Aplicaciones financieras y actuariales de la inteligencia artificial generativa en el ámbito de la docencia

Autores: Miguel Escribano Ródenas mescri@emp.ucm.es
Fernando Ricote Gil fricote@ccee.ucm.es
Mercedes Elices López meelices@ucm.es
Juan José Escribano Ródenas jjescri@issi.uned.es

Departamento de Economía Financiera, Actuarial y Estadística
Universidad Complutense de Madrid
Departamento de Ingeniería de Software y Sistemas Informáticos
Universidad Nacional de Educación a Distancia

RESUMEN

Este trabajo centra el estudio en el uso de la Inteligencia Artificial Generativa (IAG) para la resolución de problemas en el ámbito de las matemáticas financieras y actuariales. La idea consiste en generar código que permita la resolución de varios problemas en estas áreas. Se hace un uso de la ingeniería de prompts (*prompt engineering*) como base metodológica que permite especificar el problema a partir de la especificación de requisitos del software que desea construir.

Se ha hecho una introducción estableciendo cómo funciona la IAG para establecer el contexto en el que se enmarca este trabajo. Se referencian diferentes trabajos en la literatura para fijar el ámbito de este estudio. En la metodología se muestra el enfoque aplicado para la generación un prompt que es la entrada a la IAG. A continuación se describen distintos ejemplos en el ámbito de las matemáticas financieras y actuariales, en los que se ilustra: el prompt utilizado, el código generado y la ejecución de dicho código para ilustrar cómo se obtiene una solución de cada ejemplo planteado.

Los resultados obtenidos para los ejemplos planteados permiten concluir que éstos se pueden abordar mediante esta técnica. Asimismo, este enfoque permite obtener un modelo con el que poder validar la solución presentada.

Esta técnica se presenta como un enfoque técnico y metodológico que permite resolver problemas de matemáticas financieras y actuariales con resultados muy interesantes.

ABSTRACT

This work focuses on the application of Generative Artificial Intelligence (GAI) to problem-solving within the domain of financial and actuarial mathematics. The central premise consists of leveraging AI-generated code as a computational instrument for addressing a range of problems in this field. Prompt engineering is adopted as the foundational methodological framework, enabling the precise specification of problems through the formulation of software requirements specification corresponding to the desired solution.

The study opens with an introduction that establishes the operational principles of GAI, thereby situating the present work within its broader theoretical and applied context. Relevant contributions from the existing literature are reviewed to delineate the scope of this investigation. The methodology section presents the systematic approach employed to construct prompts that serve as structured inputs to the GAI system. Subsequently, a series of illustrative examples drawn from financial and actuarial mathematics are examined, each documenting the prompt formulated, the code generated in response, and the execution of said code, thus demonstrating how a solution is derived for each problem instance considered.

The results obtained across the proposed examples substantiate the conclusion that such problems are amenable to resolution through this technique. Furthermore, this approach yields a computational model that can serve as a robust framework for validating the solutions presented.

This methodology is put forward as a rigorous technical and methodological paradigm that enables the effective resolution of problems in financial and actuarial mathematics, yielding results of considerable theoretical and practical significance.

***Palabras claves*:** Inteligencia artificial generativa, generación de código, ingeniería de prompts, ingeniería del software

***Área temática*:** Metodología y docencia.

1. INTRODUCCIÓN

La Inteligencia Artificial Generativa (IAG) constituye un paradigma avanzado dentro de las ciencias de la computación, enfocado en la arquitectura de modelos con la capacidad inherente de sintetizar datos inéditos. Esta síntesis se logra replicando con alta fidelidad las propiedades estadísticas y estructurales de los corpus de entrenamiento. En clara contraposición a los sistemas discriminativos tradicionales —cuya función empírica se circunscribe a la clasificación o predicción a partir de patrones preexistentes—, los modelos generativos trascienden estas limitaciones al producir contenido original y multidimensional. Esto abarca modalidades complejas como el lenguaje natural, imágenes de alta fidelidad, espectros auditivos, código fuente y topologías tridimensionales.

Desde un rigor técnico, la eficacia de arquitecturas de vanguardia, tales como las Redes Generativas Antagónicas (GANs), los modelos autorregresivos basados en transformadores (como ChatGPT) y los modelos probabilísticos de difusión, radica en su capacidad para modelar y asimilar la distribución estocástica subyacente de vastos volúmenes de datos. A partir de este aprendizaje profundo, estas redes logran inferir y generar muestras sintéticas que no solo son estadísticamente plausibles, sino también semánticamente coherentes dentro de su contexto operativo.

En el ámbito pragmático e investigativo, la aplicabilidad de la IAG es vasta y profundamente disruptiva. Su implementación abarca desde la automatización de la síntesis creativa y la simulación de escenarios macroeconómicos, hasta el diseño algorítmico avanzado, la personalización de servicios y la resolución asistida de tareas cognitivas de alta exigencia. Consecuentemente, la Inteligencia Artificial Generativa no debe concebirse meramente como una innovación de software, sino como un catalizador tecnológico transversal que está redefiniendo los paradigmas de la investigación científica y la competitividad en los sectores industriales.

Un modelo simplificado de cómo funciona la IAG se muestra a continuación:

1. [Usuario]

↓

2. [Prompt] → [Tokenización]

↓

3. [Modelo Generativo]

↓

4. [Postprocesamiento]

↓

5. [Respuesta generada]

El ciclo operativo de la Inteligencia Artificial Generativa (IAG) inicia con el usuario [1], quien proporciona una especificación textual detallada de su requisito, formalmente denominada prompt. Una vez introducida esta instrucción, el sistema ejecuta un proceso de tokenización [2], el cual descompone el lenguaje natural en unidades léxicas computacionalmente manejables. Posteriormente, esta entrada es procesada, normalizada y transformada en representaciones matemáticas que alimentan de forma directa al modelo generativo subyacente [3]. Entre las arquitecturas fundacionales más representativas en la actualidad destacan GPT (para el procesamiento de texto), Stable Diffusion y DALL-E (para la síntesis de imágenes), MusicLM y Jukebox (para la generación auditiva), así como CodeGen y Codex (especializados en la programación asistida). Finalmente, las inferencias del modelo atraviesan una capa algorítmica de posprocesamiento [4], cristalizando en la respuesta generada definitiva [5]. Es precisamente en la generación automatizada de código mediante IAG donde se inscribe el enfoque central de la presente investigación.

