

UNIVERSIDAD CAECE

Proyecto Final - 2026

Título

Pucará: diseño y evaluación de un lenguaje IA-nativo compacto y verificable para la síntesis automática de estrategias de trading

Implementación del núcleo, adaptación de un modelo local y análisis de economía de tokens y fidelidad especificación-programa

Carrera

Ingeniería en Sistemas

Materia

Proyecto Final de Ingeniería

Autor

Federico Nicolas Lussoro

Docente y Director

Dr. Miguel Méndez Garabetti

Mar del Plata, 2026

Resumen

El crecimiento de los modelos de lenguaje aplicados a programación plantea una situación poco explorada por el diseño tradicional de lenguajes: una parte creciente del código ya no se produce mediante escritura manual, sino a partir de especificaciones en lenguaje natural. Sin embargo, los modelos continúan generando principalmente lenguajes concebidos para personas. Esa elección introduce tokens orientados a la lectura humana, estructuras más generales que las que el dominio requiere y modos de error que exigen corrección posterior. Esta tesis presenta Pucará, un lenguaje específico de dominio para estrategias de trading, diseñado desde el principio para ser generado por inteligencia artificial. El prototipo implementado utiliza expresiones simbólicas en notación prefija e índices de De Bruijn, un vocabulario de 38 primitivas, análisis estático de warm-up, semántica causal y un intérprete determinista en Rust. La cadena se completa con un backtester determinista con costos de transacción, un proceso de evidencia (e-process) para la validación, un mecanismo de integridad del núcleo y una biblioteca de abstracciones obtenida con Stitch. La evaluación se basa en Qwen2.5-Coder-7B y 40 especificaciones de estrategia. Se generaron 6.120 muestras para un brazo con el lenguaje estático y 6.120 para un brazo con adaptación conjunta del modelo, la biblioteca y el tokenizador. El brazo estático alcanzó un $\text{pass}@5$ sintáctico de 0,8999 y un $\text{pass}@5$ de supervivencia —entendida en todo el documento como supervivencia bajo los criterios definidos en el mundo sintético experimental— de 0,7726 en las 30 tareas no vistas, mientras que el brazo coevolutivo obtuvo 0,4374 y 0,3579, respectivamente. Sin embargo, cuando produjo programas válidos, su representación utilizó en promedio un 34,2 % menos de tokens BPE. La mejora de compactación no se trasladó al costo total: el brazo coevolutivo consumió 2,34 veces más tokens generados y 8,43 veces más tokens por programa superviviente, debido a terminaciones incompletas y secuencias degeneradas. La separación central del trabajo aparece en el análisis de fidelidad: al exigir

además la implementación fiel de la especificación, el pass@5 held-out se redujo de 0,7726 a 0,0331 y las aceptaciones de 2.275 a 97. De los 2.785 programas aceptados por los criterios del mundo sintético en el brazo estático, solo 106 satisfacían los seis atributos estructurales requeridos por la especificación. Esto muestra que la validez sintáctica, la ejecución y el rendimiento simulado no bastan para garantizar que un programa implemente lo que se pide.

La contribución del trabajo consiste, por lo tanto, en demostrar la viabilidad de un lenguaje IA-nativo compacto y verificable, medir con precisión la diferencia entre compactación y eficiencia extremo a extremo, y aportar un procedimiento reproducible para distinguir corrección de ejecución y fidelidad a la especificación.

Palabras clave: lenguajes de dominio específico; generación de código; modelos de lenguaje; síntesis de programas; tokenización; trading sistemático; verificabilidad; aprendizaje de bibliotecas.

