

DESARROLLO DE UN AGENTE ESTRUCTURAL DE CÁLCULO SOBRE UN SOLVER VERIFICADO

CHESTA, Bautista ^(a, b) BERTERO, Raúl ^(a) CARRONE, Federico ^(b) KINGSTON, Diego ^(a, b)

(a) Facultad de Ingeniería de la Universidad de Buenos Aires (FIUBA)

(b) LambdaClass y Ergodic Group

bchesta@fi.uba.ar · rbertero@fi.uba.ar · federico@lambdaclass.com · dkingston@fi.uba.ar

Palabras clave: inteligencia artificial, agente estructural, modelado conversacional, solver verificado

RESUMEN

El flujo de trabajo del ingeniero estructural permanece esencialmente manual a pesar de décadas de evolución del software comercial. La IA generativa permite transformar esa práctica, pero un modelo de lenguaje no puede garantizar la corrección numérica de un análisis ni el cumplimiento normativo. La tesis de este trabajo es que un agente de IA resulta confiable sólo si opera como orquestador sobre un solver verificado, sin generar resultados numéricos por cuenta propia. Se presenta un agente conversacional que traduce la intención del ingeniero en acciones de modelado sobre un solver estructural verificado mediante benchmarks y casos de referencia, ejecuta verificaciones conforme a CIRSOC 201 y 301 y genera memorias de cálculo trazables. Se plantea una evolución progresiva: a corto plazo, asistencia en modelado y verificación; a mediano plazo, iteración de diseño con el solver en el bucle; a largo plazo, una interfaz donde el ingeniero decide mientras el agente propone, calcula, verifica y documenta. Se concluye que la incorporación de IA a la ingeniería estructural resulta viable cuando se apoya en herramientas de cálculo verificadas y conserva la trazabilidad necesaria para la práctica profesional.

ABSTRACT

The structural engineer's workflow remains essentially manual despite decades of software evolution. Generative AI can transform that practice, but a language model cannot guarantee numerical correctness or code compliance. The thesis of this work is that an AI agent is reliable only if it operates as an orchestrator over a verified solver, without generating numerical results on its own. A conversational agent is presented that translates the engineer's intent into modeling actions on a structural solver verified through benchmarks and reference cases, performs code verification in accordance with CIRSOC 201 and 301, and generates traceable calculation reports. A progressive roadmap is outlined: in the short term, assistance in modeling and verification; in the medium term, design iteration with the solver in the loop; in the long term, an interface where the engineer decides while the agent proposes, calculates, verifies, and documents. It is concluded that the incorporation of AI into structural engineering is viable when it relies on verified structural analysis tools and preserves the traceability required for professional practice.

1. Introducción

1.1. La ingeniería asistida por computadora

El diseño y el cálculo asistidos por computadora le permitieron al ingeniero automatizar tareas rutinarias, acelerar el cómputo y estudiar escenarios, liberando tiempo para el análisis y la toma de decisiones, con la consiguiente reducción de plazos y costos de ingeniería y construcción. Esos programas se concibieron para un usuario humano que opera una interfaz gráfica. El ingeniero conoce los fundamentos de su disciplina y juzga críticamente los resultados, pero llegar hasta ellos tiene una curva de aprendizaje que no es trivial: hay que dominar la interfaz, entender cómo el programa presenta la salida y, cuando esa salida no se ajusta al análisis que uno quiere hacer, exportar los datos y recurrir a un post-procesamiento propio.

Conviene precisar dónde está hoy el costo. Décadas de evolución del software comercial volvieron veloz la parte estrictamente numérica del análisis. Ensamblar una matriz de rigidez y resolver un sistema lineal es un problema bien comprendido y computacionalmente barato. El tiempo del ingeniero se destina a construir el modelo, rehacerlo cada vez que cambia la arquitectura, definir cargas y combinaciones, verificar elemento por elemento contra la normativa y producir la memoria de cálculo y los planos. El cálculo es la parte rápida; el cuello de botella es el andamiaje manual y repetitivo que lo rodea.

A esa fricción se suma una barrera de costo y acceso. En Argentina y la región, la práctica depende en buena medida de software propietario cuyas licencias profesionales cuestan varios miles de dólares anuales por puesto. Programas como SAP2000, ETABS, STAAD.Pro, RAM Elements y RFEM dominan el mercado profesional ¹, y CYPECAD está muy difundido para el cálculo de viviendas y edificios de estructura sencilla, mientras que las alternativas de código abierto como OpenSees ² o Code_Aster ³ exigen scripting en lenguajes como Tcl o Python, no ofrecen una interfaz gráfica moderna y no implementan la normativa argentina.

1.2. La llegada de la IA

La inteligencia artificial (IA) generativa está cambiando ese panorama, tanto en cómo se escribe el software como en cómo se lo usa. Un agente basado en un modelo de lenguaje permite que el ingeniero exprese su objetivo en lenguaje natural y que el sistema actúe como asistente: prepara los datos de una simulación, interactúa con el programa de cálculo y transforma la salida para que el ingeniero la analice y decida. Una tendencia creciente trata a estos modelos como motores de razonamiento de propósito general, capaces de pasar del reconocimiento de imágenes a la escritura, el cómputo o el diseño de ingeniería con la misma interfaz conversacional ^{4, 5}.

Esa versatilidad tiene un límite preciso. Un modelo de lenguaje genera texto probabilístico: desconoce las leyes de la física y el comportamiento de los sistemas reales, y puede alucinar, devolviendo respuestas que suenan plausibles pero son incoherentes. En análisis estructural el riesgo es concreto: el modelado exige razonamiento espacial y consistencia topológica, y el modelado multi-paso es propenso a alucinaciones y a la acumulación de errores, limitaciones documentadas de los modelos actuales ^{6, 7}. Un agente, por lo tanto, no puede usarse sin supervisión, y el análisis crítico y la decisión siguen del lado del ingeniero.

Una manera de acotar ese riesgo es no pedirle al modelo que calcule. En lugar de mejorarlo hasta que "calcule bien", se le quita el cálculo: el agente propone diseños y escenarios, pero para que un resultado sea válido debe pasar por un programa de cálculo que el agente no puede modificar y que contiene la física del problema. El modelo decide qué herramienta determinística invocar y con qué parámetros, e integra lo que esa herramienta devuelve; nunca origina un número ni inventa geometría. Ese es el sentido preciso de *agente* que se adopta en

este trabajo (lo que se conoce como *tool use* o *function calling*, §2.1): un sistema que orquesta, no que resuelve.

