

# INTEGRACIÓN DE UN ASISTENTE DE INTELIGENCIA ARTIFICIAL EN SOFTWARE DE ANÁLISIS Y DISEÑO ESTRUCTURAL BASADO EN EL PROTOCOLO MCP

Orellana, René Gonzalo<sup>a</sup> y Durán, Mario Alberto<sup>b</sup>

Consultores en Ingeniería Estructural

<sup>a</sup> gonchio@gmail.com    <sup>b</sup> marioa.duranm@gmail.com

**Palabras clave:** inteligencia artificial, análisis y diseño asistido, MCP, ejecución de código, automatización estructural.

## RESUMEN

Este trabajo presenta el desarrollo e integración de un asistente de inteligencia artificial (IA) en software de análisis y diseño estructural por elementos finitos. El objetivo es permitir la interacción del ingeniero con el programa mediante lenguaje natural, automatizando tareas habituales de modelado, análisis y diseño. El asistente se apoya en el protocolo abierto Model Context Protocol (MCP), que estandariza la conexión entre los modelos de lenguaje y las aplicaciones externas. Para ello se implementa un servidor MCP embebido que expone una API Python en un entorno restringido (sandbox): la IA escribe y ejecuta scripts mediante una única herramienta («code mode»), preservando la integridad y seguridad de los datos del proyecto. Mediante instrucciones en lenguaje natural, el usuario genera geometría, asigna secciones y materiales, define cargas, ejecuta análisis, diseña elementos estructurales y extrae resultados. El patrón se valida con dos implementaciones reales sobre programas comerciales distintos. Entre los principales beneficios se destacan la reducción del tiempo dedicado a tareas repetitivas, la disminución de errores y una mayor accesibilidad para usuarios menos experimentados. Como desafíos se identifican la validación de los resultados generados por IA y las limitaciones del lenguaje natural en contextos técnicos especializados.

## ABSTRACT

*This paper presents the development and integration of an artificial intelligence (AI) assistant into finite element-based structural analysis and design software. The objective is to enable engineers to interact with the program through natural language, automating common tasks in structural modeling, analysis, and design. The assistant relies on the open Model Context Protocol (MCP), which standardizes the connection between language models and external applications. To this end, an embedded MCP server exposes a Python API in a restricted sandbox: the AI writes and runs scripts through a single tool («code mode»), preserving data integrity and project security. Through natural language instructions, users generate geometry, assign sections and materials, define loads, run analyses, design structural members, and extract results. The pattern is validated with two real implementations on different commercial programs. The main benefits include reduced time spent on repetitive tasks, a lower incidence of errors, and improved accessibility for less experienced users. Identified challenges include the validation of AI-generated results and the limitations of natural language in highly specialized technical contexts.*

Organiza:

## INTRODUCCIÓN

La irrupción de los modelos de lenguaje de gran escala (LLM, por sus siglas en inglés) ha transformado la manera en que los profesionales interactúan con las herramientas informáticas. En el ámbito de la ingeniería estructural, buena parte del esfuerzo de modelado reside en tareas repetitivas y propensas a error —ingresar geometría, asignar secciones y materiales, definir cientos de casos y combinaciones de carga, lanzar análisis y, finalmente, extraer y tabular resultados— que, aunque mecánicas, consumen un tiempo considerable y requieren familiaridad con la interfaz del programa.

La mayoría de los programas de cálculo ofrece una interfaz de programación de aplicaciones (API) que permite automatizar dichas tareas. Sin embargo, estas API exigen conocimientos de programación que no todos los ingenieros poseen y, en la práctica, quedan reservadas a usuarios avanzados. La hipótesis de este trabajo es que un asistente de IA puede actuar como intermediario entre el lenguaje natural del ingeniero y la API del programa, democratizando el acceso a la automatización.

El aporte central de este artículo no es un producto concreto, sino una arquitectura de referencia replicable: un patrón de integración aplicable a cualquier programa de ingeniería que disponga —o pueda disponer— de una API de scripting. Dicha arquitectura se apoya en dos pilares. El primero es el Model Context Protocol (MCP), un protocolo abierto que estandariza la conexión entre los modelos de lenguaje y las aplicaciones externas. El segundo, y quizás el más determinante, es el paradigma de la ejecución de código —conocido como «code mode»—, que consiste en dejar que la IA escriba scripts en un lenguaje interpretado y los ejecute contra el modelo, en lugar de exponer una herramienta distinta por cada función de la API.

Este trabajo se plantea como un reporte de experiencia: su aporte no es un estudio comparativo de la literatura, sino una arquitectura de referencia derivada de la implementación real sobre dos programas comerciales.

Para demostrar que el patrón es genuinamente replicable, se presentan dos implementaciones reales construidas sobre tecnologías de base radicalmente distintas. El artículo comienza por definir la terminología para un lector no programador y luego describe la arquitectura de referencia y el paradigma de la ejecución de código sobre el que se sustenta. A continuación se analiza cómo el asistente conoce la API, se detalla el flujo de una interacción y se abordan la seguridad y la privacidad. Por último, se presentan las dos implementaciones de referencia junto con ejemplos de uso concretos, se comparte la experiencia de desarrollo y se resumen los beneficios, los desafíos y las conclusiones.

## CONCEPTOS Y TERMINOLOGÍA

Dado que el público destinatario son ingenieros y no necesariamente programadores, se definen a continuación, en términos accesibles, los conceptos informáticos empleados.

