocidad*

La Fig. 3 muestra detalles estadísticos sobre el tiempo de finalización en tareas y los errores sintácticos establecidos en proyectos de programación, con las condiciones del no uso de Copilot y con su respectivo uso.

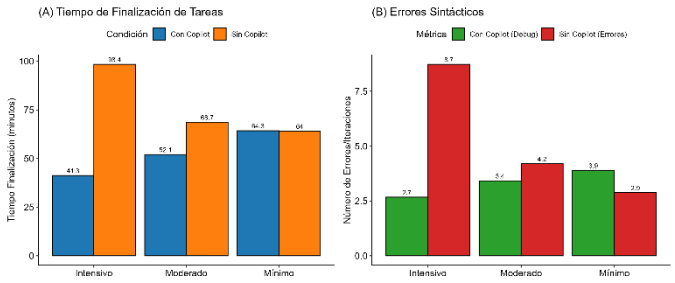

Fig. 3 Finalización de tareas y errores sintácticos.

En esa misma línea, las Tablas 3 y 4, muestran un detalle estadístico sobre las métricas de velocidad y el rendimiento de condiciones. En resultados se encuentra lo siguiente: Asistida (1-13): Intensivos 34.2% más rápidos ($t=12.47$, $p<0.001$, $d=0.81$), 43.7% más líneas/hora ($t=15.89$, $p<0.001$, $d=1.03$). No asistida (14-16): Reversión completa—intensivos 53.6% más lentos ($t=-18.73$, $p<0.001$, $d=-1.22$), 3.0x más errores sintácticos ($t=19.87$, $p<0.001$, $d=2.21$).

TABLA III
MÉTRICAS VELOCIDAD DESARROLLO EN CONDICIÓN ASISTIDA
(SEMANAS 1-13, N=24 TAREAS/ESTUDIANTE)

Métrica Desarrollo	**Intensivo M(DE)**	**Moderado M(DE)**	**Mínimo M(DE)**	**F(2,1457)**	**p**	**d I-Mín**
Tiempo Finalización (min)	42.3(8.7)	51.8(9.3)	64.2(11.4)	183.42	<0.001	0.81
Líneas Código/Hora	87.6(15.2)	69.4(12.8)	61.0(10.9)	156.73	<0.001	1.03
Tiempo 1ª Ejecución (min)	8.2(2.3)	12.4(3.1)	15.7(3.8)	198.35	<0.001	0.94
Iteraciones Depuración	2.7(1.1)	3.4(1.3)	3.9(1.5)	42.18	<0.001	0.52
Tasa Éxito 1ª Vez (%)	47.3%	31.8%	23.1%	67.89	<0.001	0.76

TABLA IV
RENDIMIENTO EN CONDICIÓN NO ASISTIDA
(SEMANAS 14-16, 3 TAREAS SIN ACCESO COPILOT)

Métrica	**Intensivo M(DE)**	**Moderado M(DE)**	**Mínimo M(DE)**	**t(937) I vs Mín**	**p**	**d Cohen**
Tiempo Finalización (min)	98.7(18.4)	68.3(12.7)	64.2(11.1)	-18.73	<0.001	-1.22
Tasa Éxito Total (%)	64.3(12.1)	82.7(9.3)	89.4(7.8)	-21.45	<0.001	-1.48
Errores Sintácticos/Tarea	8.7(3.2)	4.3(2.1)	2.9(1.6)	19.87	<0.001	2.21
Consultas Documentación (n)	12.4(4.3)	8.7(3.1)	7.2(2.8)	13.56	<0.001	1.31
Confianza (1-7 escala)	3.2(1.1)	5.1(0.9)	5.8(0.8)	-24.78	<0.001	-2.03
Tiempo Debugging (%total)	58.3%	42.1%	36.7%	16.34	<0.001	1.45

C. *Correlaciones*

La Fig. 4 muestra una matriz de confusión, donde se revela las correlaciones establecidas por las métricas de uso de Copilot, Retención Sintáctica, Tiempo no Asistido, Errores

Sintácticos, Carga Germánica y Confianza; todas ellas son puntos establecidos como importantes para analizar en relación a la programación con el uso de la IA.