Para maximizar la eficacia empírica de la IAG, resulta imperativa la aplicación de la ingeniería de prompts. Esta disciplina emergente se aboca al diseño, formulación y optimización sistemática de las instrucciones lingüísticas dirigidas a los Modelos de Lenguaje de Gran Escala (LLMs, por sus siglas en inglés). Su objetivo primordial es guiar el comportamiento algorítmico para garantizar la precisión, coherencia, relevancia y utilidad del contenido sintetizado. Esta praxis exige una convergencia interdisciplinaria que integra principios de la lingüística computacional, la interacción humano-computadora y el aprendizaje automático (Machine Learning). Asimismo, demanda una comprensión analítica profunda de las arquitecturas generativas, reconociendo tanto sus capacidades inherentes como sus sesgos y limitaciones técnicas. En la práctica, su aplicabilidad es transversal, abarcando desde la resolución de tareas cognitivas generales hasta dominios altamente especializados, tales como el razonamiento lógico, la

traducción automática, la recuperación semántica de información y el desarrollo de software.

En este contexto teórico, Ashish, K. *et al.* (2025) postulan que un prompt riguroso debe adherirse a cinco atributos fundamentales: ser conciso, lógico, explícito, adaptativo y reflexivo.

Concisión y Explicitud: La instrucción debe formularse con estricta claridad, aislando los elementos nucleares de la tarea y delimitando con exactitud el formato, el contenido y el alcance esperado. Esto mitiga la ambigüedad y orienta a la inteligencia artificial hacia resultados precisos.

Estructura Lógica: Resulta indispensable articular una secuencia racional de ideas que provea al sistema del contexto necesario y establezca las interrelaciones conceptuales, promoviendo así una alta coherencia en las respuestas generadas.

Adaptabilidad y Reflexividad: El diseño del prompt requiere un enfoque heurístico y flexible, que permita la iteración sobre diversas configuraciones para hallar el equilibrio idóneo entre la directriz técnica y la creatividad del modelo. Finalmente, la evaluación empírica y constante de los resultados es crucial; capitalizar el análisis de interacciones previas constituye un mecanismo indispensable para el refinamiento progresivo y la mejora continua del rendimiento de la IAG.

2. ANÁLISIS DEL ESTADO DEL ARTE

El análisis del estado del arte se ha estructurado en torno a tres ejes temáticos fundamentales: (1) el desarrollo teórico y metodológico de la ingeniería de prompts, (2) la implementación de la Inteligencia Artificial Generativa (IAG) en la síntesis de código, y (3) la intersección de ambas tecnologías aplicada al ámbito de la educación y la docencia.

En lo concerniente a la ingeniería de prompts, la literatura reciente subraya su carácter fundacional. Anand, N. *et al.* (2025) articulan una guía exhaustiva orientada a los Modelos de Lenguaje de Gran Escala (LLMs), abarcando desde principios conceptuales básicos hasta técnicas de optimización avanzada, sin soslayar las implicaciones éticas inherentes a su uso. Por su parte, Nazir, A. *et al.* (2023) trazan la evolución de arquitecturas conversacionales como ChatGPT; aunque su enfoque es panorámico,

evidencian de manera empírica cómo el diseño y la estructuración de las instrucciones determinan el rendimiento y la viabilidad operativa de los modelos.

En el ámbito metodológico, Peres, R. *et al.* (2023) analizan la disrupción de la IAG en la investigación, postulando la ingeniería de prompts como una competencia alfabetizadora ineludible. En su estudio, delinean estrategias pedagógicas para la formulación efectiva de instrucciones y establecen criterios para la evaluación crítica de las inferencias sintéticas. En esta misma línea, Khalifa, M., *et al.* (2024) examinan el rol catalizador de la IAG en la productividad científica, destacando su capacidad para automatizar procesos sistemáticos y elevar el rigor en la redacción de manuscritos académicos. Finalmente, Gupta, A., *et al.* (2024) aportan una dimensión sociotécnica al investigar las dinámicas de transferencia de conocimiento sobre prompting en plataformas digitales, evidenciando el surgimiento de nuevas prácticas colaborativas y el desarrollo de una alfabetización crítica comunitaria.

Respecto al segundo eje, centrado en la generación automatizada de software, los esfuerzos académicos convergen hacia la validación y la seguridad algorítmica. Foggia, A., *et al.* (2024) proponen el marco metodológico ACCA, un sistema automatizado que integra técnicas de análisis estático y dinámico para auditar la corrección del código generado y mitigar vulnerabilidades, ofreciendo así una evaluación integral de su viabilidad en entornos críticos. Desde una perspectiva funcional, Liguori, P. *et al.* (2023) evalúan un espectro de métricas automáticas provenientes de la traducción neuronal, discutiendo su idoneidad y límites epistemológicos para cuantificar la equivalencia semántica y la operatividad del código resultante. A modo de transición hacia el ámbito formativo, Park, A. *et al.* (2025) investigan la integración de agentes conversacionales en la educación superior en ciencias de la computación, concluyendo que la asistencia sintáctica y las explicaciones algorítmicas que proveen estas herramientas reducen significativamente la carga cognitiva del estudiante.

En el tercer ámbito, referente a la aplicación de la IAG en la docencia, la evidencia empírica respalda su potencial transformador. Ramazan, Y. *et al.* (2024) demuestran que el soporte en tiempo real —mediante depuración asistida, sugerencias y síntesis automática— optimiza tanto la eficiencia como la calidad del aprendizaje en programación. No obstante, este escenario no está exento de tensiones académicas. Lim, W. *et al.* (2023) examinan las paradojas inherentes a la integración de la IAG en el

ecosistema educativo, proponiendo marcos conceptuales para reconciliar la autonomía intelectual del estudiante con la asistencia algorítmica.

Para dimensionar este fenómeno de manera global, Kizilcec, R. *et al.* (2024) presentan un estudio comparativo internacional que devela las divergencias perceptivas entre educadores y discentes respecto al impacto de la IAG en los mecanismos de evaluación formativa de la programación. Complementariamente, Kohnke, L. *et al.* (2023), si bien contextualizan su investigación en la enseñanza de lenguas extranjeras, ofrecen inferencias transversales y altamente extrapolables sobre la madurez tecnológica, las actitudes y la disposición del profesorado universitario para asimilar de manera efectiva la IAG en su praxis pedagógica.