1.3. La práctica profesional

Esta restricción no es una cautela de ingeniería de software, sino que nace de cómo funciona la práctica profesional. Cada resultado estructural lleva una firma, con la consecuente autoría y responsabilidad legal. El número que dimensiona una columna no es un dato más, sino que es un compromiso del ingeniero matriculado que lo suscribe. Un agente que proporciona resultados no verificados no aporta valor de ingeniería ya que le traslada al profesional el riesgo de firmar un cálculo que nadie comprobó. La utilidad de la IA en este dominio depende, entonces, de dos condiciones. La primera es que la corrección del resultado sea una propiedad del solver y no del lenguaje. La segunda es que el sistema conserve la trazabilidad que la firma exige, de modo que cada valor reportado pueda remitirse al cálculo verificado que lo produjo, y no a una afirmación del modelo.

1.4. Contribución y organización del trabajo

La separación entre un modelo que orquesta y un solver determinístico que calcula no es una propuesta aislada de este trabajo. Es el patrón dominante en la literatura reciente de agentes de ingeniería (§2.2), y aquí se lo adopta deliberadamente como principio de diseño establecido en lugar de presentarlo como un hallazgo propio. Lo que distingue a Stabileo de esos precedentes son cuatro decisiones de diseño que, hasta donde alcanza el conocimiento de los autores, ningún sistema previo combina:

1. Acciones tipadas en lugar de generación de código libre. Los precedentes hacen que el modelo genere código Python (por ejemplo, scripts de OpenSeesPy) ejecutado luego contra un solver externo ^{7,8}. Stabileo restringe la salida del agente a un conjunto cerrado de llamadas tipadas con contratos de entrada y salida explícitos (por ejemplo, una acción para crear un pórtico con su luz y su altura como parámetros), validadas contra un esquema antes de ejecutarse. Es una frontera más estrecha y auditable que el código arbitrario.
2. Un solver propio y extensivamente testeado, no un envoltorio de terceros. El motor de Stabileo es un motor de elementos finitos escrito desde cero en Rust (un lenguaje de alto desempeño), con su propia batería de verificación contra soluciones analíticas y benchmarks de referencia, mientras que los precedentes envuelven herramientas existentes.
3. Cumplimiento normativo nativo. El sistema implementa verificación según los reglamentos argentinos CIRSOC 201 ⁹ y CIRSOC 301 ¹⁰, que ningún software de código abierto implementa de forma nativa (su grado de madurez se detalla en §5.2).
4. Ejecución en el navegador, sin instalación local. El motor compila a WebAssembly y corre localmente en el navegador del usuario, sin servidor de cálculo.

Sobre esa base, los aportes concretos de este trabajo son tres. Primero, una arquitectura en la que el agente orquesta y un solver verificado calcula, que aquí se denomina la *frontera de determinismo* (§3) y cuyo alcance y garantías se delimitan con precisión (§3.2). Segundo, su materialización en Stabileo, una plataforma de código abierto que integra el solver en Rust compilado a WebAssembly, la verificación CIRSOC, un agente conversacional multi-proveedor y la generación de memorias y planos, todo ejecutable en el navegador (§5). Tercero, una evaluación con casos de estudio de extremo a extremo cuyos resultados son salidas reales del solver, contrastados además contra un programa comercial, y un relato explícito de qué está validado y qué permanece como trabajo futuro (§6).

El resto del trabajo se organiza así: la §2 sitúa el trabajo frente a la literatura de uso de herramientas por modelos de lenguaje y de agentes sobre solvers de ingeniería; la §3 define la arquitectura y la frontera de determinismo; la §4 traza el camino hacia un solver

probado mediante verificación formal y fuzzing; la §5 describe la implementación en Stabileo; la §6 recorre casos de estudio de extremo a extremo; la §7 discute el impacto en la práctica profesional; la §8 trata las limitaciones y el trabajo futuro; y la §9 cierra con las conclusiones.

2. Trabajo relacionado

2.1. Uso de herramientas por modelos de lenguaje (tool use / function calling)

Una línea reciente de trabajo dota a los modelos de lenguaje de la capacidad de invocar herramientas externas en lugar de resolver todo internamente. ReAct intercala razonamiento y acciones, permitiendo que el modelo consulte herramientas durante la resolución de una tarea ¹¹. A modo de ejemplo, Toolformer muestra que un modelo puede aprender, de forma autosupervisada, cuándo y cómo llamar a APIs externas ¹². Gorilla conecta un modelo con un gran conjunto de APIs y reduce las invocaciones inválidas ¹³. Los proveedores actuales exponen *function calling* nativo, donde el modelo emite una llamada tipada que un sistema externo ejecuta. El hilo común es que el modelo no produce la respuesta final: enruta hacia una herramienta y compone su salida. Stabileo adopta este patrón, con la particularidad de que las herramientas son un solver estructural y módulos de verificación normativa.

2.2. Agentes con un solver o verificador como fuente de verdad

Un patrón emparentado delega la parte que exige corrección a un componente determinístico. Los enfoques de tipo *program-aided* hacen que el modelo genere un programa cuya ejecución, y no el propio modelo, produce el resultado numérico ¹⁴. La generalización es acoplar el modelo a un solver simbólico o numérico de modo que la corrección sea propiedad del verificador y no del lenguaje. Este trabajo aplica esa idea al análisis estructural: el solver verificado y los módulos de verificación CIRSOC son la fuente de verdad, y el agente queda confinado a interpretar la intención, orquestar la ejecución y explicar resultados ya calculados.

2.3. IA aplicada a la ingeniería estructural y la AEC

Los antecedentes de IA en la arquitectura, la ingeniería y la construcción (AEC) pueden ordenarse en tres generaciones. La primera (2010–2020) aplicó redes neuronales a problemas específicos, como la predicción de resistencia del hormigón, la detección de daño o la optimización topológica acelerada con redes convolucionales ¹⁵. La segunda (2020–2024) empleó operadores neuronales y *Graph Neural Networks* para sustituir parcialmente el método de elementos finitos, con aceleraciones de hasta tres órdenes de magnitud pero limitadas a familias paramétricas de geometrías ^{16, 17}. La tercera (2024–presente) usa modelos de lenguaje como interfaz conversacional y agente de orquestación sobre herramientas de ingeniería existentes ⁴. ⁵. Stabileo se inscribe en esta tercera generación: no utiliza la IA para aproximar la física, sino para orquestar un solver verificado.

2.4. Posicionamiento: qué aporta este trabajo

La Tabla 1 resume cualitativamente algunas de las herramientas de análisis estructural más usadas a nivel mundial. Lo que distingue a Stabileo no es cada característica por separado, sino su integración: según el conocimiento de los autores, ninguna herramienta reúne a la vez verificación CIRSOC nativa, un agente de IA orquestador y un núcleo de código abierto.