Agradecimientos

El autor agradece profundamente a su padre y futuro colega, Ing. Pablo J. Lussoro, sin duda la persona más inteligente que conoció en su vida, quien le dio “clases particulares” durante toda la carrera y lo guió y aconsejó como ingeniero y, más importante aún, como padre a lo largo de todos estos años; a su siempre amorosa madre, Karin V. Arenas, por su apoyo constante, su cariño y su dedicación, y con quien entendió que no todos los problemas se resuelven con matemática; a sus hermanos Agustina, Santiago, Carolina y Julieta, de cada uno de los cuales aprendió una forma distinta de ver el mundo; a sus abuelas Susana e Hilda, que hoy lo acompañan desde el Cielo en este logro, y a su abuelo Osvaldo, quienes le dieron amor, capricho y complicidad; a sus eternos amigos del club Malgor Track & Field; a su querida amiga y profesora de inglés, Lucía; y a Leonardo, su amigo desde la infancia, por el cariño y el acompañamiento de tantos años: sin ellos, transitar esta etapa habría sido imposible. El recorrido lo formó no solo como ingeniero, sino también como persona, algo de lo que está profundamente orgulloso y por lo que quedará eternamente agradecido. Agradece asimismo al Dr. Miguel Méndez Garabetti, por la dirección y la revisión de este trabajo, y a la Universidad CAECE, por la formación recibida.

<i>Resumen</i>	2
<i>Agradecimientos</i>	4
<i>Índice de contenidos</i>	9
<i>Índice de figuras</i>	9
<i>Índice de tablas</i>	10
1. Introducción	11
1.1 Propósito del proyecto	11
1.2 Problemática	11
1.3 Pregunta de investigación	12
1.4 Objetivos	13
1.5 Alcance y exclusiones	14
1.6 Organización del documento.....	14
2. Marco teórico	15
2.1 Modelos de lenguaje y generación de código	15
2.2 Tokenización y costo de representación	16
2.3 Lenguajes de dominio específico	16
2.4 Síntesis de programas y aprendizaje de bibliotecas	17
2.5 Verificación determinista y causalidad	17
2.6 Trading sistemático y evaluación por simulación	18
2.7 Definiciones operativas	18
3. Estado del arte	20
3.1 Generación de código en lenguajes de propósito general	20

3.2 DSL y representaciones orientadas al dominio	20
3.3 Aprendizaje de bibliotecas.....	21
3.4 Vocabularios extendidos y lenguajes IA-nativo.....	22
3.5 Verificación de código generado	23
3.6 Brecha abordada por Pucará	24
4. Diseño e implementación de Pucará.....	26
4.1 Descripción general.....	26
4.2 Principios de diseño IA-nativo	27
4.3 Representación canónica	27
4.4 Núcleo y vocabulario.....	28
4.5 Lexer, parser y round-trip.....	29
4.6 Intérprete y semántica numérica	29
4.7 Warm-up y causalidad.....	30
4.8 Backtester y criterios de evaluación	30
4.9 Frozen core e integridad	32
4.10 Corpus y biblioteca aprendida.....	32
4.11 Tokenizador extendido y modelo	33
4.12 Reproducibilidad.....	33
4.13 Componentes diseñados no implementados.....	33
5. Metodología experimental y evaluación.....	35
5.1 Formulación del problema	35
5.2 Diseño experimental.....	35

5.3 Tareas y particiones	36
5.4 Brazos comparados	37
5.5 Métricas.....	39
5.6 Resultados de generación.....	40
5.7 Economía de tokens.....	43
5.8 Fidelidad especificación-programa.....	46
5.9 Caso auditado t20	49
5.10 Diagnóstico del brazo coevolutivo.....	52
5.11 Limitaciones	53
6. Pruebas y validación.....	55
6.1 Estrategia de validación.....	55
6.2 Suite automatizada.....	55
6.3 Oráculos y pruebas diferenciales	56
6.4 Integridad y portabilidad	56
6.5 Validación de la vitrina.....	56
6.6 Alcance de las pruebas.....	57
7. Infraestructura experimental y reproducibilidad	58
7.1 Entornos de ejecución	58
7.2 Artefactos y digests.....	58
7.3 Repositorio y versionado	58
7.4 Procedimiento de reproducción	59
7.5 Restricciones de acceso	60

8. Conclusiones	61
8.1 Síntesis del trabajo	61
8.2 Decisiones técnicas clave	62
8.3 Contribuciones.....	62
8.4 Trabajo futuro	64
8.5 Supuestos y dependencias	65
8.6 Restricciones y limitaciones.....	66
9. Declaración sobre el uso de inteligencia artificial	67
9.1 Herramientas utilizadas.....	67
9.2 Finalidad de uso.....	67
9.3 Aporte propio.....	67
9.4 Salvaguardas y responsabilidad.....	67
Referencias.....	68
Apéndices.....	72
Apéndice A. Glosario	72
Apéndice B. Acrónimos.....	72
Apéndice C. Gramática y vocabulario	73
Apéndice D. Métricas y fórmulas.....	74
Apéndice E. Artefactos y comandos de reproducción	74