A modo de corolario, el análisis del corpus bibliográfico revela de manera inequívoca que la ingeniería de prompts constituye el núcleo operativo ineludible para la generación efectiva y segura de código asistida por IAG. En el contexto académico, esta disciplina ha trascendido su naturaleza puramente instrumental para erigirse como un vector pedagógico central. Resulta imperativo destacar que, dentro de dominios técnicos como la Ingeniería de Software, el uso de modelos generativos y la parametrización lingüística de sus entradas ya no se conciben como simples recursos auxiliares, sino como competencias estructurales, críticas e indispensables para la formación de las futuras generaciones de desarrolladores.

3. METODOLOGÍA

El diseño metodológico propuesto para la consecución de los objetivos de la presente investigación se estructura en cuatro fases secuenciales y sistemáticas, orientadas a garantizar la trazabilidad, rigurosidad y validez en la generación automatizada de código mediante Inteligencia Artificial Generativa (IAG):

1. Especificación e Ingeniería de Prompts. Esta etapa inicial exige una especificación de requisitos exhaustiva, donde se delimita de forma granular la problemática computacional a resolver. Resulta imperativa la utilización de un vocabulario técnico riguroso y estrictamente acotado al dominio de aplicación; esta precisión semántica es fundamental para mitigar la entropía y evitar heurísticas ambiguas por parte del modelo. Asimismo, se debe definir explícitamente la naturaleza del

constructo de salida esperado, ya sea el desarrollo de algoritmos de cálculo, la parametrización de visualizaciones gráficas o la estructuración de modelos tabulares.

2. Análisis e interpretación del código sintético tras la inferencia del modelo. La IAG proporciona un artefacto de software fuente, comúnmente respaldado por documentación detallada e incrustaciones de comentarios algorítmicos. Durante esta fase heurística, se procede a una inspección estática y exégesis del código. Esta interpretación analítica es un paso crítico, puesto que permite descifrar la lógica operativa subyacente, anticipar el comportamiento sistémico en tiempo de ejecución y establecer una base sólida para futuros procesos de refactorización, escalado o depuración.

3. Despliegue y ejecución computacional La materialización de la lógica programada contempla tres arquitecturas operativas principales, seleccionables en función de las necesidades de abstracción y procesamiento:

Entornos de desarrollo integrado (IDE) locales: Arquitecturas clásicas que proveen herramientas avanzadas para la depuración y auditoría del software. Su uso exige la configuración previa y exhaustiva de todas las librerías y dependencias a las que invoca el código generado.

Infraestructuras de computación en la nube (*Cloud Computing*): Plataformas que elevan el nivel de abstracción de la infraestructura subyacente, delegando la gestión de recursos al proveedor y permitiendo focalizar los esfuerzos investigadores exclusivamente en la ejecución y rendimiento del modelo.

Arquitecturas híbridas: Modelos integrados que convergen el grado de control microscópico de un IDE local con la elasticidad, versatilidad y potencia de procesamiento de los entornos *cloud*.

4. Validación empírica y refinamiento: En esta etapa culminante, se evalúa empíricamente la viabilidad del software generado midiendo la exactitud, calidad y consistencia de sus resultados. Dicho proceso demanda un profundo conocimiento del dominio experto para certificar si la solución propuesta es coherente y realista frente a las dinámicas del problema original. Para garantizar el rigor científico, se diseñan e implementan pruebas de control (como tests unitarios) basadas en escenarios o problemas tipificados cuyas soluciones basales ya son conocidas. Esta contrastación permite certificar la validez de la herramienta o, en su defecto, retroalimentar el sistema para refinar iterativamente el modelo.

4. EJEMPLOS PRÁCTICOS DE GENERACIÓN DE CÓDIGO

En la presente sección se detalla la resolución empírica de una serie de problemas de naturaleza financiera y actuarial, abordados bajo un paradigma metodológico fundamentado en la Inteligencia Artificial Generativa (IAG). Para garantizar la reproducibilidad y el rigor científico en cada uno de los casos estudiados, se ha instrumentado un protocolo sistemático estructurado en las siguientes cinco fases operativas:

1. Diseño y especificación del *prompt*: Formulación de las directrices de entrada (ingeniería de *prompts*) para guiar el modelo generativo.

2. Inspección del código sintético: Observación estática y análisis preliminar de la estructura algorítmica generada.

3. Exégesis algorítmica: Revisión de la fundamentación lógica y descriptiva del código, a partir de la explicación provista por la propia herramienta de IAG.

4. Ejecución computacional: Despliegue y compilación del código resultante en un entorno de desarrollo integrado (IDE).

5. Validación y análisis de resultados: Evaluación crítica de las salidas de la aplicación y constatación empírica de su viabilidad frente al problema planteado.

La implementación técnica de los escenarios analizados se ha desarrollado empleando Python. La elección de este lenguaje de programación se fundamenta en su alta legibilidad, su vasto ecosistema de librerías científicas y la notable facilidad para la comprensión del código generado. No obstante, cabe subrayar que la arquitectura lógica de las soluciones inferidas posee un carácter agnóstico; por consiguiente, los resultados y metodologías son íntegramente extrapolables a otros lenguajes de programación de propósito general.

Para optimizar el rendimiento y la precisión de las respuestas, la formulación de los *prompts* se ha llevado a cabo en lengua española. Aunque cabría pensar que en lengua inglesa los resultados tendrían una mayor riqueza, no se ha encontrado ninguna limitación por el uso del castellano en la especificación del prompt en lo que se refiere a la calidad técnica de los resultados obtenidos.

Finalmente, en lo concerniente a la infraestructura de software, la validación empírica y la ejecución algorítmica se han instrumentado a través del entorno de

desarrollo Visual Studio Code (Microsoft). Por su parte, el núcleo computacional seleccionado para llevar a cabo las inferencias y la generación automatizada de código ha sido el modelo avanzado Gemini Pro, desarrollado por Google, dada su probada capacidad para el razonamiento lógico y la síntesis de software.