Herramienta	Tipo	CIRSOC	Navegador	IA integrada
SAP2000 / ETABS	Comercial	Parcial	No	No
STAAD.Pro	Comercial	No	No	No
RAM Elements	Comercial	No	No	No
RFEM / RSTAB	Comercial	No	No	No
CYPECAD	Comercial	Sí	No	No
OpenSees	Open-source	No	No	No
Code_Aster	Open-source	No	No	No
SkyCiv	SaaS	No	Sí	Parcial
Stabileo	Open-source	Sí	Sí	Sí

Tabla 1. Comparación cualitativa de herramientas de análisis estructural y el posicionamiento de Stabileo.

3. Arquitectura del agente: la frontera de determinismo

3.1. Capas: intención, cómputo determinístico, documentación

La arquitectura separa tres responsabilidades (Figura 1). La capa de intención es el agente de IA: interpreta el lenguaje natural, formula preguntas de clarificación y emite acciones tipadas. La capa de cómputo determinístico reúne todo lo que produce datos de ingeniería, es decir el solver que resuelve, los módulos CIRSOC que verifican, los generadores que expanden una tipología en un modelo completo y la ingestión de planos que traduce geometría; todo ello es reproducible y auditable. La capa de documentación produce memorias, planos y cálculos a partir de resultados ya calculados. La regla que vertebra el sistema es que el modelo de lenguaje nunca genera un número ni inventa geometría: ambos provienen de componentes determinísticos.

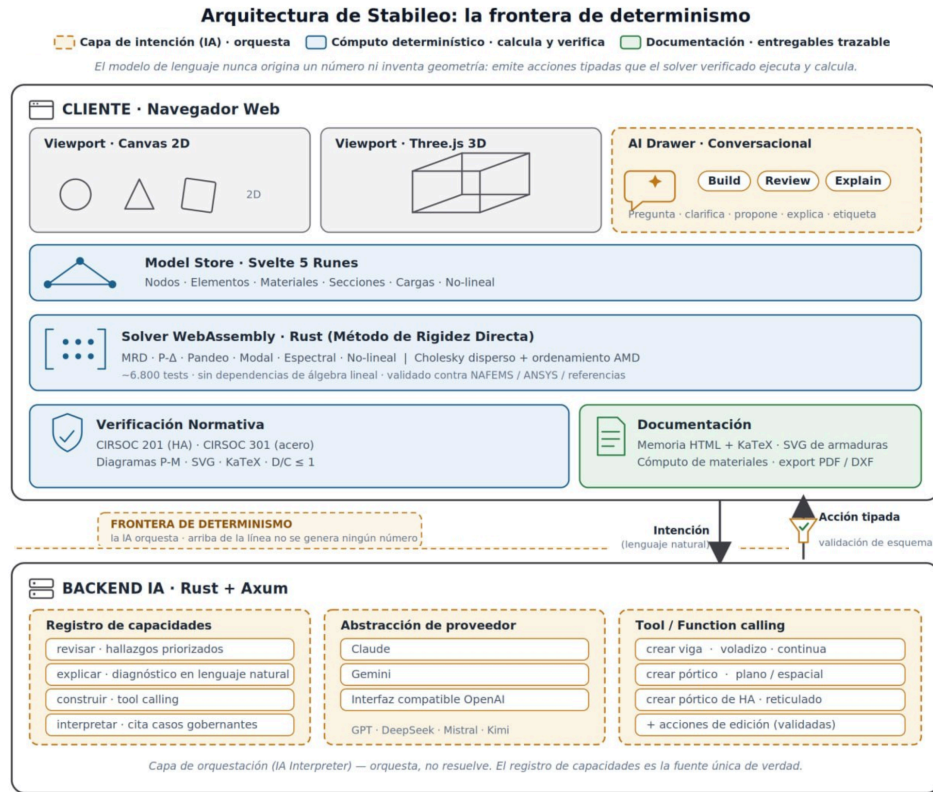


Figura 1. Arquitectura de Stabileo: capas de intención (IA), cómputo determinístico y documentación, separadas por la frontera de determinismo.

3.2. Qué garantiza la frontera y qué no

La frontera de determinismo garantiza que toda afirmación numérica del agente es trazable a un resultado específico del solver verificado (combinación, estación, elemento); que toda salida de la IA se valida contra el esquema del modelo en el backend y en el frontend antes de importarse, rechazando respuestas malformadas o con campos desconocidos; y que, si el servicio de IA no está disponible, el solver, el editor y las verificaciones siguen operando (degradación segura).

La frontera no garantiza, en cambio, que el modelo idealizado sea el correcto para el problema real, que es una decisión de ingeniería, ni que el diseño sea óptimo, ni que las cargas y los supuestos estén completos, ni que la interpretación normativa no requiera revisión. La frontera protege la corrección del cálculo, no la corrección de la intención. Esa distinción es deliberada y se discute en la §7.

3.3. Supervisión humana: aceptar el modelo antes de calcular

El agente se comporta como un asistente de borrador y no como un ejecutor autónomo. El agente propone un cambio, el usuario ve un resumen de lo que se va a modificar y puede aplicar, reintentar o cancelar; y cada cambio aplicado constituye un único paso de deshacer. Cada respuesta del agente lleva, además, una etiqueta de confianza visible que indica si se trata de asesoramiento, de una respuesta basada en el artefacto del solver, de una heurística normativa o de un borrador generado. El ingeniero acepta el modelo antes de que se vuelva activo y antes de adoptar cualquier conclusión de verificación. La supervisión humana es parte fundamental del contrato.

4. Verificación formal y fuzzing: hacia un solver probado

4.1. Motivación

La frontera de determinismo definida en §3 garantiza que toda afirmación numérica del agente es trazable a una salida del solver. Esa garantía tiene, sin embargo, un límite preciso: el solver es confiable porque está extensivamente testado, no porque esté probado. Una suite de tests, por exhaustiva que sea, cubre sólo los casos que sus autores imaginaron. La verificación formal y el fuzzing son dos herramientas que permiten acotar esa brecha, una por la vía de la prueba matemática y la otra por la exploración sistemática del espacio de entradas.

El punto es más relevante en el contexto de uso. Un agente de IA orquesta el solver: construye modelos complejos, los modifica en varios pasos y consulta los resultados para proponer nuevas acciones. En ese ciclo el solver recibe entradas que ningún ingeniero revisó antes de que llegaran al motor. Si el motor se comporta de forma incorrecta en algún caso de borde (una matriz de rigidez casi singular, una geometría degenerada, una combinación de cargas extrema), el agente puede producir y citar ese resultado sin advertencia. La frontera de determinismo protege la trazabilidad; la verificación formal protege la corrección del motor al que esa trazabilidad apunta.