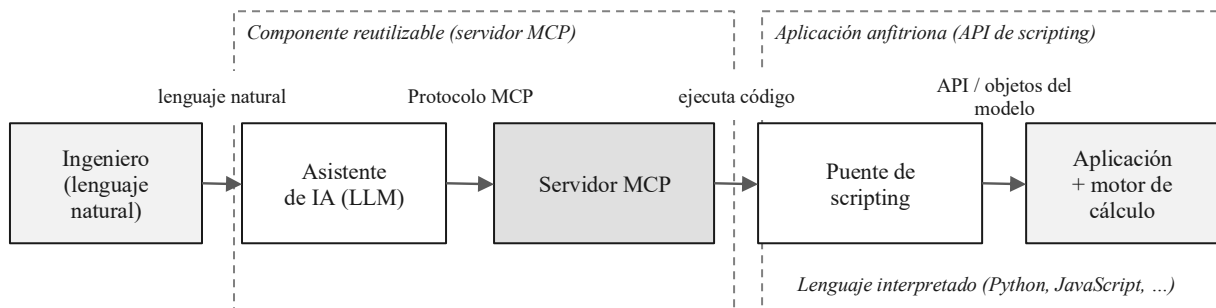
- **Inteligencia artificial generativa / LLM:** modelo de lenguaje de gran escala (LLM) entrenado para comprender y generar texto y código. Es el «motor» que interpreta la consulta del ingeniero y redacta las instrucciones necesarias para resolverla.
- **Agente o asistente de IA:** programa que utiliza un LLM para conversar con el usuario y, además, ejecutar acciones en su nombre por medio de herramientas.
- **Interfaz de programación (API):** conjunto de funciones que un programa publica para ser controlado desde el exterior (crear nudos, barras, cargas, lanzar el análisis, leer resultados) sin necesidad de operar la interfaz gráfica.
- **Lenguaje interpretado o de scripting:** lenguaje que se ejecuta directamente, sin una etapa previa de compilación a código máquina (por ejemplo, Python o JavaScript). Su disponibilidad es la condición técnica que habilita la ejecución de código por parte de la IA.
- **Protocolo de Contexto de Modelo (MCP):** protocolo abierto que estandariza la forma en que un asistente de IA descubre y utiliza las herramientas de una aplicación. Cumple, para la IA, un papel análogo al de un conector universal: una vez que una aplicación «habla» MCP, cualquier asistente compatible puede operarla sin desarrollos a medida.
- **Servidor y cliente MCP:** el servidor MCP es el programa que publica las herramientas de una aplicación; el cliente MCP es el componente del asistente de IA que las descubre y utiliza. La analogía, en términos de ingeniería estructural, es la de un laboratorio de ensayos y el ingeniero proyectista: el laboratorio (servidor) publica en su reglamento interno qué ensayos puede realizar, con qué parámetros y en qué formato entrega los resultados; el ingeniero (cliente) solicita los ensayos que necesita completando un formulario estandarizado, sin necesidad de conocer el funcionamiento interno de los equipos ni la organización del laboratorio.
- **Herramienta (tool):** cada una de las acciones discretas que el servidor ofrece al asistente. El asistente decide cuándo y con qué datos invocarlas.
- **Modo código (code mode):** enfoque en el que la IA, en lugar de invocar muchas herramientas específicas, escribe un script en un lenguaje interpretado y lo ejecuta a través de una única herramienta. Es el eje de la arquitectura propuesta,

desarrollado en detalle más adelante.

- **Entorno restringido (sandbox):** entorno de ejecución aislado y vigilado, en el que el código generado por la IA se ejecuta con permisos muy limitados, sin acceso al sistema de archivos ni a la red, de modo que no pueda dañar el equipo ni el proyecto.
- **Habilidades (skills):** documentos breves, en lenguaje natural, que describen cómo usar cada área de la API. El asistente las carga a demanda, como un manual de consulta.

## ARQUITECTURA DE REFERENCIA

La integración se organiza en una cadena de capas con responsabilidades bien delimitadas (Figura 1). El ingeniero formula su pedido en lenguaje natural; el asistente de IA lo interpreta y se comunica con el servidor MCP mediante el protocolo MCP; el servidor recibe un script y lo ejecuta a través de un puente de scripting que traduce el lenguaje interpretado en llamadas a la API; y la aplicación opera su motor de cálculo sobre el modelo. La respuesta recorre el camino inverso hasta volver al ingeniero en lenguaje natural.

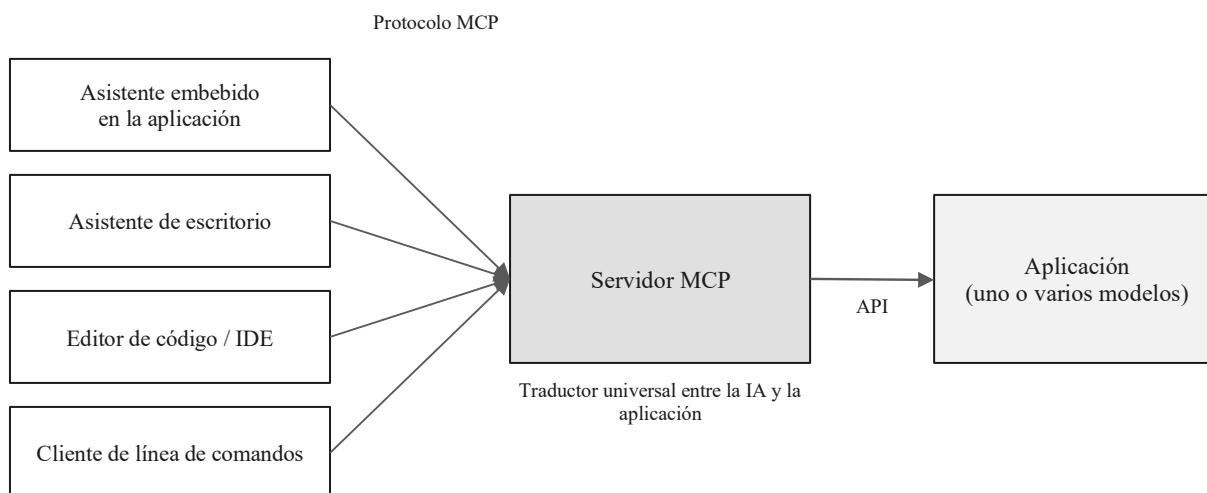


*Figura 1 — Arquitectura de referencia. El servidor MCP y el puente de scripting constituyen un componente reutilizable; el único contrato con la aplicación anfitriona es su API de scripting.*

Dos propiedades hacen que el patrón sea independiente del producto. Primero, el único punto de contacto con la aplicación es su API de scripting: el servidor nunca accede a las interioridades del programa, sólo a las funciones que este publica. Segundo, el servidor MCP y el puente de scripting conforman un componente reutilizable que puede acompañar a cualquier aplicación que cuente con un lenguaje interpretado capaz de alcanzar su motor.

## Un servidor, múltiples asistentes

Al adoptar un protocolo abierto, un mismo servidor puede ser utilizado por distintos asistentes sin desarrollos adicionales (Figura 2): un asistente embebido en la propia aplicación, un asistente de escritorio, un editor de código con IA o un cliente de línea de comandos. La lógica de control se escribe una sola vez y queda disponible para todo el ecosistema de herramientas compatibles con MCP.



*Figura 2 — El servidor MCP se comporta como un traductor universal: distintos asistentes de IA lo reutilizan para operar uno o varios modelos de la aplicación.*