Índice de contenidos

Índice de figuras

Figura 1. Flujo de síntesis y verificación de Pucará.	10
Figura 2. Arquitectura lógica de Pucará.	26
Figura 3. Partición temporal del mundo sintético instrumentado.	37
Figura 4. Funnel de evaluación F0.5.	41
Figura 5. Resultados held-out de síntesis.	42
Figura 6. Compactación de la representación válida.	44
Figura 7. Costo de generación extremo a extremo.	45
Figura 8. Fidelidad estructural sobre los programas supervivientes del brazo B.	47
Figura 9. Curvas de capital de los 21 programas distintos de la tarea t20.	51

Índice de tablas

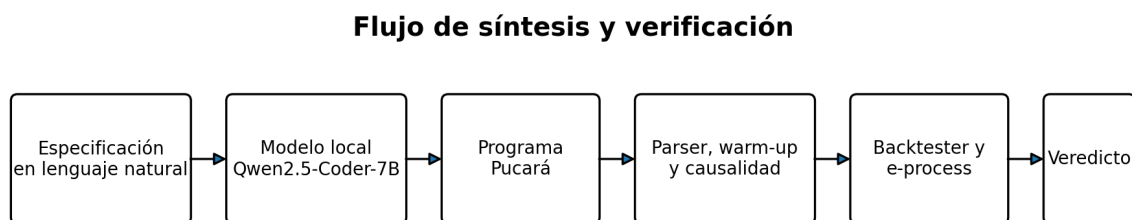
Tabla 1. Objetivos específicos y evidencia asociada.	13
Tabla 2. Definición de las métricas principales.....	18
Tabla 3. Estado del arte: antecedentes, adaptaciones y contribución original.	25
Tabla 4. Componentes principales del artefacto.....	26
Tabla 5. Categorías funcionales del núcleo actual.....	28
Tabla 6. Comparación de los brazos experimentales.	35
Tabla 7. Hiperparámetros de entrenamiento y generación.	39
Tabla 8. Resultados globales del funnel F0.5.	40
Tabla 9. Resultados pass@5 sobre tareas no vistas.	41
Tabla 10. Auditoría de economía de tokens.	44
Tabla 11. Fidelidad estructural por eje.	46
Tabla 12. Verificación del artefacto final.	55
Tabla 13. Afirmaciones principales y evidencia que las sostiene.	63
Tabla 14. Principales limitaciones y efecto sobre las conclusiones.	66
Tabla 15. Glosario de términos.....	72
Tabla 16. Acrónimos y abreviaturas.....	72
Tabla 17. Inventario de artefactos y digests principales.....	74

1. Introducción

1.1 Propósito del proyecto

Pucará es un lenguaje de dominio específico para expresar estrategias de trading sistemático mediante programas destinados a ser generados por modelos de lenguaje. La propuesta parte de una inversión del supuesto habitual del diseño de lenguajes: el código no se optimiza para su escritura manual, sino para su producción automática, su ejecución determinista y su verificación. La persona define la estrategia y sus restricciones; el modelo produce una representación ejecutable; el sistema comprueba la forma del programa, su disponibilidad temporal y su comportamiento bajo un procedimiento de simulación.

El lenguaje se desarrolló como un artefacto experimental completo y no como una notación aislada. Incluye un analizador léxico, un parser, una forma canónica, un intérprete, análisis de warm-up, un backtester, un ledger de candidatos, oráculos independientes, mecanismos de integridad y una infraestructura para ajustar un modelo especializado.



El modelo propone; el núcleo determinista y los verificadores deciden.

Figura 1. Flujo de síntesis y verificación de Pucará.

Fuente: elaboración propia sobre la arquitectura implementada.

1.2 Problemática

Los modelos de lenguaje especializados en código aprendieron sobre grandes volúmenes de software escrito para personas. Cuando se solicita una implementación, generalmente producen Python, JavaScript, Rust u otro lenguaje general. Estos lenguajes

tienen ecosistemas maduros, pero también introducen elementos que no expresan directamente la intención de una estrategia: importaciones, nombres, estructuras de control generales, conversión de tipos, manejo explícito de índices y convenciones de estilo. Cada uno de estos elementos se envía al modelo, se regenera y se factura como parte de la secuencia.