4.2. Qué es la verificación formal

La verificación formal es el uso de prueba matemática para establecer que un sistema se comporta según su especificación para todas las entradas posibles, y no sólo para las que se eligieron testear. La distinción respecto del testing convencional es de fondo: el testing puede encontrar errores; la verificación formal puede demostrar su ausencia dentro del dominio especificado.

Las dos técnicas principales son el *model checking* y la demostración de teoremas. El *model checking* explora exhaustivamente todos los estados de un sistema finito y comprueba que una propiedad se cumple en cada uno, y es la herramienta natural para protocolos de comunicación y circuitos de hardware. La demostración de teoremas construye una prueba lógica formal con un asistente de pruebas como Lean 4, Coq o Isabelle; es más expresiva, maneja sistemas infinitos o continuos y es la técnica relevante para los algoritmos numéricos que componen un solver de elementos finitos.

Aplicada a un solver estructural, la verificación formal consiste en escribir una especificación matemática de las propiedades que el motor debe satisfacer (por ejemplo, que la matriz de rigidez global ensamblada es simétrica definida positiva para elementos bien formados, que la factorización de Cholesky produce una descomposición $K = LL^T$ exacta, o que la condición de equilibrio $\Sigma R = \text{carga aplicada}$ se cumple algebraicamente) y luego demostrar, con un asistente de pruebas, que la implementación satisface esa especificación. Si la prueba cierra, esas propiedades son garantías matemáticas y no conjeturas empíricas.

Este enfoque es coherente con la práctica de verificación formal de software: del mismo modo en que un compilador formalmente verificado garantiza propiedades del código que produce, un solver formalmente verificado garantiza propiedades de los resultados que entrega al agente.

4.3. Qué es el fuzzing y cómo complementa la verificación formal

El fuzzing es una técnica de testing que genera automáticamente grandes volúmenes de entradas (aleatorias, mutadas o construidas según una estrategia) y las ejecuta contra el sistema bajo prueba para detectar comportamientos incorrectos. A diferencia del testing convencional, no exige que el autor especifique casos concretos: el fuzzer recorre el espacio de entradas de forma sistemática y reporta las que violan una propiedad.

En un solver estructural la variante más relevante es el fuzzing basado en propiedades, donde en lugar de comparar la salida con un valor esperado se afirman invariantes matemáticos que deben cumplirse para cualquier entrada válida. Algunas propiedades naturales para un solver de elementos finitos son las siguientes.

1. Equilibrio global: para cualquier modelo bien formado, la suma de las reacciones iguala la carga total aplicada, $\Sigma R = P$, con independencia de la geometría, los materiales o las condiciones de apoyo.
2. Simetría de la matriz de rigidez: K es simétrica por construcción, $K = K^T$.
3. Definitud positiva: K es definida positiva para cualquier estructura estáticamente determinada o hiperestática sin mecanismos.
4. Invariancia ante transformaciones rígidas: desplazar o rotar un modelo no cambia los esfuerzos internos.

El fuzzer genera miles de modelos aleatorios válidos y comprueba estas propiedades en cada uno; cualquier modelo que las viole es un error. El enfoque es especialmente valioso para encontrar casos de borde numéricos (matrices de rigidez casi singulares, elementos con rigidez relativa muy grande o muy pequeña, geometrías degeneradas) que una suite de tests convencional rara vez cubre.

Una segunda variante útil es el fuzzing diferencial: correr el mismo modelo aleatorio en Stabileo y en una referencia de confianza, como OpenSees ² o soluciones analíticas para casos simples, y verificar que los resultados coinciden dentro de una tolerancia. Es particularmente útil para validar casos fuera de los benchmarks estándar y para detectar regresiones cuando el motor evoluciona.

Fuzzing y verificación formal son complementarios. El fuzzing encuentra los casos de borde que luego motivan la especificación formal: cuando el fuzzer halla una entrada que viola una propiedad, esa propiedad pasa a ser candidata para una prueba que garantice que no puede ocurrir.

5. Implementación en Stabileo

5.1. El motor de cálculo y su validación

El motor está escrito íntegramente en Rust, sin dependencias externas de álgebra lineal (la factorización de Cholesky, los ordenamientos, los eigensolvers y las rutinas matriciales están implementados desde cero), lo que permite compilarlo a WebAssembly y ejecutarlo localmente en el navegador del usuario, sin servidor de cálculo. Las capacidades del motor abarcan análisis lineal 2D y 3D, no-linealidad geométrica (P- Δ , pandeo, corotacional), dinámica (modal, espectral, integración temporal) ¹⁸ y una variedad de elementos (frame, reticulado, cable, placas y cáscaras). La suite de validación del motor reporta, en la última lectura del repositorio, del orden de 6.800 tests pasando con cero fallas, y se contrastó contra programas comerciales como STAAD.Pro sobre los mismos modelos, reproduciendo sus resultados con diferencias inferiores al 0,1 %, un desempeño en línea con el estado del arte.

5.2. Verificación normativa CIRSOC (alcance en hormigón y acero)

La normativa argentina presenta diferencias respecto de las normas internacionales de las que deriva. El CIRSOC 201-2005 ⁹, basado en el ACI 318-05, y el CIRSOC 301-2017 ¹⁰, basado en el AISC 360-10, incorporan adaptaciones al contexto local.

El módulo de hormigón armado (CIRSOC 201) está implementado tanto en el motor Rust como en la capa de verificación del frontend, con salida paso a paso auditable y cobertura de tests dedicada. Este cubre flexión simple y doble (con armadura mínima), corte, columnas por flexo-compresión mediante diagrama de interacción, esbeltez por amplificación de momento,

torsión, flexión biaxial, losas uni- y bidireccionales, y verificaciones de servicio. A modo de ejemplo, la armadura mínima de flexión cumple

$$A_{s,\min} = \max \left(\frac{0,25 \sqrt{f_c}}{f_y} \cdot bd; \frac{1,4}{f_y} \cdot bd \right) \quad (1)$$

el diseño de estribos por corte responde a

$$V_s = \frac{A_v f_y d}{s} \quad (2)$$

la amplificación de momento por esbeltez a

$$M_c = \delta_{ns} \cdot M_2 \quad (3)$$

con $\delta_{ns} = C_m / (1 - P_u / (0,75 P_c)) \geq 1$, y la flexión biaxial a la fórmula recíproca de Bresler:

$$\frac{1}{P_n} = \frac{1}{P_{nx}} + \frac{1}{P_{ny}} - \frac{1}{P_0} \quad (4)$$

El módulo de acero (CIRSOC 301) está implementado en TypeScript en la capa de verificación del frontend. No existe todavía un módulo equivalente en el motor Rust. El módulo consume los esfuerzos producidos por el solver y aplica las disposiciones del reglamento sobre la salida ya calculada (tracción, compresión con pandeo según KL/r, flexión con pandeo lateral-torsional, corte e interacción combinada H1). A diferencia del módulo de hormigón, esta lógica aún no cuenta con una batería de tests dedicada al momento de la redacción, por lo que su madurez es preliminar; se reporta así explícitamente para no atribuirle un grado de validación que todavía no tiene.