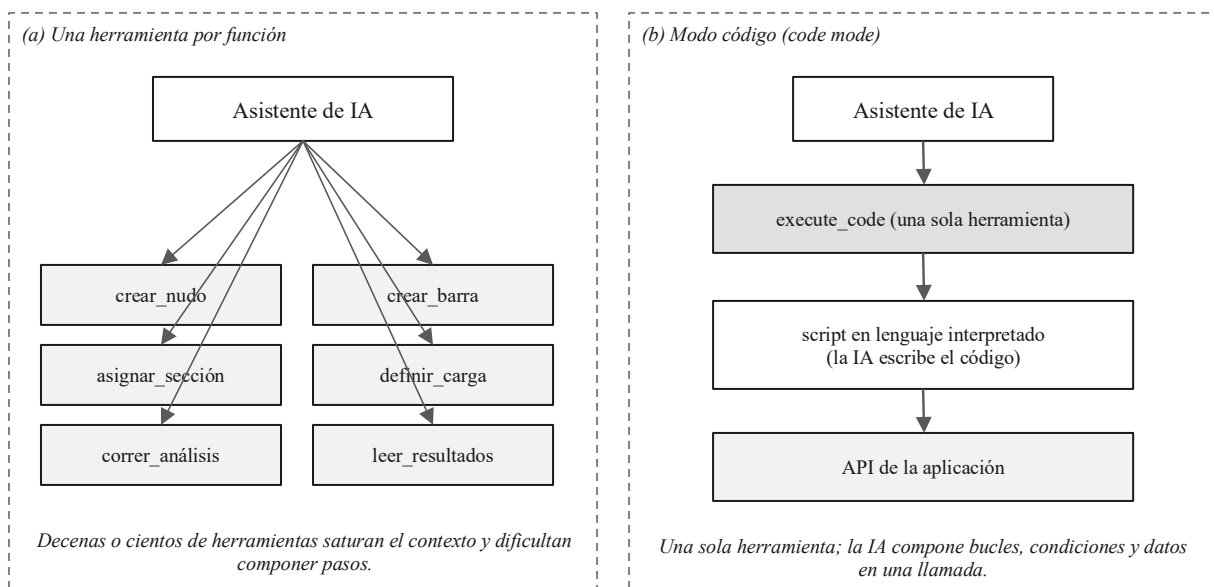
## Componentes del servidor

| Componente                     | Responsabilidad                                                                                                                           |
|--------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| Capa de transporte             | Recibe las peticiones del asistente como proceso hijo (modo stdio) o como servicio HTTP local, y aplica las defensas de seguridad de red. |
| Registro de instancias         | Detecta los modelos abiertos en el equipo y les asigna identificadores estables (alias) para dirigirse a cada uno sin ambigüedad.         |
| Ejecutor / entorno restringido | Valida el script generado por la IA, lo ejecuta de forma aislada y normaliza el resultado. Es el corazón de la seguridad del sistema.     |
| Puente de scripting            | Conecta el lenguaje interpretado con el motor de la aplicación y ejecuta cada llamada respetando el modelo de hilos que esta exige.       |
| Documentación / habilidades    | Entrega, a demanda, la guía de uso de la API. Su peso depende de cuán autodescriptiva sea la API.                                         |

*Tabla 1 — Componentes del servidor MCP y sus responsabilidades.*

## EL PARADIGMA DE LA EJECUCIÓN DE CÓDIGO (CODE MODE)

Existen dos maneras de dar a un asistente de IA el control de una aplicación. La primera, más intuitiva, consiste en publicar una herramienta MCP por cada función de la API: una para crear un nudo, otra para crear una barra, otra para asignar una sección, y así sucesivamente. La segunda, adoptada en esta arquitectura, consiste en publicar una única herramienta de propósito general —`execute_code`— y dejar que la IA escriba un script completo que invoque la API directamente (Figura 3).



*Figura 3 — Dos enfoques para dar control a la IA. A la izquierda, una herramienta por función; a la derecha, el modo código: una sola herramienta y un script escrito por la IA.*

El modo código presenta ventajas decisivas. Los LLM actuales se han entrenado con enormes volúmenes de código fuente, de modo que escribir un script correcto es una de las tareas en las que mejor se desempeñan. Una sola llamada puede contener bucles, condiciones y procesamiento de datos —por ejemplo, crear un pórtico completo iterando sobre niveles y vanos—, algo que con herramientas atómicas exigiría decenas de invocaciones sucesivas. Además, no es necesario enumerar cientos de herramientas en el contexto del modelo, lo que reduce el consumo de tokens y la confusión. Por último, el script aprovecha todo el ecosistema del lenguaje (manejo de listas, matemática, lectura de archivos), heredando una expresividad que sería costoso replicar herramienta por herramienta.

El requisito técnico para aplicar el modo código es disponer de un lenguaje interpretado o de scripting cuyo intérprete pueda alcanzar el motor de la aplicación. Python es la elección natural por su difusión en ingeniería, pero el patrón es válido para cualquier lenguaje dinámico —JavaScript, Lua u otros— que no requiera una etapa de compilación. A cambio de esta potencia, el enfoque traslada dos responsabilidades al diseño: garantizar la seguridad de la ejecución y asegurar que la IA conozca la API que va a invocar, dos cuestiones que se abordan más adelante.

## CÓMO CONOCE LA IA LA API

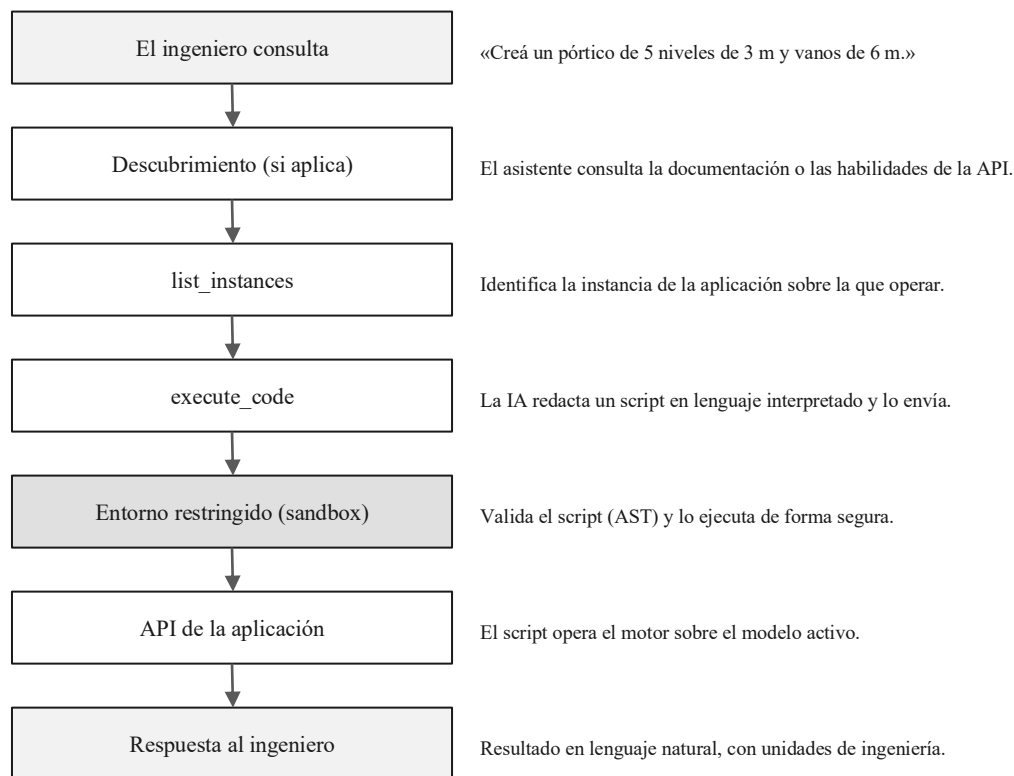
Para escribir un script correcto, la IA debe conocer las funciones disponibles, sus argumentos y sus convenciones. Se identificaron dos estrategias complementarias, cuya elección depende de la naturaleza de la API.