4.1. Construcción de una aplicación para el cálculo de operaciones de amortización y operaciones de constitución

Para ello se ha utilizado el siguiente prompt:

¿Puedes darme un código python que permita seleccionar entre operación de amortización y operación de constitución? En la operación de amortización, debe haber un editable que permita introducir el capital nominal o principal del préstamo en euros, y debe haber un selector que permita definir si la amortización es mediante el método francés (términos amortizativos constantes) o mediante el método de cuotas constantes (cuotas de amortización constantes). Una vez seleccionado el tipo de amortización debe haber un botón "CALCULAR PRÉSTAMO" que visualice el cuadro de amortización con todos los periodos y con las columnas: termino amortizativo, cuota de amortización, cuota de interés, capital amortizado y capital vivo. Debe haber un botón para visualizar de forma gráfica el capital vivo (deuda pendiente) periodo a periodo de forma continua entre periodos (cuando crece el capital vivo, antes de amortizar hay que establecer un barrido de puntos para graficarlo). Al poner el ratón sobre la gráfica es necesario que aparezca un tooltip con el valor graficado (abscisa y ordenada). En todas las graficas debe estar implementado el zooming y el panning, y debe poderse serializarse la imagen como SVG. Además debe implementarse la serialización de los datos fichero JSON y la importación desde formato JSON. En la operación de constitución debe haber un editable que permita introducir en euros el capital a constituir en euros y debe haber un selector que permita definir si la operación de constitución es de imposiciones constantes o de cuotas de constitución constantes. Una vez seleccionado el tipo de operación de constitución debe haber un botón "CALCULAR OPERACION DE CONSTITUCION" que visualice el cuadro de constitución con todos los periodos y con las columnas: imposiciones, cuota de interés, capital constituido, cuota de constitución y capital pendiente de constituir. Debe haber un botón para visualizar de forma gráfica el capital

constituido (deuda pendiente) periodo a periodo de forma continua entre periodos (cuando crece el capital constituido, después de cada imposición y hasta la siguiente imposición hay que establecer un barrido de puntos para graficarlo). Al poner el ratón sobre la gráfica es necesario que aparezca un tooltip con el valor graficado (abscisa y ordenada). En todas las gráficas debe estar implementado el zooming y el panning, y debe poderse grabar la imagen como SVG. Además debe implementarse la serialización de los datos fichero JSON y la importación desde formato JSON. Es necesario que se incluya como tipo de interés el tanto nominal y su fraccionamiento, junto con las unidades de los periodos de tiempo (mensuales, bimensuales, trimestrales, cuatrimestrales o semestrales), número de periodos de tiempo y el tipo de interés efectivo de ese periodo. La operación de constitución tiene que ser prepagable, es decir, las imposiciones se sitúan al comienzo de cada periodo. La serialización a JSON ha de llevarse a cabo tanto desde la pestaña de la operación de amortización como desde la pestaña de la operación de constitución. Necesito que incluyas la serialización a JSON y la carga desde JSON en ambas pestañas. También necesito que en la visualización se pueda exportar a SVG y se lleve a cabo mediante un botón.

La IAG proporciona el código y un fragmento del código se ve en el siguiente listado de la figura 1.

```
def calculate_loan(self):
        try:
            C0, im, n = float(self.loan_principal.text()), self.get_effective_rate(self.l_freq, self.l_type,
self.l_rate), int(self.l_n.text())
            method, data, Ck, total_Ak = self.loan_method.currentIndex(), [], C0, 0
            for k in range(1, n + 1):
                Ik = Ck * im
                if method == 0:
                    a = C0 * (im / (1 - (1 + im)**-n))
                    Ak = a - Ik
                else:
                    Ak = C0 / n
                    a = Ak + Ik
                total_Ak += Ak; Ck -= Ak
                data.append({"type": "loan", "period": k, "a": a, "Ak": Ak, "Ik": Ik, "total_A": total_Ak,
"Ck": max(0, Ck), "freq": self.l_freq.currentText(), "rate_used": im})
            self.current_data = data
            self.update_table(self.loan_table, data, ["a", "Ak", "Ik", "total_A", "Ck"])
        except Exception as e: QMessageBox.critical(self, "Error", str(e))
```

```
    def calculate_fund(self):
        try:
            Cn, im, n = float(self.fund_target.text()), self.get_effective_rate(self.f_freq, self.f_type,
self.f_rate), int(self.f_n.text())
            method, data, Sk = self.fund_method.currentIndex(), [], 0
            d_const = Cn / ((( (1+im)**n - 1 ) / im ) * (1 + im))
            for k in range(1, n + 1):
                if method == 0:
                    d = d_const
                    Ik = (Sk + d) * im
                    ck = d + Ik
                else:
                    ck = Cn / n
                    d = (ck - Sk * im) / (1 + im)
                    Ik = (Sk + d) * im
                Sk += ck
                data.append({"type": "fund", "period": k, "d": d, "Ik": Ik, "Sk": Sk, "ck": ck, "rem": max(0,
Cn - Sk), "freq": self.f_freq.currentText(), "rate_used": im})
            self.current_data = data
            self.update_table(self.fund_table, data, ["d", "Ik", "Sk", "ck", "rem"])
        except Exception as e: QMessageBox.critical(self, "Error", str(e))
```

Figura1. Fragmento del listado del programa obtenido.

En el listado de la figura 1, se muestra un fragmento del código en los que se calcula el cuadro de amortización del préstamo y el cuadro de constitución de la operación de constitución.

Cuando se ejecuta el código se muestra la interfaz que nos proporciona el programa, tal y como se ve en la figura 2.

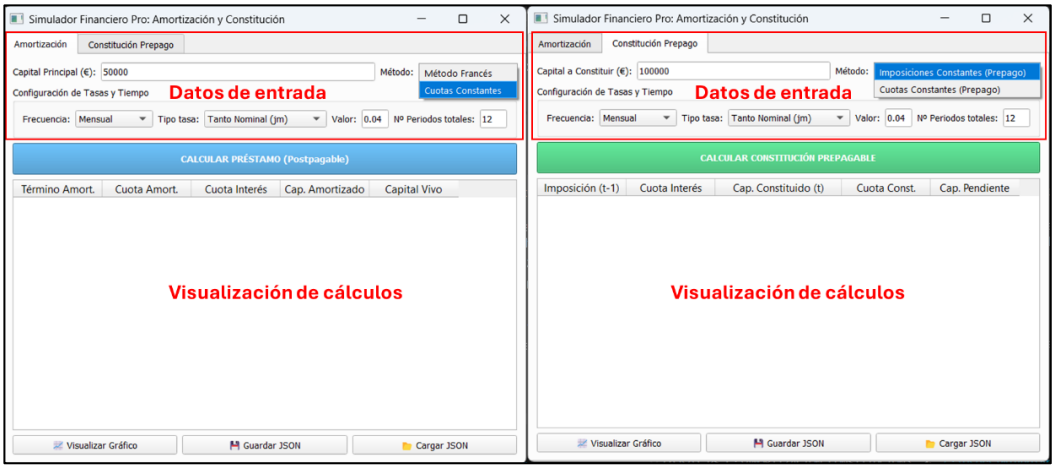


Figura 2. Interfaz generada que se visualiza para la solución del problema.