5.3. Las acciones tipadas del agente

El agente no es un asistente conversacional genérico, sino un conjunto de cuatro capacidades discretas, cada una con su contrato de entrada y salida:

1. Revisión de modelos, que produce hallazgos priorizados sobre el artefacto del solver.
2. Explicación de diagnósticos en lenguaje natural.
3. Construcción de modelos mediante *tool calling*.
4. Interpretación de resultados, que responde citando los casos gobernantes.

El constructor opera mediante un bucle conversacional. Primero, el usuario expresa su intención. Segundo, el agente clarifica y emite una llamada tipada (por ejemplo, una acción para crear un pórtico con su luz y su altura como parámetros). Tercero, el backend ejecuta un generador determinístico que la expande en un modelo completo. A continuación, el modelo aparece como vista previa y el usuario aprueba, modifica o cancela.

Hay nueve generadores deterministas que cubren las tipologías más frecuentes: vigas simples, voladizos, vigas continuas, pórticos planos y espaciales, marcos multi-vano y multi-piso, reticulados y pórticos de hormigón armado. Sobre un modelo existente, un conjunto de acciones de edición permite agregar un vano o un piso, cambiar las familias de secciones, cambiar la estrategia de apoyos, agregar o quitar cargas y eliminar elementos con validación de integridad estructural.

Un registro de capacidades en el backend describe cada acción con sus parámetros, tipos, rangos, valores por defecto y limitaciones, y funciona como fuente única de verdad, a saber: alimenta los prompts del modelo, valida las entradas antes de ejecutar la acción y describe la superficie disponible al frontend. El servicio admite modelos de varios proveedores (Claude, OpenAI, DeepSeek, Mistral, Kimi y Gemini) mediante adaptadores nativos y una interfaz compatible con OpenAI, con el contrato de *tool calling* normalizado entre todos.

5.4. Generación de la memoria de cálculo y los planos

El sistema genera memorias de cálculo en HTML con ecuaciones renderizadas en KaTeX, que incluyen dibujos SVG de secciones transversales con barras, estribos y cotas; elevaciones de vigas y columnas con distribución de armadura; diagramas de interacción P-M con el punto de demanda superpuesto; cómputo de materiales (volumen de hormigón, peso de acero, cuantía); detalles de nudo viga-columna; y planos de armadura de losa. Las memorias se exportan a PDF mediante la impresión nativa del navegador, y la geometría se exporta a DXF para el despiece.

6. Casos de estudio

Esta sección recorre el flujo integrado sobre casos trabajados. Los resultados estructurales y de verificación de §6.1 y §6.3 son salidas reales del solver y del módulo CIRSOC 201 de Stabileo, obtenidas ejecutando el motor sobre los modelos descritos.

6.1. Del pedido al diseño verificado: un pórtico de HA que se corrige cuando no verifica

El ingeniero describe la estructura objetivo en lenguaje natural. Por ejemplo, un pórtico de hormigón armado de 6 m de luz y 4 m de altura, con bases empotradas y una carga gravitatoria distribuida sobre la viga. El agente formula las preguntas de clarificación necesarias (¿acero u hormigón?, ¿bases empotradas o articuladas?, ¿se considera carga lateral?) y emite la acción que crea el pórtico, que el backend expande de forma determinística en un modelo de 4 nodos, 3 elementos de pórtico y 6 grados de libertad libres. Se adopta hormigón H-21 ($E \approx 21.500$ MPa), una viga inicial de 25×40 cm, columnas de 30×30 cm y una carga distribuida de 30 kN/m sobre la viga (Figura 2).

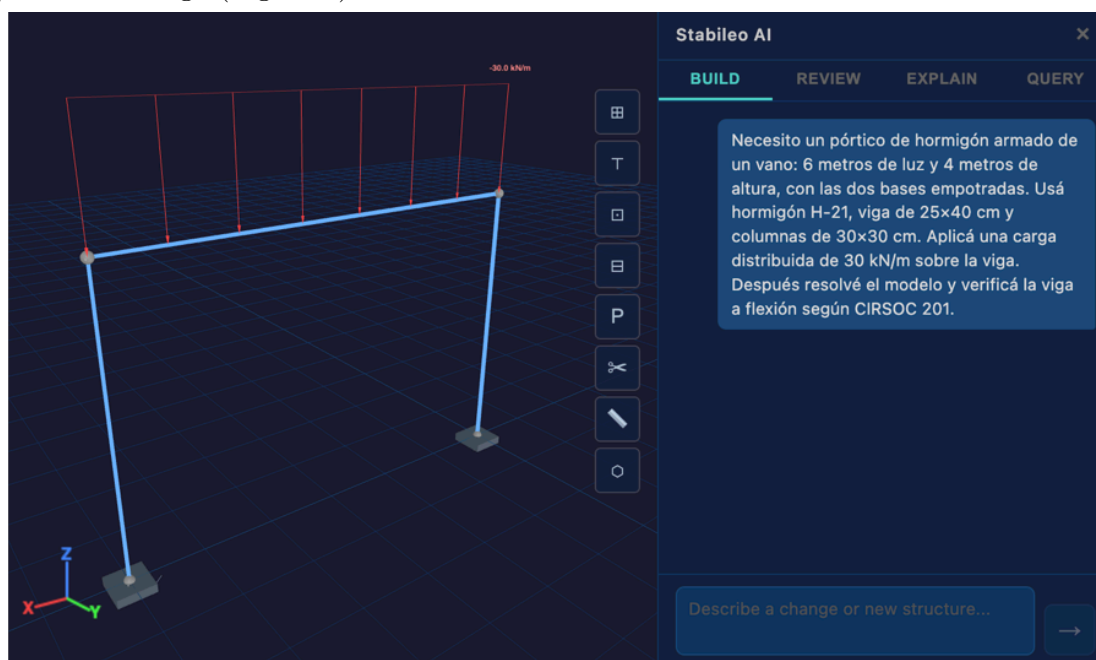


Figura 2. Modelado del pórtico de §6.1: vista 3D con la carga distribuida (izquierda) y el pedido en lenguaje natural en el AI Drawer (derecha).