El costo no es el único problema. Un lenguaje general permite demasiadas formas de expresar la misma operación y demasiadas formas de producir código inválido o semánticamente inapropiado. En el trading, por ejemplo, un programa puede ejecutarse sin errores y producir una curva de capital atractiva incluso si utiliza información futura, juzga mal el riesgo o implementa una señal diferente a la especificada. La evaluación no puede reducirse a si el código compila o si el backtest es positivo. Pucará intenta abordar este problema mediante un núcleo reducido en notación prefija y una representación cerrada sobre un conjunto de primitivas de dominio. La reducción de la superficie sintáctica busca acortar los programas y facilitar el control. Sin embargo, la compactación textual por sí sola no garantiza ahorros operativos.

Si el modelo necesita más intentos para completar una expresión, el costo total puede aumentar. Esta distinción entre longitud de una solución y costo para obtenerla es uno de los ejes centrales de la evaluación.

1.3 Pregunta de investigación

¿En qué medida un lenguaje de dominio específico diseñado para ser generado por una inteligencia artificial permite expresar y verificar estrategias de trading de forma compacta, y cuáles son los límites observados al adaptar conjuntamente el modelo y su vocabulario?

La pregunta tiene dos dimensiones. La primera es inherente al propio lenguaje: densidad, determinismo, causalidad y verificabilidad. La segunda es el generador: la capacidad de aprender el lenguaje, generar las expresiones completas y utilizar un vocabulario ampliado. El diseño experimental separa ambas dimensiones para no atribuir al lenguaje fallas que pertenecen al generador, y para detectar cuándo la compactación aparente se pierde por reintentos fallidos.

1.4 Objetivos

Objetivo general. Diseñar, implementar y evaluar Pucará como un lenguaje orientado a la inteligencia artificial que sintetiza programas de estrategias de trading a partir de especificaciones en lenguaje natural, de manera compacta, determinista y verificable.

Los objetivos específicos se formularon de manera que cada uno pudiera verificarse mediante un artefacto o una medida concreta. La Tabla 1 relaciona cada objetivo con la evidencia utilizada en el resto de la memoria.

Tabla 1. Objetivos específicos y evidencia asociada.

Objetivo específico	Evidencia de cumplimiento
Implementar un núcleo ejecutable, una representación canónica y un análisis temporal.	Código Rust, parser/printer, intérprete, warm-up y suite automatizada.
Comprobar que un modelo local de 7B puede generar programas Pucará.	Brazo B de F0.5: validez por muestra y pass@5 sobre 40 tareas.
Medir la compactación obtenida por una biblioteca aprendida y un tokenizador extendido.	Compresión estructural de Stitch y auditoría BPE sobre 12.240 salidas.
Distinguir validez sintáctica, supervivencia bajo los criterios del mundo sintético experimental y fidelidad a la especificación.	Funnel de evaluación y detectores estructurales sobre 2.785 supervivientes.
Evaluar la adaptación conjunta del modelo y el vocabulario.	Comparación compute-matched entre brazos B y C bajo un gate prerregistrado y diagnóstico post-hoc.
Garantizar trazabilidad y reproducibilidad.	Commits, tags, digests, manifiestos, oráculos y reejecución t20.

Fuente: elaboración propia.

1.5 Alcance y exclusiones

El alcance comprende el prototipo experimental del lenguaje, la cadena de síntesis, la simulación determinista y la evaluación de dos configuraciones del mismo modelo base. El dominio se limita a estrategias sistemáticas sobre series temporales y posiciones multiactivo en un mundo sintético controlado. El objetivo no es descubrir estrategias rentables ni hacer recomendaciones financieras, sino evaluar el proceso de representación y síntesis. La ejecución con capital real, la demostración de rentabilidad, la validación en datos históricos del mercado, un compilador nativo, un sistema de tipos refinado, la ejecución en WebAssembly o FPGA, y una interfaz de usuario quedan fuera del alcance evaluado en esta tesis; varios de esos componentes integran el diseño a largo plazo que se discute en el capítulo 8, pero no forman parte del resultado aquí medido.