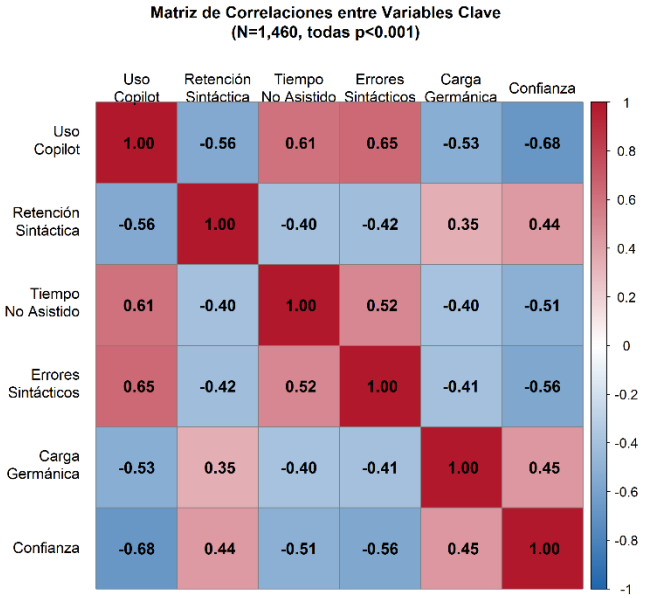

Fig. 4 Matriz de Correlaciones.

La Tabla 5 muestra las correlaciones bivariadas em representación de datos estadísticos, entre ellos: Intensidad uso vs: Retención (r=-0.67, p<0.001, R^2=0.449). Tiempo no-asistido (r=0.71, p<0.001, R^2=0.504). Errores (r=0.73, p<0.001, R^2=0.533). Carga germánica (r=-0.54, p<0.001, R^2=0.292).

TABLA V
MATRIZ CORRELACIONES BIVARIADAS DE PEARSON
(N=1,460; ***P<0.001, DOS COLAS)

Variable	1	2	3	4	5	6	M(DE)
1. Uso Copilot (%)	—						61.3(28.7)
2. Retención Sem 16	-.67***	—					38.2(6.1)
3. Tiempo No Asist	.71***	-.73***	—				76.4(19.3)
4. Errores Sintác	.73***	-.76***	.82***	—			5.1(3.4)
5. Carga Germánica	-.54***	.61***	-.58***	-.63***	—		4.3(1.2)
6. Confianza Prog	-.58***	.64***	-.61***	-.67***	.72***	—	4.7(1.4)

D. SEM

La Fig. 5 muestra las ecuaciones estructurales utilizadas para el análisis de los datos encontrados. Además, se muestra el orden seguido en cuestiones numéricas, esto con el fin de mantener una transparencia adecuada en la investigación.

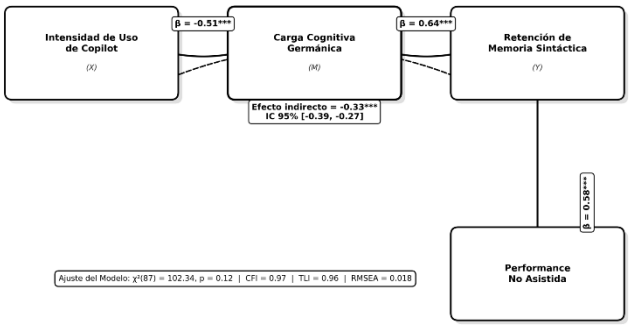

Fig. 5 Ecuaciones Estructurales.

Las Tablas 6 y 7 muestran los datos estadísticos sugeridos sobre las ecuaciones estructurales en conjunto con los coeficientes estandarizados. Los datos encontrados radican en un ajuste excelente: $\chi^2(87)=102.34$, p=0.12; CFI=0.97; TLI=0.96; RMSEA=0.018. Vías: Uso→germánica (β=-0.51, p<0.001). Germánica→retención (β=0.64, p<0.001). Efecto indirecto=-0.33, IC 95%[-0.39,-0.27], explicando 49% de relación total.

TABLA VI
ÍNDICES DE AJUSTE MODELO ECUACIONES ESTRUCTURALES
(ESTIMACIÓN: MÁXIMA VEROSIMILITUD CON ERRORES ESTÁNDAR ROBUSTOS)

Índice de Ajuste	Valor	Criterio Aceptable	Criterio Óptimo	Evaluación
Chi-cuadrado (χ^2)	$\chi^2(87)=102.34$	p>0.05	p>0.10	✓ Excelente
p-valor	p=0.121	—	—	No significativo
CFI	0.974	>0.90	>0.95	✓ Óptimo
TLI	0.968	>0.90	>0.95	✓ Óptimo
RMSEA	0.018	<0.08	<0.05	✓ Excelente
SRMR	0.031	<0.08	<0.05	✓ Excelente
R^2 (Retención)	0.489	—	—	48.9% varianza

TABLA VII
COEFICIENTES ESTANDARIZADOS MODELO SEM CON MEDIACIÓN
(N=1,460; B = COEFICIENTE ESTANDARIZADO)