Ejecutado el solver de Stabileo, las reacciones verticales son de 90 kN por columna y se cumple el equilibrio global $\Sigma R_z = 180 \text{ kN} = q \cdot L$ de forma exacta. El momento flector gobernante de la viga resulta $M_u = 80,8 \text{ kN} \cdot \text{m}$ en el centro de la luz. La relación demanda/capacidad (D/C) compara la solicitación con la resistencia de diseño, en este caso el cociente $M_u / \varphi M_n$, y el elemento verifica cuando $D/C \leq 1$. La verificación CIRSOC 201 a flexión de la viga 25×40

armada con 3Ø16 arroja una capacidad $\varphi M_n = 73,3 \text{ kN} \cdot \text{m}$, es decir $D/C = 1,10 > 1$: la viga no verifica. El agente explica el diagnóstico citando los valores producidos por el solver y propone aumentar la sección.

El ingeniero acepta y la viga pasa a 30×55 cm. Al re-resolver el modelo, el momento gobernante de la viga sube a 105,6 kN · m; la nueva capacidad $\varphi M_n = 108,6 \text{ kN} \cdot \text{m}$ da $D/C = 0,97 \leq 1$: ahora verifica (Tabla 2). Una verificación manual independiente reproduce estos esfuerzos y capacidades, lo que respalda la trazabilidad del resultado. El punto central no es la magnitud de los esfuerzos, sino que cada número citado por el agente corresponde a una salida específica del solver verificado, y que la decisión de aumentar la sección la toma y la firma el ingeniero.

Estado	Sección viga	Armadura	M_u [kN·m]	φM_n [kN·m]	D/C	Verifica
Inicial	25×40 cm	3Ø16	80,8	73,3	1,10	No
Corregido	30×55 cm	3Ø16	105,6	108,6	0,97	Sí

Tabla 2. Verificación a flexión de la viga del pórtico de §6.1, antes y después de la corrección.

Conviene notar que el momento de demanda aumenta de 80,8 a 105,6 kN · m al agrandar la viga: en un pórtico hiperestático una sección más rígida atrae mayor momento, de modo que la verificación correcta exige volver a resolver el modelo y no sólo recalcular la capacidad de la nueva sección. Es precisamente en este punto donde tener el solver en el bucle aporta valor, y donde un agente que estimara el efecto sin recalcular incurriría en error.

6.2. Exploración rápida de variantes

Una vez que el modelo existe, probar alternativas es bastante directo. Cambiar la familia de secciones, modificar la estrategia de apoyos o agregar una carga lateral por nivel se expresan como acciones de edición y se ejecutan en el navegador en segundos. Sobre el pórtico de §6.1 se aplicaron tres variantes y, para cada una, se contó el número de acciones de cada flujo, clasificadas en decisiones de ingeniería y acciones mecánicas de re-modelado (Tabla 3). En las tres, el agente requiere una sola acción de decisión y ninguna acción mecánica, mientras que el flujo convencional requiere la misma decisión más cinco a siete acciones mecánicas, con idéntico resultado estructural. La conclusión defendible no es de tiempo ni de mejor diseño: a igual decisión de ingeniería y mismo resultado, el agente elimina la entrada mecánica repetitiva.

Variante	Decisión (agente)	Mecánicas (agente)	Decisión (conv.)	Mecánicas (conv.)	Resultado (D/C)
Cambiar viga 30×55 → 35×65	1	0	1	5–7	0,97 → 0,88
Bases empotradas → articuladas	1	0	1	5–7	1,02 (no verifica)
Agregar carga lateral (10 kN/nivel)	1	0	1	5–7	0,97

Tabla 3. Acciones por variante: flujo con agente frente al flujo convencional, sobre el pórtico de §6.1.

6.3. Modificar un proyecto ya calculado: eliminar una columna

El valor del agente se vuelve más nítido cuando el proyecto ya está calculado y aparece un cambio. Se parte de un pórtico de hormigón armado de cuatro vanos de 5 m y tres pisos de 3 m, con bases empotradas, hormigón H-21, vigas de 25×50 cm (4Ø16), columnas de 35×35 cm

y una carga distribuida de 25 kN/m sobre todas las vigas (20 nodos, 27 elementos, 45 grados de libertad libres). Resuelto y verificado, todas las vigas cumplen CIRSOC 201 a flexión con holgura: la viga gobernante tiene $D/C = 0,46$.

Por una decisión arquitectónica se elimina una columna interior de la planta baja. El ingeniero lo indica en lenguaje natural; la acción de edición la quita con validación de integridad estructural y el modelo se vuelve a resolver, manteniéndose el equilibrio global ($\Sigma R_z = 1500$ kN = carga total) antes y después. Las vigas que apoyaban en esa columna pasan a salvar 10 m y, junto con las que quedan alineadas encima, redistribuyen el apoyo perdido: seis vigas superan $D/C = 1$ y el momento de demanda de la viga gobernante casi se cuadruplica, de 54 a 204 kN · m (D/C de 0,43 a 1,63; Figura 3). El agente señala las vigas que dejan de verificar y propone una sección de transferencia más profunda.

Flujo del agente: modificar un proyecto ya calculado y re-verificar (§5.3)

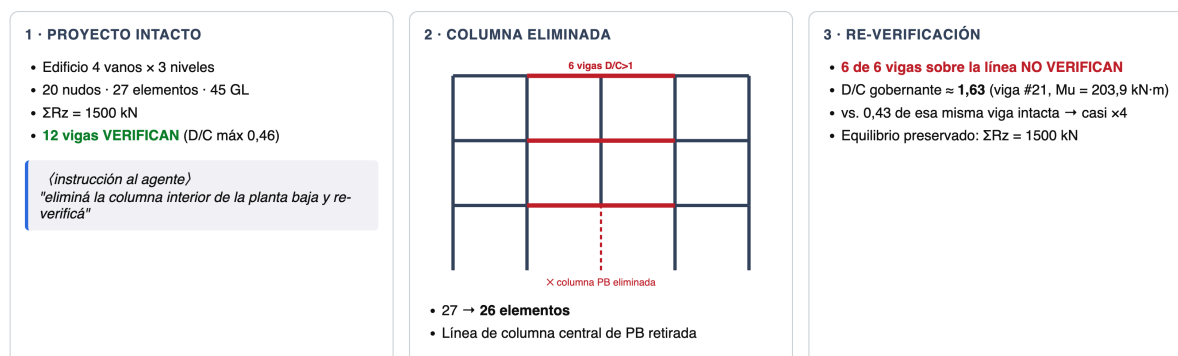


Diagrama del flujo de §5.3. Los esfuerzos, las relaciones D/C y las reacciones son salidas reales del motor y del módulo CIRSOC 201 de Stabileo (caso de prueba engine/tests/paper_examples.rs::a3_building_column_removal); el esquema del pórtico reproduce la geometría real del modelo.