- **Habilidades cargables a demanda.** cuando la API es extensa, idiomática o contiene sutilezas (orden de operaciones, convenciones de índices, unidades), conviene entregar la documentación en forma de habilidades que el asistente carga sólo cuando las necesita. Así se mantiene reducido el contexto y se concentra el conocimiento en una única fuente autorizada, fácil de corregir y mejorar.
- **API autodescriptiva.** cuando la API se diseña con pocas funciones, bien nombradas y autoexplicativas, la propia API actúa como documentación. En este caso, la IA infiere su uso a partir de los nombres y firmas, y las habilidades resultan innecesarias o mínimas.

Ambas estrategias se observan en los casos de estudio que se presentan más adelante y constituyen la principal diferencia entre ellos: una API amplia heredada favorece las habilidades, mientras que una API nueva y deliberadamente concisa puede prescindir de ellas.

## FLUJO DE UNA INTERACCIÓN

La Figura 4 ilustra el recorrido de una consulta típica. Ante un pedido como «creá un pórtico de 5 niveles de 3 m y vanos de 6 m», el asistente, de ser necesario, consulta la documentación de la API, identifica el modelo sobre el que debe operar y redacta el script correspondiente, que envía mediante `execute_code`. El servidor valida el script en el entorno restringido, lo ejecuta a través de la API y devuelve el resultado, que el asistente presenta al ingeniero en lenguaje natural y con las unidades correctas.



*Figura 4 — Flujo de una consulta en lenguaje natural, desde el pedido del ingeniero hasta la respuesta, con la ejecución validada del script en el centro.*

Un detalle técnico recurrente es la conciliación entre el modelo de ejecución del asistente —asíncrono— y el de la mayoría de las aplicaciones de escritorio, cuyo motor e interfaz operan en un único hilo. El servidor resuelve esta tensión ejecutando cada script en un hilo dedicado y esperándolo con un tiempo límite, de modo que una operación que se demore no bloquee al resto del sistema y el asistente reciba siempre una respuesta: el resultado o un mensaje de error claro.

## SEGURIDAD Y PRIVACIDAD

Delegar en una IA la ejecución de código sobre un modelo estructural plantea una pregunta legítima: ¿qué impide que el asistente modifique datos que no debería tocar, acceda a archivos del equipo o envíe información del proyecto a servidores externos? La respuesta no es una única barrera, sino varias capas independientes —al modo de los redundantes sistemas de seguridad en estructuras críticas— de manera que el fallo de una no comprometa al conjunto.

Desde el punto de vista de la privacidad, la decisión de diseño más relevante es que todo el procesamiento ocurre localmente en el equipo del usuario. El modelo estructural, el código generado por la IA y cualquier resultado numérico nunca abandonan la máquina: no se envían a la nube ni a servidores del fabricante. Para estudios de ingeniería que trabajan con proyectos confidenciales —infraestructura, edificios institucionales, instalaciones industriales— esta característica es tan importante como la precisión del cálculo.

Desde el punto de vista de la integridad del modelo, el código que propone la IA se ejecuta en un entorno restringido (sandbox) que actúa como un perímetro de contención: el script puede leer y modificar el modelo a través de la API autorizada, pero no puede importar librerías externas, leer o escribir archivos del disco, ni realizar llamadas de red. La analogía estructural es directa: así como se diseñan juntas de expansión para que un movimiento no se propague a toda la estructura, el sandbox garantiza que un script erróneo o malintencionado no pueda causar daño fuera del perímetro definido.

Finalmente, el canal de comunicación entre el asistente y el servidor se protege mediante autenticación por token, restricción a la dirección local del equipo y firma digital del ejecutable. Estas medidas impiden que un tercero en la red local pueda enviar instrucciones al servidor haciéndose pasar por el asistente. El conjunto de capas sigue el principio de defensa en profundidad: cada barrera falla de forma segura y, ante la duda, bloquea.

## DOS IMPLEMENTACIONES DE REFERENCIA

Para validar la replicabilidad del patrón, la arquitectura se implementó en dos programas comerciales de análisis y diseño estructural a los que los autores tienen acceso y sobre los que desarrollan habitualmente su trabajo profesional: STAAD.Pro y RAM Elements. Su elección obedece únicamente a esta disponibilidad y a que constituyen el entorno sobre el que los autores pueden materializar su propia implementación; no supone una recomendación ni una comparación de méritos entre productos. El patrón propuesto es independiente del programa y aplicable a cualquier aplicación que exponga —o pueda exponer— una API de scripting. Pese a estar contruidos sobre tecnologías de base muy distintas, ambos comparten el mismo esquema —servidor MCP, modo código y entorno restringido— y difieren únicamente en cómo resuelven el puente de scripting y la documentación de la API.

- **STAAD.Pro — API externa con habilidades.** programa de análisis y diseño estructural por elementos finitos con una API COM extensa y madura (OpenSTAAD). El puente de scripting es un envoltorio de Python sobre dicha API,

y el servidor MCP es un ejecutable independiente que la aplicación lanza y supervisa. Dada la amplitud de la API, se emplean habilidades cargables a demanda.

- **RAM Elements — intérprete embebido y API autodescriptiva.** programa de análisis y diseño estructural de código nativo (Delphi / Object Pascal) en el que se embebió un intérprete de Python dentro del propio ejecutable, y el servidor MCP funciona como un servicio HTTP embebido en la aplicación. La API se diseñó desde cero con un número reducido de funciones autodescriptivas, por lo que no requiere habilidades.

| Aspecto                      | STAAD.Pro                                                       | RAM Elements                                       |
|------------------------------|-----------------------------------------------------------------|----------------------------------------------------|
| Lenguaje base del programa   | C++ / .NET (nativo)                                             | Delphi / Object Pascal (nativo)                    |
| API de scripting             | OpenSTAAD (API COM, cientos de funciones)                       | API Python nativa (≈ 78 funciones)                 |
| Puente de scripting          | Envoltorio Python sobre la API COM, en un proceso externo       | Intérprete Python embebido en el propio ejecutable |
| Ubicación del servidor MCP   | Ejecutable independiente, lanzado y supervisado por el programa | Servicio HTTP embebido dentro del programa         |
| Transporte                   | stdio y HTTP local                                              | HTTP local                                         |
| Conocimiento de la API       | Habilidades cargadas a demanda (API extensa)                    | API autodescriptiva; sin habilidades               |
| Identificación de instancias | Tabla de objetos en ejecución del sistema + alias               | Archivo de descubrimiento por proceso              |
| Herramienta central          | execute_code (modo código)                                      | execute_code (modo código)                         |

*Tabla 2 — Comparación de las dos implementaciones de referencia. El patrón se conserva pese a la disparidad de las tecnologías de base.*

Lo relevante no es el desempeño de uno u otro programa, sino que dos aplicaciones tan diferentes como STAAD.Pro —con su API COM externa de larga data— y RAM Elements —con intérprete Python embebido en un ejecutable nativo— convergen en la misma arquitectura. Esto sugiere que el patrón —servidor MCP más modo código— es transferible a otros programas de ingeniería que puedan ofrecer un lenguaje interpretado sobre su motor, con independencia del fabricante o de la tecnología subyacente.