En la interfaz de la figura 2, se observa que hay una parte de datos financieros de entrada: capital a constituir, capital principal, método de amortización y de constitución, número de periodos totales e incluso si el tanto es nominal o efectivo. Por otro lado, hay una parte de visualización de los cálculos. La interfaz permite serializar los datos del préstamo/operación de constitución y la carga de los datos en formato JSON.

En la figura 3, se observa el cuadro de amortización del préstamo calculado y la evolución dinámica del capital vivo. Esta gráfica se puede exportar a formato SVG.

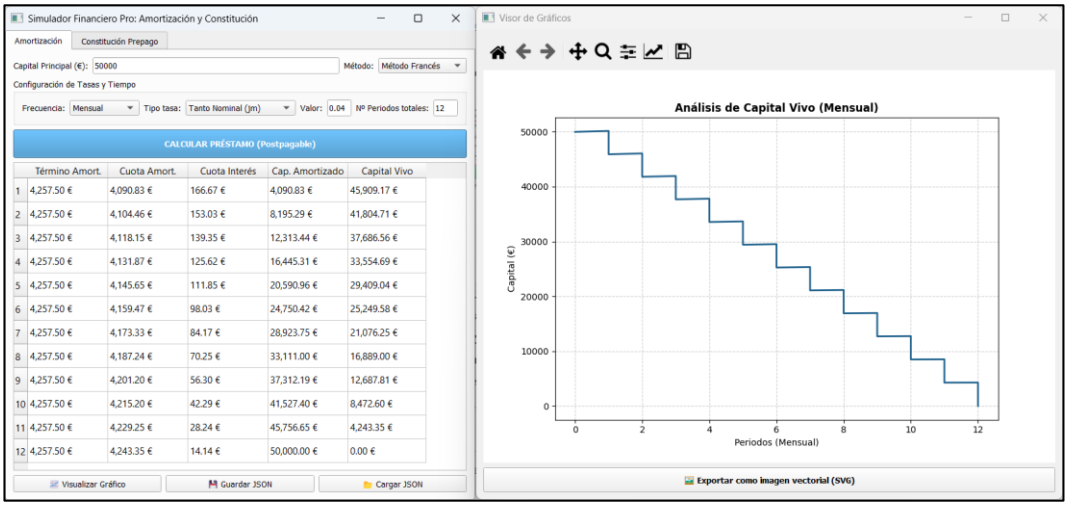


Figura 3. Ejecución del préstamo y visualización del capital vivo.

En la figura 4, se observa el cuadro de constitución de la operación de constitución calculado y la evolución dinámica del capital constituido. Esta gráfica se puede exportar a formato SVG.

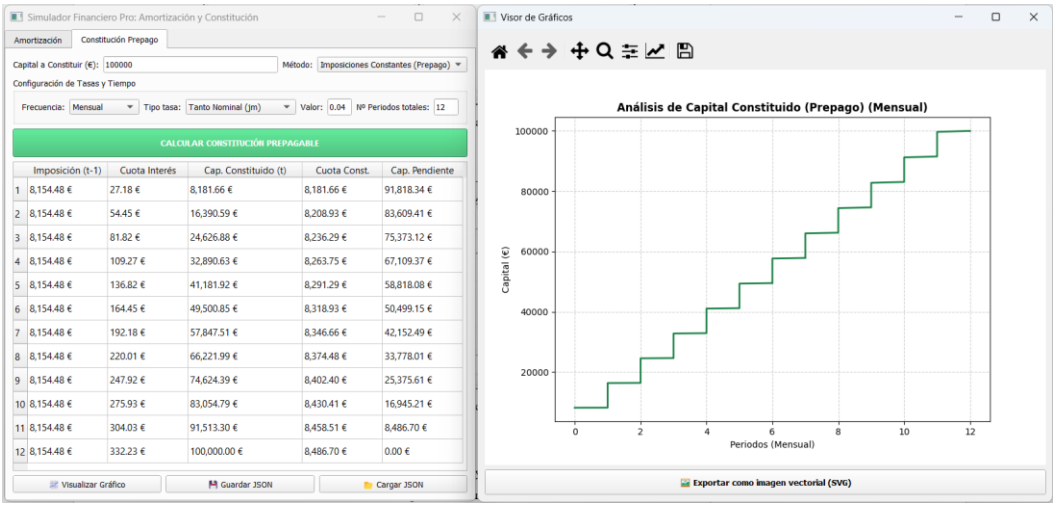


Figura 4. Ejecución de la operación de constitución y visualización del capital constituido.

4.3. Cálculo de las provisiones técnicas (reserva) mediante Chain-Ladder

Se trata del procedimiento del cálculo para el valor de las provisiones técnicas (reservas) para siniestros pendientes en una compañía de seguros. Mediante un cálculo más o menos complejo Gisler, A. *et al.* (2008), a partir de la información histórica recogida en forma de triángulos, consiste en estimar el valor que deberían tener las provisiones técnicas para hacer frente al valor de siniestros futuros.

El procedimiento que se muestra en este trabajo es el *Bayesian Chain Ladder (BCL)*, que se describe a continuación. Cuando ocurre un accidente (siniestro), la aseguradora no siempre lo paga de inmediato o de una sola vez. A veces, los pagos tardan meses o años en liquidarse, e incluso hay siniestros que ya ocurrieron pero que el cliente aún no ha reportado (esto se conoce como IBNR: *Incurred But Not Reported*). La aseguradora necesita saber cuánto dinero guardar hoy para hacer frente a esos pagos en el futuro.

El punto de partida es el triángulo de desarrollo. Este triángulo agrupa los siniestros por el año en que ocurrieron (filas) y rastrea cómo se van pagando a lo largo del tiempo (columnas). El objetivo matemático es "rellenar" la parte vacía del triángulo (el futuro).