Figura 3. Flujo de §6.3: proyecto intacto, eliminación de una columna interior de planta baja y re-verificación (seis vigas dejan de verificar).

Al adoptar 30×65 cm (5Φ20) en las vigas afectadas y re-resolver, todas vuelven a verificar ($D/C \leq 0,61$; Tabla 4). Lo relevante es el flujo: un cambio que manualmente exige re-modelar, re-analizar, re-verificar y re-detallar muchos elementos se resuelve como una conversación corta, con cada número trazable al solver y sujeto a la aprobación del ingeniero.

Estado de la viga gobernante	Sección	M_u [kN·m]	φM_n [kN·m]	D/C	Verifica
Intacto	25×50 (4Φ16)	54,2	125,3	0,43	Sí
Tras quitar la columna	25×50 (4Φ16)	203,9	125,3	1,63	No
Re-dimensionado	30×65 (5Φ20)	189,8	313,7	0,61	Sí

Tabla 4. Viga gobernante del edificio de §6.3 en los tres estados: intacto, tras quitar la columna y re-dimensionada.

6.4. De los resultados a los planos (memoria, CAD, despiece)

Del modelo verificado salen directamente los entregables. La memoria de cálculo reúne datos del modelo, resultados del análisis, verificaciones por elemento con ecuaciones, dibujos SVG de armaduras, diagrama de interacción P-M y cómputo de materiales. La geometría se exporta a DXF para el despiece. El recorrido se cierra de la intención al plano sin instalar software ni escribir código (Figura 4).

Memoria de cálculo — CIRSOC 201 · Viga §5.1 corregida (30×55 cm, 3Ø16)

H-21 ($f'_c=21$ MPa) · ADN-420 ($f_y=420$ MPa) · L=6.00 m · recubrimiento 2.5 cm · d=0.50 m

Flexión

- $M_u = 105.56$ kN·m (solver 2D real)
- $A_{s,req}$ (tracción) = 5.74 cm²
- Armadura provista: **3 Ø16 = 6.03 cm²** ($\geq A_{s,req}$ ✓)
- a (bloque de Whitney) = 4.73 cm
- $\epsilon_t \geq 5\text{‰} \rightarrow$ F.C.T. $\rightarrow \phi = 0.90$
- $\phi M_n = 108.61$ kN·m

Corte

- $V_u = 90.00$ kN
- Estribos: 2 ramas Ø8 c/25 cm
- $\phi V_n = 152.41$ kN

D/C flexión = $0.972 \leq 1 \rightarrow$ VERIFICA

D/C corte = $0.591 \leq 1 \rightarrow$ VERIFICA

Capacidades $\phi M_n/\phi a$: módulo CIRSOC 201 nativo
(check_cirsoc201_members). Demanda M_u/V_u : solver 2D. Corte: diseño
app CIRSOC. Dibujo: funciones SVG de la app.

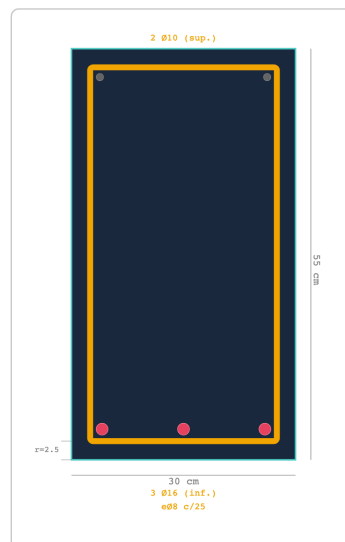


Figura 4. Memoria de cálculo de Stabileo para la viga corregida de §6.1 (30×55 cm, 3Ø16): verificación CIRSOC 201 a flexión y corte.

6.5. Diagnóstico de un modelo mal planteado

El sistema también es útil cuando el modelo está mal planteado. Las compuertas de validación rechazan modelos malformados (elementos que referencian nodos inexistentes, ausencia de apoyos que deja un mecanismo) y mantienen el modelo previo sin cambios, mostrando el error preciso. Ante una estructura inestable, el solver lo señala. La capacidad de revisión produce hallazgos priorizados con severidad, elementos afectados y sugerencias de corrección, y la de explicación traduce un código de diagnóstico en una guía en lenguaje natural. El agente convierte la falla en retroalimentación accionable, en lugar de devolver silenciosamente un número incorrecto, que es justamente el riesgo que la frontera de determinismo busca evitar.

7. Discusión: impacto en la práctica profesional

Conviene leer este trabajo como una continuidad y no como una ruptura: el CAD se llevó la parte mecánica del dibujo y la planilla de cálculo la de la aritmética repetitiva, sin llevarse el criterio; el agente es el eslabón siguiente, que absorbe el andamiaje mecánico (modelar, re-modelar, documentar) pero deja el juicio del lado del ingeniero. El acceso de código abierto y la ejecución en el navegador bajan la barrera de entrada para estudiantes, estudios pequeños y la academia, y el modo educativo del sistema, que guía al estudiante por las etapas del análisis con las ecuaciones a la vista, extiende ese beneficio a la enseñanza; aun así conviene ser preciso, porque bajar la barrera para correr un análisis no baja la barrera para interpretarlo, de modo que la herramienta democratiza el acceso y no la pericia. El riesgo principal es la sobreconfianza, ya que una explicación fluida puede transmitir una seguridad que el modelo no justifica; por eso el sistema mantiene visible la frontera entre lo que afirmó el solver y lo que redactó el agente, con etiquetado de confianza, aceptación humana, trazabilidad y degradación segura. Ninguna de esas medidas transfiere la responsabilidad: el número firmado sigue siendo del ingeniero.

8. Limitaciones y trabajo futuro

El agente conversacional opera hoy a nivel de asistencia: genera vigas, pórticos regulares (2D y 3D), reticulados y marcos multi-vano y multi-piso a partir de lenguaje natural, y soporta acciones de edición sobre modelos existentes. Sin embargo, no itera automáticamente un diseño buscando la solución más liviana o económica. El bucle solver-en-el-bucle (en el que el agente genera una propuesta, la resuelve, evalúa y modifica sin intervención humana) está previsto en la hoja de ruta, pero no está implementado. La verificación CIRSOC 201 está implementada y testeada, pero el módulo CIRSOC 301 reside sólo en la capa de frontend en TypeScript y aún no cuenta con cobertura de tests dedicada. La cobertura normativa nativa se limita a CIRSOC 201 y 301; otras normas (Eurocode, ACI, AISC) son trabajo futuro. La personalización de plantillas de memoria por estudio profesional aún no está expuesta.