## EJEMPLOS DE USO

A continuación se presentan dos casos de uso concretos que ilustran el tipo de tarea que el asistente puede resolver en la práctica. Ambos son representativos de tareas habituales en el ejercicio profesional que, sin asistencia de IA, demandarían una secuencia manual de pasos repetitivos y propensos a error.

## Estudio de alternativas de cercha para tinglado (RAM Elements)

Un ingeniero necesita comparar tres tipologías de cercha para la cubierta de un tinglado industrial de 24 m de luz: Pratt, Warren y Howe, todas con la misma altura de cuerda (2,0 m) y el mismo espaciado de correas. El objetivo es identificar la opción con menor peso de acero y menor relación demanda/capacidad en los elementos más solicitados, para iguales cargas de viento y nieve según reglamento.

El flujo de trabajo con el asistente se reduce a una instrucción en lenguaje natural: «Modelá las tres cerchas en instancias separadas, asigná perfiles de la serie L normalizados, aplicá las cargas de diseño y compárame el peso total y la relación D/C máxima en cada alternativa». El asistente genera un script único que, a través de la API de RAM Elements, crea los tres modelos, define los casos de carga, lanza el análisis y el diseño de acero, y extrae los resultados en una tabla comparativa. La tarea —que manualmente implicaría construir y verificar cada modelo por separado— se completa en una sola interacción, con resultados listos para la toma de decisión del ingeniero.

Es importante hacer notar que mientras más precisas sean las instrucciones, los resultados en general estarán más acordes a lo esperado. Este proceso puede ser iterativo hasta que el asistente genere un script que satisfaga el pedido del ingeniero. El asistente utilizado para este caso es Visual Studio Code, el mismo que es gratuito y de código abierto.

Para este estudio, las instrucciones en lenguaje natural son:

```
Quiero comparar tres tipologías de cercha para la cubierta de un tinglado industrial y encontrar cuál es la más liviana que cumple con el diseño.
Geometría:
Luz libre: 24 m
Altura de cercha: 2 m
Cerchas simplemente apoyadas (pin en un extremo, roller en el otro)
Tipologías a comparar: Pratt, Howe y Warren
Pratt y Howe: 8 paneles iguales de 3 m
Warren: diagonales alternadas con nodos superiores en X = 0, 3, 9, 15, 21, 24 m
y nodos inferiores en X = 0, 6, 12, 18, 24 m, con montantes solo en los extremos
Material: Acero A36 para todos los elementos.
Secciones candidatas:
Cordones superior e inferior: perfiles W, listados en el archivo W.txt
Diagonales y montantes: doble ángulo T2L, listados en el archivo T2L.txt
Diagonales y montantes deben tener articulaciones de momento en ambos extremos (bielas)
Cargas (sobre el cordón superior):
Carga muerta: 500 N/m (sin incluir peso propio, que debe calcularse aparte activando self-weight)
Carga viva: 2000 N/m
Casos de carga:
CM: Carga Muerta (categoría DL), con peso propio habilitado
```

CV: Carga Viva (categoría LL)

**Combinaciones:**

D1: 1.4·CM (diseño)

D2: 1.2·CM + 1.6·CV (diseño)

S1: 1.0·CM (servicio)

S2: 1.0·CM + 1.0·CV (servicio)

**Código de diseño: AISC 360-22 LRFD**

**Optimización:**

Para cada tipología, encontrá la combinación más liviana de secciones (una W para cordones, una T2L para diagonales y una T2L para montantes) que cumpla con el diseño (ratio D/C  $\leq 1.0$  en todos los elementos). El peso propio real se debe obtener de las reacciones del análisis, no estimado. Ordená las secciones candidatas de menor a mayor peso por metro y usá un método eficiente (no fuerza bruta) para recorrer el espacio de combinaciones.

**Resultado:**

Mostrá para cada tipología: secciones óptimas asignadas, peso propio real (kN) y ratio D/C máximo

Identificá cuál de las tres es la más liviana que cumple

Armá un modelo final con las tres cerchas óptimas una al lado de la otra (separadas en Z: Pratt en Z=0, Howe en Z=5 m, Warren en Z=10 m)

Guardá el modelo final en optimal\_compare\_trusses.retx

Este texto captura la intención completa del script. Un ingeniero no necesitaría decir cómo buscar (binary search, coordinate descent, cache), solo qué quiere: la combinación más liviana que cumpla, de forma eficiente. La estrategia algorítmica es responsabilidad de la IA.

### Resultado de la Optimización — Armaduras de 24 m, h = 2 m

Código de diseño: AISC 360-22 LRFD

**Ganadora: Pratt — 21.53 kN (la más liviana)**

| Tipología | Cuerdas | Diagonales          | Verticales          | Peso propio | Max D/C |
|-----------|---------|---------------------|---------------------|-------------|---------|
| Pratt     | W 10X26 | T2L<br>2X2X1/8X3/8  | T2L<br>3X3X3/16X3/8 | 21.53 kN    | 0.69    |
| Howe      | W 10X26 | T2L<br>4X4X5/16X3/8 | T2L<br>2X2X1/8X3/8  | 25.93 kN    | 0.83    |
| Warren    | W 12X30 | T2L<br>4X4X5/16X3/8 | T2L<br>2X2X1/8X3/8  | 28.07 kN    | 0.85    |

*Tabla 3 — Resultados del estudio comparativo de cerchas optimizadas en RAM Elements (AISC 360-22 LRFD).*

La armadura Pratt resultó un 17% más liviana que la Howe y un 23% más liviana que la Warren. Esto es consistente con la teoría: en una Pratt con carga gravitacional, las diagonales trabajan en tracción (secciones más livianas) y las verticales en compresión (moderada), mientras que en la Howe las diagonales comprimen y necesitan secciones más robustas.

Organiza:

Conviene señalar que los pesos propios de la Tabla 3 no son estimaciones: se obtienen de las reacciones verticales del análisis de cada modelo, tal como se exigió en las instrucciones, y la relación demanda/capacidad máxima permanece por debajo de la unidad en las tres tipologías (0,69; 0,83 y 0,85), lo que confirma que la comparación se realiza entre soluciones igualmente verificadas según AISC 360-22 LRFD. Como comprobación de orden de magnitud, el peso propio de la Pratt resulta coherente con un cálculo elemental: sus cordones W10X26 ( $\approx 0,38$  kN/m sobre unos 48 m de cuerda superior e inferior) aportan del orden de 18 kN, a los que se suman las diagonales y montantes —perfiles notablemente más livianos—, en línea con los 21,53 kN informados.

Los valores que reporta el asistente son trazables hasta las llamadas de la API y verificables con los procedimientos de ingeniería habituales.