El enfoque tradicional (determinista): El método Chain-Ladder clásico toma los datos del triángulo y calcula promedios ponderados para obtener un "factor de desarrollo" fijo. Por ejemplo, te dirá: *"Las reservas necesarias son exactamente 1.5 millones de euros"*. El problema es que no te dice qué tan seguro está de ese número.

El enfoque bayesiano (probabilístico): El BCL no te da un solo número, sino una distribución de probabilidades completa. Te dirá: *"Lo más probable es que necesites 1.5 millones de euros, pero hay un 5% de probabilidad de que necesites más de 2 millones de euros, y un 1% de probabilidad de que necesites 2.5 millones de euros"*.

El modelo se basa en el Teorema de Bayes, que actualiza nuestras creencias a medida que obtenemos nueva información. En el contexto actuarial, se compone de tres pilares:

1. La distribución a Priori (*Priors*): Antes de mirar los datos puros del triángulo, el actuario introduce su experiencia o el conocimiento de la industria en el modelo. Por

ejemplo, si es un seguro de autos, el actuario sabe que los pagos suelen estabilizarse rápido, y le da esa "pista" inicial al algoritmo mediante una distribución estadística.

2. La verosimilitud (*Likelihood*): Esta es la evidencia real. El modelo analiza los datos históricos observados en el triángulo de siniestros.

3. La distribución a Posteriori (*Posteriors*): El modelo combina la experiencia del actuario (*Priors*) con los datos reales (*Likelihood*) para generar miles de simulaciones usando algoritmos como el MCMC (*Markov Chain Monte Carlo*) que incluiremos en el código. El resultado es una curva de probabilidad que describe todos los posibles escenarios de cómo se comportarán los siniestros en el futuro.

El sector asegurador está fuertemente regulado (por normativas como Solvencia II en Europa), y las empresas no solo necesitan estimar un promedio, sino medir el riesgo de los peores escenarios:

- Cálculo del valor en riesgo (VaR): Permite a las aseguradoras calcular exactamente cuánto capital extra deben tener guardado para sobrevivir a un escenario extremo (por ejemplo, el percentil 99.5%).
- Cuantificación de la incertidumbre: En carteras de seguros nuevas o con pocos datos (donde el triángulo tiene muchos huecos), el método tradicional falla estrepitosamente. El BCL, al apoyarse en los *priors* (conocimiento experto), maneja la falta de datos de manera mucho más robusta.

En resumen, el BCL transforma una simple hoja de cálculo matemática en un simulador de riesgos sofisticado, dándole al actuario un mapa completo de todas las posibles realidades financieras de la empresa.

Para ver esta técnica funcionando se ha utilizado el siguiente prompt:

¿Puedes darme un código python para la aplicación del modelo Bayesian Chain-Ladder mediante el uso de triángulos de desarrollo,
con una implementación en Python y con adaptación a entrada por: años, meses,…
y Excel multihoja? Se necesita la curva de probabilidad final para calcular el percentil 95% de las reservas estimadas. Es necesaria una distribución predictiva completa, calculando cuantiles y el Valor en Riesgo (VaR)

Se muestra en la figura 5, un fragmento del código generado para el ajuste del modelo bayesiano.

```python
def fit_bayesian_model(self, sheet_name, tune=2000, draws=2000):
        """
        Aplica un modelo BCL utilizando PyMC.
        """
        if sheet_name not in self.triangles:
            print(f"La hoja {sheet_name} no existe.")
            return
        triangle = self.triangles[sheet_name]
        y_obs, dev_idx = self.extract_link_ratios(triangle)
        num_dev_periods = triangle.shape[1] - 1
        print(f"\nAjustando modelo BCL para la hoja '{sheet_name}'...")
        with pm.Model() as bcl_model:
            # 1. Priors: TruncatedNormal para asegurar factores > 0
            mu_f = pm.TruncatedNormal('mu_f', mu=1.5, sigma=1.0, lower=0.0, shape=num_dev_periods)
            sigma_f = pm.HalfNormal('sigma_f', sigma=0.2, shape=num_dev_periods)
            # 2. Likelihood
            mu_obs = mu_f[dev_idx]
            sigma_obs = sigma_f[dev_idx]
            likelihood = pm.Normal('obs', mu=mu_obs, sigma=sigma_obs, observed=y_obs)
            # 3. Inferencia
            self.trace = pm.sample(
                draws=draws,
                tune=tune,
                target_accept=0.95,
                return_inferencedata=True,
                progressbar=False # Oculto la barra para salida limpia en consola
            )
        print("\n¡Muestreo MCMC completado!")
        summary = az.summary(self.trace, var_names=['mu_f'])
        print("\nResumen de los Factores de Desarrollo Bayesianos (mu_f):")
        print(summary)
        return self.trace
```

Figura 5. Fragmento de código para el ajuste del modelo bayesiano

En la figura 6, se muestra un fragmento de código para el cálculo de las reservas.

```python
def calculate_reserves(self, sheet_name):
        """
        Calcula la distribución de las reservas (IBNR) totales usando
        las muestras de la distribución a posteriori y extrae los percentiles.
        """
        if self.trace is None:
            print("Debes ajustar el modelo primero llamando a fit_bayesian_model().")
            return

        triangle = self.triangles[sheet_name]
        I, J = triangle.shape
```

```
        # Extraer las miles de simulaciones generadas para los factores
        # Forma original: (cadenas, muestras, periodos). Lo aplanamos a (total_muestras, periodos)
        mu_f_samples = self.trace.posterior['mu_f'].values.reshape(-1, J - 1)
        num_samples = mu_f_samples.shape[0]
        # Array para guardar la reserva total de cada simulación
        total_reserve_samples = np.zeros(num_samples)
        for i in range(I):
            # Encontrar la última columna con datos reales para el año de origen 'i'
            valid_indices = np.where(~np.isnan(triangle[i, :]))[0]
            if len(valid_indices) > 0:
                last_idx = valid_indices[-1]
                latest_observed = triangle[i, last_idx]

                # Si el año aún no ha llegado a su último periodo de desarrollo
                if last_idx < J - 1:
                    # Extraer los factores de desarrollo futuros necesarios para este año
                    future_factors = mu_f_samples[:, last_idx:]

                    # Calcular el Coste Último (Ultimate) multiplicando el último valor observado
                    # por el producto de todos los factores de desarrollo futuros
                    ultimate_samples = latest_observed * np.prod(future_factors, axis=1)

                    # La reserva IBNR es el Coste Último menos lo que ya hemos observado
                    ibnr_samples = ultimate_samples - latest_observed

                    # Sumar esta reserva a la reserva total de la aseguradora
                    total_reserve_samples += ibnr_samples
        # Calcular los estadísticos clave sobre la distribución final de reservas
        mean_reserve = np.mean(total_reserve_samples)
        p05 = np.percentile(total_reserve_samples, 5)
        p50 = np.percentile(total_reserve_samples, 50)
        p95 = np.percentile(total_reserve_samples, 95) # El VaR al 95%
        print("\n" + "="*50)
        print(" DISTRIBUCIÓN PREDICTIVA DE RESERVAS (IBNR) TOTALES ")
        print("="*50)
        print(f"Reserva Promedio Estimada:      {mean_reserve:,.2f}")
        print(f"Mediana (Percentil 50%):        {p50:,.2f}")
        print(f"Reserva al 95% de Certeza (VaR): {p95:,.2f}")
        print(f"Intervalo de Confianza 90%:     [{p05:,.2f} - {p95:,.2f}]")
        print("="*50)
        return total_reserve_samples
```