La tesis tampoco afirma que cada primitiva central tenga un token BPE: la auditoría final muestra 29 primitivas atómicas de 38, además de 14 abstracciones aprendidas que se introducen como tokens atómicos.

1.6 Organización del documento

El capítulo 2 introduce los conceptos necesarios para interpretar el lenguaje y las métricas. El capítulo 3 revisa antecedentes de generación de código, DSL, aprendizaje de bibliotecas y verificación. El capítulo 4 describe la arquitectura y la implementación de Pucará. El capítulo 5 presenta la metodología experimental, los resultados y su análisis. El capítulo 6 separa las pruebas del software de la evaluación científica. El capítulo 7 documenta los entornos, artefactos y procedimientos de reproducción. El capítulo 8 sintetiza las contribuciones, limitaciones y líneas futuras. El capítulo 9 declara el uso de herramientas de inteligencia artificial. Las referencias y los apéndices completan la trazabilidad técnica.

2. Marco teórico

2.1 Modelos de lenguaje y generación de código

Los modelos de lenguaje autoregresivos estiman la probabilidad del siguiente token condicionado por la secuencia previa. La arquitectura Transformer permite modelar dependencias a gran escala mediante mecanismos de atención (Vaswani et al., 2017). El aumento del tamaño del modelo y del corpus produjo sistemas capaces de completar texto, seguir instrucciones y generar código (Brown et al., 2020). En programación, la salida se evalúa por corrección funcional y no por similitud textual; dos programas diferentes pueden resolver la misma tarea y una respuesta plausible puede fallar al ejecutarse.

HumanEval y el trabajo sobre Codex formalizaron el uso de muestreo repetido y de $\text{pass}@k$ para medir la probabilidad de obtener al menos una solución correcta entre k intentos (Chen et al., 2021). El enfoque es pertinente para Pucará porque el generador es estocástico y cada tarea admite múltiples programas. El valor $\text{pass}@5$ no equivale a una tasa de acierto de muestras individuales: se calcula por tarea y luego se promedia. La tesis mantiene separadas ambas cantidades.

Los estudios de síntesis con modelos de lenguaje muestran que la capacidad de generar programas mejora con escala, ajuste supervisado y retroalimentación, pero también que los modelos tienen dificultades con cadenas largas de operaciones y con restricciones semánticas (Austin et al., 2021). Pucará reduce el espacio mediante un DSL y delega la decisión final en un sistema determinista.

2.2 Tokenización y costo de representación

Un token BPE es una unidad del vocabulario del modelo. No coincide necesariamente con una palabra, un carácter ni un token léxico del lenguaje. Una primitiva como `add` puede ocupar una unidad BPE, mientras que un identificador compuesto puede dividirse en varias. La extensión del tokenizador agrega unidades indivisibles para símbolos relevantes, pero exige ampliar la matriz de embeddings y entrenar las nuevas filas.

La economía de tokens debe medirse en tres niveles. La compresión estructural compara el tamaño del árbol antes y después de introducir abstracciones. La longitud de una salida cuenta los tokens del programa producido. La eficiencia de extremo a extremo divide el volumen total generado por el número de soluciones útiles. Un sistema puede mejorar los dos primeros niveles y empeorar el tercero si aumenta el número de salidas incompletas. Esta separación es esencial para interpretar F0.5.

2.3 Lenguajes de dominio específico

Un lenguaje específico de dominio limita su poder expresivo para describir con mayor precisión las actividades de un área específica. Fowler (2010) distingue entre DSLs internos contruidos sobre el lenguaje anfitrión y DSLs externos con su propia sintaxis y analizador. Pucará es un DSL externo: el texto se procesa directamente y no se traduce primero a Python u otro lenguaje de propósito general. Hay tres ventajas de reducir el espacio sintáctico. La intención se expresa con menos elementos; el verificador puede reconocer propiedades estructurales; y el generador enfrenta menos alternativas equivalentes. El costo es menos familiaridad y la necesidad de construir herramientas personalizadas. En Pucará ese costo se acepta porque el autor previsto del código es un modelo, no una persona.