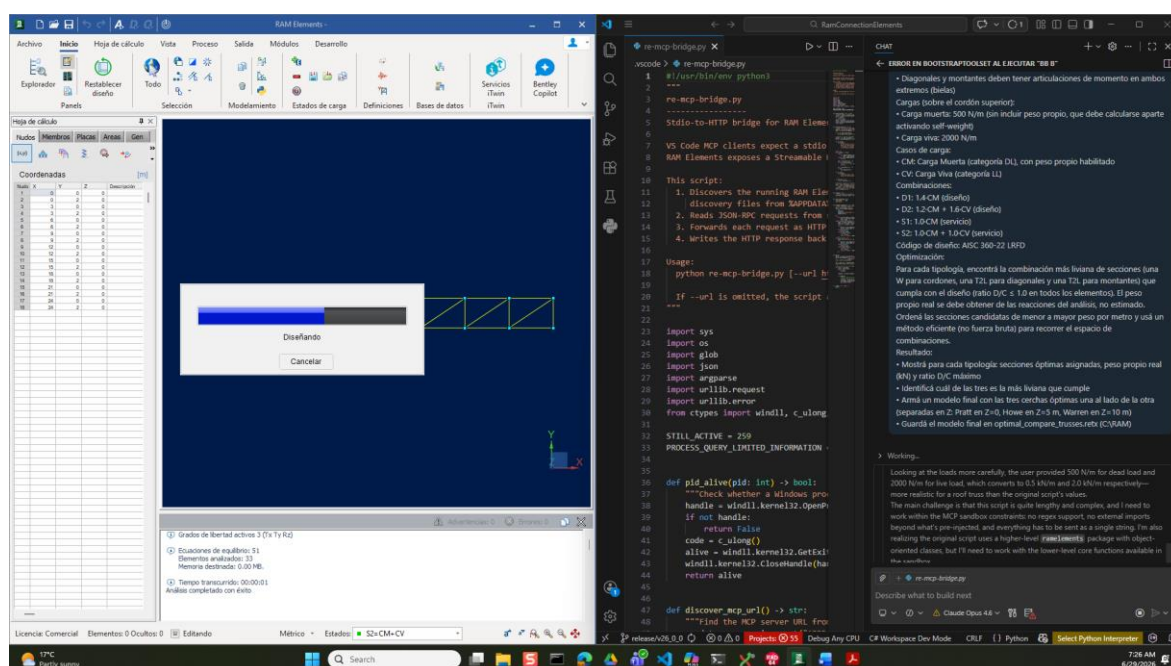
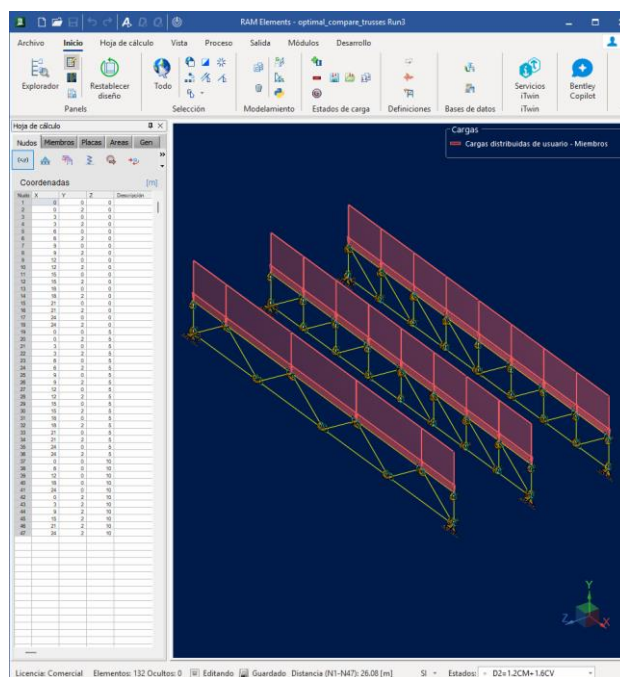
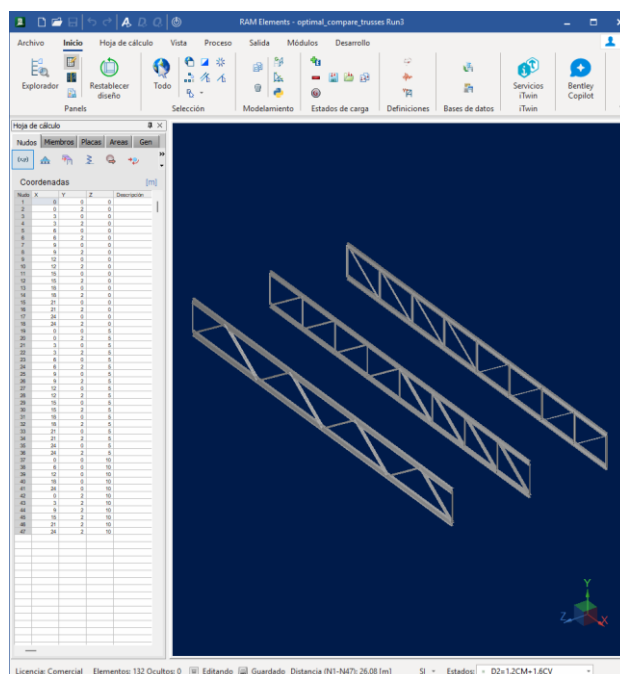


Figura 5 — Instrucciones en Visual Studio Code (asistente) y este ejecutando en RAM Elements.



*Figura 6 — Modelo final optimizado generado por el asistente: las tres cerchas óptimas (Pratt, Howe y Warren) dispuestas en paralelo.*



*Figura 7 — Vista complementaria del modelo final optimizado generado por el asistente.*

## Generación de geometría compleja para un edificio (STAAD.Pro)

El segundo ejemplo corresponde a la generación de la geometría de una torre de acero con sistema de riostras diagonales (diagrid) y giro progresivo en altura: una estructura cuya planta cuadrada rota unos pocos grados en cada nivel, de modo que el conjunto describe una torsión continua. Definir manualmente la posición de cada nudo y barra —con coordenadas que dependen de una rotación acumulada piso a piso— es una tarea tediosa y muy propensa a errores de transcripción.

Para este caso se utilizó Claude Desktop, otro cliente compatible con MCP que se conecta al mismo servidor openstaad-mcp de STAAD.Pro. Esto ilustra una ventaja central de la arquitectura: al adoptar un protocolo abierto, un mismo servidor puede operarse indistintamente desde un editor de código, un asistente embebido o un cliente de escritorio, sin desarrollos adicionales.

Las instrucciones en lenguaje natural entregadas al asistente fueron:

Creá un modelo estructural 3D de una torre de acero tipo «diagrid» con giro progresivo, usando el sistema de coordenadas de STAAD: X = dirección horizontal, Y = vertical (elevación), Z = dirección horizontal.

### Geometría:

Altura total: 300 ft

Número de pisos: 20

Altura entre pisos: 15 ft

Dimensiones en la base: 40 ft x 40 ft

Nodos del piso base (Y = 0): (-20, 0, -20), (20, 0, -20), (20, 0, 20), (-20, 0, 20)

### Generación de pisos:

Para cada nivel i (i = 0 a 20): elevación Y = i · 15 ft; crear 4 nodos de esquina que formen un cuadrado de 40 ft x 40 ft.

Rotar cada placa de piso 3 grados alrededor del eje Y global respecto del piso anterior.

Acumular la rotación progresivamente para generar la torre con giro.