Figura 6. Fragmento de código para cálculo de la reserva

Para observar cómo funciona la aplicación vamos a ver cómo realiza los cálculos con un triángulo como el que se encuentra en la tabla 1.

Periodo_Origen	1	2	3	4
2019	1000	1500	1750	1800
2020	1100	1600	1900	
2021	1050	1580		
2022	1200			

Tabla1. Datos empleados en la ejecución del programa.

Con estos datos de entrada el programa devuelve la tabla 2 por la consola.

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
mu_f[0]	1.487	0.050	1.400	1.578	0.002	0.005	1195.0	914.0	1.00
mu_f[1]	1.174	0.088	1.015	1.341	0.004	0.010	1323.0	607.0	1.01
mu_f[2]	1.046	0.179	0.706	1.416	0.004	0.005	2150.0	1891.0	1.00

Tabla 2. Resumen de factores de desarrollo bayesianos (mu_f)

mu_f[0], mu_f[1], mu_f[2]: Representan los "saltos" (factores de desarrollo) de un periodo de desarrollo al siguiente.

- mu_f[0] es el factor para pasar del periodo 1 al 2.
- mu_f[1] es del periodo 2 al 3.
- mu_f[2] es del periodo 3 al 4

mean (media): Es el valor promedio estimado para el factor. Por ejemplo, para mu_f[0], el modelo estima que los siniestros se multiplicarán en promedio por 1.487 (un incremento del 48.7%). Este sería el equivalente al factor que se usaría en un análisis *Chain-Ladder* clásico.

sd (desviación estándar): Mide cuánta incertidumbre o variación hay alrededor de esa media. Un dato interesante interesante: la *sd* de mu_f[0] es pequeña (0.050), pero la de mu_f[2] es grande (0.179). Porque para el primer periodo hay muchos datos históricos en el triángulo, pero para el tercer periodo tienes muy pocos, así que el modelo está menos seguro.

hdi_3% y hdi_97% (*Highest Density Interval*): Este es el equivalente bayesiano al intervalo de confianza. Juntos, forman un intervalo del 94% de probabilidad.

Para mu_f[0], el modelo nos está diciendo: "Estoy un 94% seguro de que el verdadero factor de desarrollo está entre 1.400 y 1.578".

Nótese que cómo para mu_f[2] el intervalo es muy grande (de 0.706 a 1.416). El modelo admite que, por la falta de datos en la "cola" del triángulo, el factor podría incluso ser menor a 1 (lo que implicaría recuperaciones o ajustes a la baja en las reservas).

mcse_mean y mcse_sd (*Monte Carlo standard error and standard deviation*): Indican el error introducido específicamente por el hecho de usar simulaciones en lugar

de una fórmula matemática exacta. Como ambos valores son muy cercanos a 0 (ej. 0.002), significa que el número de simulaciones que se han realizado ha sido suficiente.

ess_bulk y ess_tail (*Effective Sample Size*)*:* Hay valores como 1195.0 o 2150.0. Esto significa que de los miles de muestras, hay un número altísimo de muestras verdaderamente independientes y útiles, tanto en el centro de la distribución (*bulk*) como en los extremos/peores escenarios (*tail*).

*r_hat (*R de *Gelman-Rubin):* Es la prueba de fuego de la convergencia. Compara las 4 "cadenas" (simulaciones paralelas) que corrió el modelo. Si todas llegaron a las mismas conclusiones estadísticas, el valor es 1.00. La regla de oro es que el *r_hat* debe ser menor o igual que 1.01. Como tenemos 1.00 y 1.01, significa que el modelo convergió perfectamente.

Finalmente el programa obtiene la distribución predictiva de reservas (IBNR) totales que se muestran en la tabla 3.

Reserva Promedio Estimada:	1,434.83
Mediana (Percentil 50%):	1,369.29
Reserva al 95% de Certeza (VaR):	3,281.06
Intervalo de Confianza 90%:	[-277.84 - 3,281.06]

Tabla 3. Distribución predictiva de reservas

Reserva Promedio Estimada. Significa que en promedio, las simulaciones indican que la aseguradora necesita guardar 1,434.83 para hacer frente a los siniestros futuros que aún no se han pagado o reportado (IBNR). Este número suele ser muy parecido al que se obtendría con el método tradicional de *Chain-Ladder* determinista.

Mediana (Percentil 50%). El promedio (1,434) es mayor que la mediana (1,369). En estadística, esto significa que la distribución está "sesgada a la derecha" (tiene una cola larga). Es un comportamiento clásico y muy real en los seguros: la mayoría de las veces los siniestros son normales y controlables (lo que tira de la mediana hacia abajo), pero hay unos pocos escenarios catastróficos que cuestan muchísimo dinero (lo que empuja el promedio hacia arriba). La mediana nos dice que en el **50% de los escenarios simulados, la reserva necesaria fue menor a 1,369.29**, y en el otro 50% fue mayor.