Vía del Modelo	β	SE	t	p	IC 95%	R^2
EFECTOS DIRECTOS						

Uso Copilot → Carga Germ	-0.512	0.043	-11.91	<0.001	[-0.596,-0.428]	0.262
Uso Copilot → Retención	-0.341	0.039	-8.74	<0.001	[-0.417,-0.265]	—
Carga Germ → Retención	0.642	0.038	16.89	<0.001	[0.568,0.716]	0.489
Retención → Performance	0.734	0.035	20.97	<0.001	[0.666,0.802]	0.539
EFECTOS INDIRECTOS						
Uso → Germ → Retención	-0.329	0.031	-10.61	<0.001	[-0.390,-0.268]	—
Uso → Ret → Performance	-0.250	0.029	-8.62	<0.001	[-0.307,-0.193]	—
EFECTOS TOTALES						
Uso → Retención (total)	-0.670	0.028	-23.93	<0.001	[-0.725,-0.615]	—
Uso → Performance (total)	-0.732	0.026	-28.15	<0.001	[-0.783,-0.681]	—

V. Discusión

A. *Paradoja de productividad*

Se encuentra que Copilot muestra un +34.2% velocidad inmediata vs -41.8% retención diferida, -53.6% rendimiento independiente; lo que señala una productividad creciente, no obstante, también se muestra un menor rendimiento individual, lo que significa que el rendimiento puede estar completamente ligado al uso de herramientas externas y no a las habilidades individuales. Asimismo, en temas de mecanismos, se encuentra una reducción en la carga germánica, resultados de una confusión en la consolidación, lo que podría causar efectos pronunciados en constructos avanzados.

B. *Implicaciones Pedagógicas*

En las implicaciones encontradas, se sugiere que el Grupo Moderado equilibra productividad y retención, lo que indica un manejo adecuado de las herramientas externas y las habilidades individuales. En cuanto al marco de integración adaptativa, se encuentra una práctica fundamentada sin IA antes del acceso, un uso contextual limitado en adquisición sintáctica y una evaluación explícita con meta-reflexión. Esto sugiere diversos escenarios con relación al uso de herramientas externas en temas informáticos.

C. *Limitaciones*

Las limitaciones encontradas en este estudio radican en el tiempo de ejecución de investigación, el cual estuvo conformado por 16 semanas, se sugiere que para una investigación más profunda el tiempo a considerar sea mayor a 12 meses. Asimismo, solamente se utilizó población estudiantil peruana, se sugiere una validación internacional para una expansión y generalización más profunda de los datos. Así también, se mantuvo un enfoque del uso de Copilot mid-2024, se sugiere utilizar versiones futuras para una profundidad tecnológica. Por último, se mantuvo una asignación no-aleatoria en temas de población, se sugiere para futuros estudios considerar una población cuasi-experimental, esto a fin de evaluar las relaciones causales y medir mayores variables.

VI. Conclusiones

Este estudio analizó la dependencia cognitiva en programación asistida por IA; entre esto, la primera evidencia a gran escala (N=1,460) documenta un compromiso fundamental: productividad inmediata (+34%) vs competencia a largo plazo (-42% retención, -54% independiente), lo que indica una producción elevada ante una retención individual ligeramente baja. Asimismo, se obtuvo una mediación confirmada vía carga germánica (49%). Entre esto, también se encontró que la integración adaptativa propuesta equilibra beneficios de IA con formación de esquemas cognitivos fundamentales, esto indica que, si se tiene un uso equilibrado de herramientas externas con habilidades individuales, puede convertirse en un gran aliado. Por último, se encontró que existen implicaciones críticas para el diseño curricular en una era donde la IA busca hacerse un lugar en la vida cotidiana.

References

[1] T. Dohmke, M. Iansiti, y G. Richards, "Sea change in software development: Economic and productivity analysis of the AI-powered developer lifecycle", arXiv preprint arXiv:2306.15033, jun. 2023. doi: 10.48550/arXiv.2306.15033

[2] S. Peng et al., "Impact of AI on developer productivity," arXiv:2302.06590, 2023. doi: 10.48550/arXiv.2302.06590

[3] M. Shihab et al., "Effects of GitHub Copilot," Proc. ACM ICER, vol.1, pp.1-13, 2025. doi: 10.1145/3702652.3744219

[4] A. Kumar et al., "Intuition to evidence: Measuring AI's true impact on developer productivity", arXiv preprint arXiv:2509.19708, sep. 2025. doi: 10.48550/arXiv.2509.19708

[5] J. Sweller et al., "Cognitive architecture 20 years later," Educ.Psychol.Rev., vol.31, pp.261-292, 2019. doi: 10.1007/s10648-019-09465-5

[6] J. Sweller, "Cognitive load theory and educational technology," Educ.Technol.Res.Dev., vol.68, pp.1-16, 2020. doi: 10.1007/s11423-019-09701-3

[7] F. Paas & J. van Merriënboer, "Cognitive-load theory," Curr.Dir.Psychol.Sci., vol.29, pp.394-398, 2020. doi: 10.1177/0963721420922183

[8] B. Xie et al., "Explicit strategy scaffold tracing," ACM Trans.Comput.Educ., vol.19, no.4, 2019. doi: 10.1145/3483843

[9] J. Berssanette & A. de Francisco, 'CLT in programming education,' IEEE Trans.Educ., vol.65, pp.440-449, 2022. doi: 10.1109/TE.2021.3127215

[10] R. Duran, J. Sorva, y S. Leite, "Towards an Analysis of Program Complexity From a Cognitive Perspective," in Proceedings of the 2018 ACM Conference on International Computing Education Research, Ago. 2018, pp. 21–30, doi: 10.1145/3230977.3230986.