### Sistema estructural:

Sistema diagrid: conectar nodos entre pisos mediante diagonales; cada módulo abarca 2 pisos y forma patrones romboidales en el perímetro.

Vigas perimetrales: conectar nodos horizontalmente en cada nivel.

Columnas: crear miembros verticales siguiendo las posiciones de nodos rotadas (geometría con giro).

### Secciones:

Miembros diagrid: HSST12X12X0.5.

Vigas perimetrales y columnas: W14X90.

### Material:

Asignar todos los miembros como STEEL.

### Apoyos y cargas:

Asignar empotramientos en los nodos de la base (Y = 0).

Agregar carga de peso propio (self-weight).

### Análisis:

Ejecutar el análisis estructural.

Organiza:

#### Unidades:

Verificar las unidades actuales del modelo antes de la creación y aplicar la conversión necesaria para que toda la geometría y las propiedades queden definidas en unidades consistentes.

El asistente interpreta la descripción, calcula las coordenadas de cada nudo aplicando la rotación acumulada mediante expresiones trigonométricas dentro del script, genera las riostras diagonales, las vigas y las columnas, asigna secciones y material, define los apoyos y la carga de peso propio, y ejecuta el análisis. El modelo —cuya coherencia geométrica sería muy difícil de garantizar manualmente— queda resuelto en una sola interacción.

#### Resultado del modelo — Torre diagrid de 20 pisos, giro total 60°

| Parámetro           | Valor                                       |
|---------------------|---------------------------------------------|
| Altura total        | 300 ft ( $\approx 91,4$ m)                  |
| Pisos               | 20 $\times$ 15 ft ( $\approx 4,57$ m c/u)   |
| Giro total          | 60° (3°/ piso, progresivo)                  |
| Planta base         | 40 ft $\times$ 40 ft ( $\approx 12,2$ m)    |
| Nudos / barras      | 84 / 324                                    |
| Vigas perimetrales  | 84 $\times$ W14X90                          |
| Columnas            | 80 $\times$ W14X90                          |
| Riostras diagonales | 160 $\times$ HSST12X12X0.5                  |
| Material / apoyos   | Acero (STEEL) / 4 empotramientos en la base |

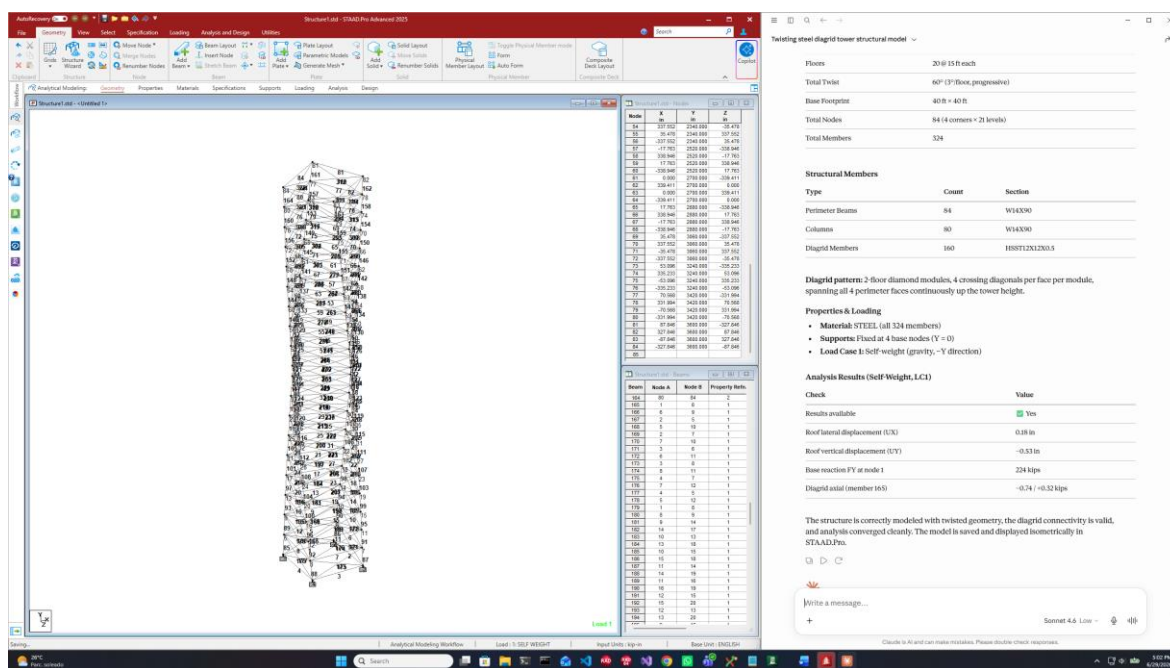
*Tabla 4 — Modelo generado por el asistente: geometría y miembros.*

| Verificación (peso propio, LC1)           | Valor                                          |
|-------------------------------------------|------------------------------------------------|
| Desplazamiento lateral en cubierta (UX)   | 0,18 in ( $\approx 4,6$ mm)                    |
| Desplazamiento vertical en cubierta (UY)  | -0,53 in ( $\approx -13,5$ mm)                 |
| Reacción vertical en la base (nudo 1, FY) | 224 kips ( $\approx 996$ kN)                   |
| Esfuerzo axial en riostra (barra 165)     | -0,74 / +0,32 kips ( $\approx -3,3$ / +1,4 kN) |

*Tabla 5 — Resultados del análisis bajo peso propio extraídos por el asistente desde STAAD.Pro.*

El análisis convergió correctamente y los resultados son coherentes con una sollicitación de peso propio: los desplazamientos en la cubierta son pequeños —del orden de un centímetro en vertical y algo menos en horizontal— y el leve corrimiento lateral es consistente con la asimetría que introduce el giro progresivo de la planta. La validez de la conectividad del diagrid se confirma al obtener esfuerzos de tracción y compresión en sus miembros, y la suma de las reacciones verticales en la base reproduce el peso propio total de la estructura. Estos valores, extraídos por el asistente a través de la misma API que utilizaría la interfaz gráfica, pueden verificarse con los procedimientos de ingeniería habituales.

Conviene destacar un aspecto que las tablas evidencian de manera implícita: el asistente mantuvo la consistencia de unidades de forma automática. Siguiendo las instrucciones del pedido y apoyándose en las habilidades correspondientes, verificó las unidades activas del modelo antes de crear la geometría y aplicó las conversiones necesarias entre el sistema imperial y el SI, de modo que toda la geometría, las secciones y los resultados quedaran expresados de forma coherente. Este control — trivial de enunciar, pero responsable de una proporción importante de los errores de modelado en la práctica— ilustra que, con instrucciones precisas y habilidades bien redactadas, el asistente no sólo automatiza tareas mecánicas, sino que además preserva la integridad numérica del modelo.



servidor MCP —estimada en cinco a seis semanas de trabajo tradicional— se completó en el orden de dos jornadas de un desarrollador experimentado. La IA propuso enfoques, leyó el código fuente existente, generó el código repetitivo (decenas de funciones con un patrón común) y redactó documentación; el desarrollador aportó el conocimiento del dominio, eligió entre alternativas y corrigió los detalles. La IA fue, en este sentido, un multiplicador de la capacidad del ingeniero, no un reemplazo.