Reserva al 95% de Certeza (VaR): Este es el número más importante de toda la tabla para la gestión de riesgos y normativas como Solvencia II. **Representa el valor en riesgo (VaR). Este valor nos está diciendo:** *"Si guardamos en el banco 3,281.06, se tiene un 95% de probabilidad de tener dinero suficiente para pagar todos los siniestros*

futuros sin quebrar". El promedio sugería guardar ~1,434, pero para estar realmente seguro (cubriendo el 95% de los peores escenarios posibles), hay que guardar más del doble. Esto refleja la alta incertidumbre (volatilidad) que el modelo detectó en los periodos de desarrollo más tardíos.

Intervalo de Confianza 90%: Este intervalo elimina el 5% de los escenarios más optimistas y el 5% de los más pesimistas, quedándose con el 90% central de la realidad posible. El límite superior coincide con el VaR al 95% (**3,281.06**). Hay un número negativo. En los mejores escenarios posibles (el 5% inferior), el modelo BCL está estimando que el coste final (*ultimate*) será menor que lo que ya se ha pagado o provisionado. Matemáticamente, la reserva IBNR es el coste final menos lo que ya se ha pagado; si el coste final disminuye, el IBNR se vuelve negativo. En el mundo real, esto ocurre por **recuperaciones**: cuando la aseguradora vende los restos de un coche siniestrado (salvamentos), cuando recobra dinero demandando a un tercero culpable (subrogación), o cuando se cancelan reservas que estaban sobreestimadas.

5. CONCLUSIONES DEL TRABAJO REALIZADO

La Inteligencia Artificial Generativa (IAG) se erige en la presente investigación como un recurso metodológico altamente eficaz y transformador para el abordaje y la resolución de problemáticas complejas en los ámbitos financiero y actuarial.

El análisis empírico ha validado la aplicabilidad de la IAG mediante la resolución de escenarios aplicados al dominio financiero y actuarial. La arquitectura metodológica propuesta demuestra que la convergencia secuencial entre la ingeniería de prompts y el uso de entornos de desarrollo integrados (IDE) viabiliza la síntesis automatizada de software funcional, capaz de dar respuesta precisa a los problemas planteados.

La eficacia de las soluciones computacionales obtenidas está intrínsecamente subordinada a la especificación rigurosa y exhaustiva de las instrucciones lingüísticas (prompts). Por ello, cada directriz de entrada ha sido meticulosamente diseñada y contextualizada para modelar con exactitud las variables y restricciones de cada caso de estudio.

La etapa de despliegue computacional y la posterior validación empírica se han ejecutado de manera sistemática a través de IDEs, consolidando a estas plataformas como

instrumentos indispensables para la compilación, certificación y aseguramiento de la calidad de los resultados algorítmicos generados.

Bajo este nuevo paradigma algorítmico, el núcleo del esfuerzo investigador experimenta un desplazamiento estratégico. La carga operativa transita desde la codificación manual tradicional hacia el diseño arquitectónico del prompt y la posterior inspección, depuración analítica y optimización del código dentro del entorno de desarrollo.

Finalmente, se concluye que la explotación efectiva y segura de la tecnología IAG demanda una convergencia de saberes. Resulta imperativo integrar sólidas competencias en ingeniería de software —específicamente en el manejo de IDEs y la comprensión lectora de lenguajes de programación— con un profundo conocimiento experto del dominio de aplicación. Este dominio temático es el criterio definitivo y necesario para auditar el rigor y validar la pertinencia de las soluciones inferidas por el modelo.

6. REFERENCIAS BIBLIOGRÁFICAS

- ASHISH, K., NISUMBA, S., RAMKUMAR, R. (2025) “Enhancing data analysis and programming skills through structured prompt training: The impact of generative AI in engineering education”. Computers and Education: Artificial Intelligence: Vol 8.
- FOGGIA, A., IMPROTA, C., LIGUORI P. (2024) “Automating the Correctness Assessment of AI-Generated Code for Security Purposes”. Journal of Systems and Software. Volumen 216.
- NAZIR, A., WANG, Z. (2023) “A comprehensive survey of ChatGPT: Advancements, applications, prospects, and challenges”. Meta-Radiology. Volume 1.
- PARK, A., TAEKYUNG, K. (2025) “Code suggestions and explanations in programming learning: Use of ChatGPT and performance”. The International Journal of Management Education. Volume 23, Issue 2.
- LIGUORI, P., IMPROTA, C., NATELLA, R., CUKIC B (2023). “Who evaluates the evaluators? On automatic metrics for assessing AI-based offensive code generators”. Expert Systems with Applications. Volume 225.

- PERES, R., SCHREIR, M., SWEIDEL, D., SORESCU, A. “On ChatGPT and beyond: How generative artificial intelligence may affect research, teaching, and practice”. International Journal of Research in Marketing. Volume 40. Issue 2
- RAMAZAN, Y., FATMA G. (2024). “The effect of generative artificial intelligence (AI)-based tool use on students' computational thinking skills, programming self-efficacy and motivation”. Computers and Education: Artificial Intelligence.Volume 4
- LIM, W., GUNASEKARA, A.,PALLANT, J.L., PALLANT, J.I., PACHENKINA, E. (2023). “Generative AI and the future of education: Ragnarok or reformation? A paradoxical perspective from management educators”. International Journal of Management Education. Volume 21. Issue 2
- KIZILCEC, R., HUBER, E. y otros (2024). “Perceived impact of generative AI on assessments: Com”paring educator and student perspectives in Australia, Cyprus, and the United States. Computers and Education: Artificial Intelligence. Volume 7
- KOHNKE, L., MOORHOUSE, B., ZOU, D. (2023) Exploring generative artificial intelligence preparedness among university language instructors: A case study. Computers and Education: Artificial Intelligence. Volume 5.
- ANAND, N., AJANTHA, D., KULDEEP, K. (2025). “Mastering Prompt Engineering. Deep Insights for Optimizing Large Language Models (LLMs)”. Elsevier. ISBN: 9780443339042
- KHALIFA, M.,ALBADAWY, M. (2024).” Using artificial intelligence in academic writing and research: An essential productivity tool”. Computer Methods and Programs in Biomedicine Update. Volume 5.
- GUPTA, A., SHIVERS-MCNAIR, A.(2024) “Wayfinding” through the AI wilderness: Mapping rhetorics of ChatGPT prompt writing on X (formerly Twitter) to promote critical AI literacies”. Computers and Composition. Volume 74.
- GISLER, A., y WÜTHRICH, M. V. (2008). Credibility for the chain ladder reserving method. ASTIN Bulletin: The Journal of the IAA, 38(2), 565–600.