En cuanto a la confiabilidad del propio motor de cálculo, la incorporación de la verificación formal y el fuzzing presentados en §4 se plantea en tres etapas de complejidad y horizonte crecientes. A corto plazo, el paso inmediato es incorporar testing basado en propiedades con Proptest, una biblioteca de fuzzing para Rust, afirmando los invariantes de equilibrio, simetría y definitud positiva sobre modelos generados aleatoriamente. Esta extensión de la suite de validación existente no requiere cambios arquitectónicos y aumenta de forma significativa la cobertura del espacio de entradas más allá de los benchmarks actuales.

A mediano plazo, el objetivo es una especificación formal del núcleo de ensamble de la matriz de rigidez y de la factorización de Cholesky en Lean 4, con una prueba de que la implementación la satisface. Es un trabajo natural como tema de tesis en FIUBA, en la intersección de la ingeniería civil, la informática y los métodos formales. La implementación en Rust facilita la tarea, porque su sistema de tipos ya descarta clases enteras de errores de memoria y existe tooling creciente para la verificación de Rust, como Verus y Kani.

A largo plazo, la visión es un solver cuyo núcleo numérico crítico esté formalmente verificado y cuya capa de validación de entradas esté cubierta por fuzzing diferencial continuo contra referencias. En ese escenario, la frontera de determinismo deja de ser una práctica de ingeniería para convertirse en una garantía matemática: el agente orquesta un motor probado, las acciones tipadas se validan antes de ejecutarse y cada número citado en la memoria de cálculo es trazable a un cómputo cuya corrección está demostrada y no sólo testeada.

9. Conclusiones

En la práctica contemporánea, el cálculo nunca fue el cuello de botella: la resolución de sistemas lineales es rápida y está bien comprendida. El tiempo del ingeniero se va en el andamiaje que la rodea: modelar, rehacer, verificar y documentar. Un agente de IA puede comprimir ese andamiaje si, y sólo si, nunca origina el número: orquesta un solver verificado que calcula y módulos normativos que verifican, y documenta citando esos resultados.

Los aportes de este trabajo son la frontera de determinismo como principio arquitectónico, su materialización en una plataforma de código abierto que combina solver browser-native, verificación CIRSOC y un agente de IA orquestador, y una demostración de extremo a extremo del flujo, ilustrada con un caso resuelto cuyos esfuerzos y verificaciones son salidas reales del solver, presentada con un relato honesto de lo validado y lo pendiente. La incorporación de inteligencia artificial generativa al cálculo estructural es viable cuando se apoya en herramientas verificadas y conserva la trazabilidad que la práctica profesional exige. Lo que este trabajo muestra es la posibilidad técnica concreta de esa orquestación, sobre una base abierta, auditable y operable directamente en el navegador.

Referencias

1. Computers & Structures Inc. *SAP2000 Structural Analysis Program*. Berkeley, CA, 2024.
2. Mazzoni, S., McKenna, F., Scott, M. H., Fenves, G. L. *OpenSees: Open System for Earthquake Engineering Simulation*. PEER, University of California, Berkeley, 2000.
3. Électricité de France. *Code_Aster: Analysis of Structures and Thermomechanics for Studies and Research*. <https://>
4. Brown, T. B. et al. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.
5. OpenAI. GPT-4 Technical Report. arXiv:2303.08774, 2023.
6. Geng, Z., Liu, J., Cao, R., Cheng, L., Frangopol, D. M., Cheng, M. A Novel Multi-Agent Architecture to Reduce Hallucinations of Large Language Models in Multi-Step Structural Modeling. arXiv:2603.07728, 2026.
7. Geng, Z., Liu, J., Cao, R., Cheng, L., Wang, H., Cheng, M. A Lightweight Large Language Model-Based Multi-Agent System for 2D Frame Structural Analysis. arXiv:2510.05414, 2025.
8. Liu, J., Geng, Z., Cao, R., Cheng, L., Bocchini, P., Cheng, M. A Large Language Model-Empowered Agent for Reliable and Robust Structural Analysis. arXiv:2507.02938, 2025.
9. INTI-CIRSOC. *Reglamento CIRSOC 201-2005: Reglamento Argentino de Estructuras de Hormigón*. Buenos Aires, 2005.
10. INTI-CIRSOC. *Reglamento CIRSOC 301-2017: Reglamento Argentino de Estructuras de Acero*. Buenos Aires, 2017.
11. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y. ReAct: Synergizing Reasoning and Acting in Language Models. *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2210.03629.
12. Schick, T. et al. Toolformer: Language Models Can Teach Themselves to Use Tools. *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, pp. 68539–68551. arXiv:2302.04761.
13. Patil, S. G., Zhang, T., Wang, X., Gonzalez, J. E. Gorilla: Large Language Model Connected with Massive APIs. arXiv:2305.15334, 2023.
14. Gao, L. et al. PAL: Program-aided Language Models. *Proc. 40th International Conference on Machine Learning (ICML)*, PMLR 202, pp. 10764–10799, 2023. arXiv:2211.10435.
15. Abueidda, D. W. et al. Topology optimization of 2D structures with nonlinearities using deep learning. *Computers & Structures*, vol. 237, 106283, 2020.
16. Pfaff, T. et al. Learning Mesh-Based Simulation with Graph Networks. *Proc. International Conference on Learning Representations (ICLR)*, 2021.
17. Li, Z. et al. Physics-Informed Neural Operator for Learning Partial Differential Equations. *ACM/JMS Journal of Data Science*, 2024.
18. Chopra, A. K. *Dynamics of Structures: Theory and Applications to Earthquake Engineering*. 5th ed. Pearson, 2017.

Datos de los autores

CHESTA, Bautista

Facultad de Ingeniería de la Universidad de Buenos Aires (FIUBA), LambdaClass y Ergodic Group

bchesta@fi.uba.ar

BERTERO, Raúl

Facultad de Ingeniería de la Universidad de Buenos Aires (FIUBA)

rbertero@fi.uba.ar

CARRONE, Federico
LambdaClass y Ergodic Group
federico@lambdaclass.com

KINGSTON, Diego
Facultad de Ingeniería de la Universidad de Buenos Aires (FIUBA), LambdaClass y Ergodic
Group
dkingston@fi.uba.ar