- **Apostar a un estándar abierto.** adoptar un protocolo abierto, en lugar de una integración a medida para un único asistente, multiplicó el alcance del trabajo y lo dejó disponible para todo el ecosistema de herramientas compatibles.
- **El modo código como eje.** dejar que la IA escriba scripts, en vez de encadenar herramientas atómicas, fue la decisión de mayor impacto: aprovecha aquello en lo que los LLM sobresalen y permite componer operaciones complejas en una sola llamada.
- **Documentar según la API.** no existe una única estrategia para que la IA conozca la API. Una API extensa heredada favorece las habilidades; una API nueva y concisa puede prescindir de ellas. Diseñar la API pensando en que la lea una IA simplifica todo lo demás.
- **El equilibrio entre potencia y seguridad.** existe una tensión permanente entre la potencia de ejecutar código arbitrario y la seguridad. La respuesta no fue una única barrera, sino varias capas modestas que, combinadas, ofrecen garantías sólidas y fallan de forma segura.
- **Las particularidades de cada plataforma.** la interacción con motores de un solo hilo y con puentes entre lenguajes resultó delicada: fue necesario ejecutar cada script en su propio hilo, gestionar tiempos límite y depurar problemas sutiles de codificación de datos y de acceso entre hilos. Reconocer estas limitaciones y diseñar en torno a ellas simplificó el sistema.
- **La aplicación como dueña del ciclo de vida.** delegar en la aplicación anfitriona el arranque, la supervisión y la finalización del servidor evitó procesos huérfanos y mantuvo al componente de IA deliberadamente simple respecto de su propio ciclo de vida.
- **Un método transferible.** el flujo de trabajo con IA —describir lo que se necesita, dejar que lea el código y proponga un enfoque, evaluar en conjunto, delegar lo repetitivo y revisar y corregir— es independiente del lenguaje y aplicable a cualquier pila tecnológica.

## BENEFICIOS Y DESAFÍOS

Entre los beneficios observados se destacan la reducción significativa del tiempo dedicado a tareas repetitivas, la disminución de errores de transcripción y una mayor accesibilidad para usuarios poco familiarizados con la programación o con la propia interfaz del programa. El asistente actúa, en la práctica, como un colega incansable que conoce la API y traduce el lenguaje del ingeniero en operaciones concretas.

No obstante, persisten desafíos importantes. El primero y más relevante es la validación de los resultados generados por la IA: el asistente puede equivocarse o «alucinar», por lo que el ingeniero debe permanecer en el circuito, revisando críticamente cada resultado antes de adoptarlo. La transparencia del sistema ayuda en este punto: todo valor devuelto proviene de la misma llamada de la API que utilizaría la interfaz gráfica, de modo que la validación de ingeniería tradicional sigue siendo aplicable. Un segundo desafío son las limitaciones del lenguaje natural en contextos técnicos altamente especializados, donde un término ambiguo puede dar lugar a interpretaciones erróneas; la mitigación pasa por consultas precisas, por una API o habilidades bien redactadas y por la supervisión del profesional.

## CONCLUSIONES Y TRABAJO FUTURO

Se presentó una arquitectura de referencia, independiente del producto, para integrar asistentes de IA en software de ingeniería. El patrón combina el protocolo abierto MCP con el paradigma de la ejecución de código: una única herramienta a través de la cual la IA escribe y ejecuta scripts en un lenguaje interpretado, dentro de un entorno restringido y seguro. Su implementación en dos programas de tecnologías de base radicalmente distintas demuestra que se trata de un patrón genuinamente replicable, aplicable a cualquier aplicación capaz de exponer un lenguaje interpretado sobre su motor.

La experiencia confirma que un asistente de IA puede reducir sustancialmente el tiempo de las tareas rutinarias de modelado y hacer la automatización accesible a un público más amplio, siempre que el ingeniero conserve su rol de validador de los resultados. Como trabajo futuro se prevén la ampliación de la cobertura de la API, una mayor estandarización de la documentación orientada a la IA, esquemas de salida más estrictos y la exploración de flujos de trabajo de larga duración que hoy se ejecutan de forma sincrónica.

Una línea especialmente prometedora, habilitada por los avances recientes en agentes de IA, es la automatización en bucle (loop): en lugar de resolver una consulta en un único paso, el asistente itera de forma autónoma —genera una solución, ejecuta el análisis, evalúa los resultados frente a un criterio y refina el modelo— hasta

converger en un diseño que satisface las restricciones impuestas. Este esquema, descrito por Anthropic como el patrón evaluador-optimizador y, en su forma más autónoma, como un agente que emplea herramientas con la retroalimentación del entorno en un bucle [3], resulta particularmente eficaz cuando existen criterios de evaluación claros y el refinamiento iterativo aporta valor medible. Aplicado a la ingeniería estructural, abre la puerta a procesos de optimización y verificación iterativa con mínima intervención manual, como el dimensionamiento de elementos, la búsqueda de configuraciones de menor peso o el ajuste de modelos hasta cumplir un conjunto de reglamentos. Dado que el patrón propuesto se apoya en protocolos y APIs públicas a través de MCP, estos bucles agénticos pueden componer y encadenar distintas aplicaciones —análisis, diseño, detallado y reportería— en un mismo flujo, siempre bajo la supervisión del ingeniero, que define los objetivos y valida el resultado final. Conviene subrayar que, aun cuando la IA persiga un óptimo según el criterio que se le indique, la responsabilidad profesional sigue recayendo en el ingeniero estructural: es él quien debe revisar que la solución propuesta cumpla con las normativas y los estándares de ingeniería aplicables, así como con las hipótesis de diseño y los criterios de seguridad que el algoritmo no necesariamente contempla. La automatización en bucle acelera la exploración de alternativas, pero no sustituye el juicio ni la verificación normativa del profesional.

## REFERENCIAS

- [1] Anthropic, Model Context Protocol — especificación del protocolo abierto. Disponible en: <https://modelcontextprotocol.io>
- [2] Anthropic, Code execution with MCP — building more efficient agents (modo código). Disponible en: <https://www.anthropic.com/engineering/code-execution-with-mcp>
- [3] Anthropic, Building Effective Agents — patrones de sistemas agénticos (evaluador-optimizador y agentes en bucle). Disponible en: <https://www.anthropic.com/engineering/building-effective-agents>
- [4] Python Software Foundation, The Python Language Reference. Disponible en: <https://docs.python.org/3/reference/>
- [5] Python4Delphi — integración de un intérprete de Python en aplicaciones Delphi. Disponible en: <https://github.com/pyscripter/python4delphi>
- [6] Bentley Systems, Infrastructure AI Co-Innovation Initiative. Disponible en: <https://www.bentley.com/software/infrastructure-ai-co-innovation-initiative/>
- [7] Bentley Systems, openstaad-mcp — servidor MCP de código abierto para STAAD.Pro. Disponible en: <https://github.com/BentleySystems/openstaad-mcp>