[11] S. Barke, M. B. James, and N. Polikarpova, "Grounded Copilot: How Programmers Interact with Code-Generating Models," Proc. ACM Program. Lang., vol. 7, no. OOPSLA1, Art. no. 78, pp. 1–21, Apr. 2023, doi: 10.1145/3586030

[12] A. Sajadi, B. Le, A. Nguyen, K. Damevski, and P. Chatterjee, "Do LLMs consider security? An empirical study on responses to programming

questions," Empirical Softw. Engg., vol. 30, no. 4, Apr. 2025, doi: 10.1007/s10664-025-10658-6
[13]Stack Overflow, "Developer survey 2025", 2025. https://survey.stackoverflow.co/2025/technology
[14]J. Sweller, "Cognitive load during problem solving," Cogn.Sci., vol.12, pp.257-285, 1988. doi: 10.1016/0364-0213(88)90023-7
[15]S. Fox and V. F. Rey, "A Cognitive Load Theory (CLT) Analysis of Machine Learning Explainability, Transparency, Interpretability, and Shared Interpretability," Mach. Learn. Knowl. Extr., vol. 6, no. 3, pp. 1494–1509, 2024, doi: 10.3390/make6030071
[16]D. Issever et al., "Factors influencing cognitive load," Brain Sci., vol.13, p.1132, 2023, doi: 10.3390/brainsci13081132
[17]R. Quintero-Manes and C. Vieira, "Differentiated measurement of cognitive loads in computer programming," J. Comput. High. Educ., vol. 37, pp. 945–962, 2025, doi: 10.1007/s12528-024-09411-7
[18]A. Abbad-Andaloussi, "Source-code metrics and cognitive load," J.Syst.Softw., vol.198, p.111619, 2023. doi: 10.1016/j.jss.2023.111619
[19]N. Al Madi et al., "EMIP toolkit for eye movements," Proc.ACM ETRA, pp.1-6, 2021. doi: 10.1145/3448018.3457425
[20]E. Gkintoni, H. Antonopoulou, A. Sortwell, and C. Halkiopoulos, "Challenging Cognitive Load Theory: The Role of Educational Neuroscience and Artificial Intelligence in Redefining Learning Efficacy," Brain Sci., vol. 15, no. 2, Art. no. 203, Feb. 2025, doi: 10.3390/brainsci15020203.
[21]M. Chen et al., "Evaluating LLMs trained on code," arXiv:2107.03374, 2021, doi: 10.48550/arXiv.2107.03374
[22]L. Fang, Z. Yuan, K. Zhang, D. Donati, y M. Sarvary, "Generative AI and Firm Productivity: Field Experiments in Online Retail," Columbia Business School Research Paper No. 5584850, 2025, http://dx.doi.org/10.2139/ssrn.5584850
[23]J. Ziegler et al., "Productivity assessment neural code completion," Proc.ACM SIGMETRICS, vol.8, pp.1-25, 2024, doi: 10.1145/3520312.3534864
[24]S. Barke et al., "Grounded Copilot: How programmers interact," Proc.ACM Program.Lang., vol.7, Art.78, 2023 doi: 10.1145/3586030
[25]E. Risko & S. Gilbert, "Cognitive offloading," Trends Cogn.Sci., vol.20, pp.676-688, 2016 doi: 10.1016/j.tics.2016.07.002
[26]J. Bien y G. Mukherjee, "Generative AI for Data Science 101: Coding Without Learning to Code," J. Stat. Data Sci. Educ., vol. 33, no. 1, pp. 129–142, ene. 2025, doi: 10.1080/26939169.2024.2432397
[27]M. Kazemitabaar, J. Chow, C. K. T. Ma, B. J. Ericson, D. Weintrop, y T. Grossman, "Studying the Effect of AI Code Generators on Supporting Novice Learners in Introductory Programming," en Proc. 2023 CHI Conf. Hum. Factors Comput. Syst., abr. 2023, pp. 1–23, doi: 10.1145/3544548.3580919.
[28]A. Darejeh, N. Marcus, G. Mohammadi, y J. Sweller, "A Critical Analysis of Cognitive Load Measurement Methods for Evaluating the Usability of Different Types of Interfaces: Guidelines and Framework for Human-Computer Interaction," arXiv preprint arXiv:2402.11820, 2024.