2.4 Síntesis de programas y aprendizaje de bibliotecas

La síntesis de programas busca construir expresiones que satisfagan una especificación. DreamCoder combina búsqueda simbólica y aprendizaje neuronal en un ciclo wake-sleep que amplía una biblioteca de conceptos reutilizables (Ellis et al., 2021). Stitch formula el aprendizaje de biblioteca como compresión de un corpus y utiliza síntesis top-down para encontrar abstracciones con gran cobertura (Bowers et al., 2023). LILO integra modelos de lenguaje, Stitch y documentación automática para producir bibliotecas interpretables (Grand et al., 2024).

Pucará adopta la idea de que los patrones frecuentes pueden convertirse en unidades del lenguaje, pero agrega dos restricciones. Cada abstracción debe expandirse al núcleo y conservar exactamente su comportamiento y su warm-up. Además, su nombre se incorpora al tokenizador como una unidad atómica. La evaluación comprueba por separado si la biblioteca comprime y si el modelo aprende a usarla de forma productiva.

2.5 Verificación determinista y causalidad

La ejecución determinista exige que el mismo programa, con los mismos datos y parámetros, produzca el mismo resultado. En el prototipo se fijan el orden de las operaciones, la biblioteca matemática, los generadores pseudoaleatorios, los splits y los digests. Para reducir errores de sustitución, las variables ligadas se representan con índices de De Bruijn, que reemplazan nombres por la distancia al ligador correspondiente (de Bruijn, 1972).

La causalidad temporal se refiere a que un valor en la barra t no dependa de datos posteriores. Pucará no expone retardos negativos y calcula un onset o warm-up para cada término. Ese onset es la primera barra en la que la salida puede considerarse válida. La verificación se refuerza con pruebas metamórficas: modificar el futuro de una serie no puede cambiar resultados anteriores.

2.6 Trading sistemático y evaluación por simulación

Una estrategia sistemática define reglas de señal, normalización, dimensionamiento y construcción de posiciones. El marco de Carver (2015) ofrece componentes reproducibles para señales de tendencia, volatilidad y diversificación. Pucará utiliza ese dominio porque admite especificaciones estructuradas y ejecución barra por barra. Las pruebas retrospectivas tienen riesgos conocidos: la elección entre muchas variantes puede inflar el índice de Sharpe y llevar a un sobreajuste (Bailey et al., 2014). El índice de Sharpe desinflado resuelve parte de ese sesgo cuando se conocen el número de pruebas y la distribución de los rendimientos (Bailey & López de Prado, 2014). En el experimento se emplea un screen de Sharpe en el tramo in-sample y un e-process en el tramo de holdout.

El e-process permite acumular evidencia bajo detención opcional; e_max representa el máximo de ese proceso de evidencia, no exposición financiera.

2.7 Definiciones operativas

Tabla 2. Definición de las métricas principales.

Término	Definición operativa
Programa válido	Salida que contiene una expresión Pucará completa, parsea, realiza round-trip canónico y supera el análisis de warm-up.
Superviviente	Programa no memorizado que devuelve una posición, supera Sharpe $IS > 0,5$ y alcanza $e_max \geq 20$ en holdout.
pass@5	Probabilidad estimada de obtener al menos un éxito en cinco muestras de una tarea; se promedia sobre tareas.
Fidelidad	Satisfacción de los seis atributos estructurales registrados para la especificación: señal, topología, filtro, topes, amortiguación y salida.
Token BPE	Unidad de segmentación del tokenizador del modelo, distinta del token léxico del parser.
Compresión estructural	Relación entre nodos del corpus antes y después de reescribirlo con abstracciones.
IS	Tramo de entrenamiento o screen temporal $[0, 450)$ en el mundo instrumentado.

Término	Definición operativa
Holdout	Tramo [450, 700) utilizado por el e-process durante la evaluación.
True-OOS	Tramo [700, 900), ausente de la ruta de selección y utilizado solo de manera exploratoria en la vitrina.

Fuente: elaboración propia.

3. Estado del arte

3.1 Generación de código en lenguajes de propósito general

La mayor parte de la investigación sobre modelos de código evalúa lenguajes generales. Codex se midió sobre funciones Python a partir de docstrings y mostró que el muestreo repetido eleva la cobertura de tareas (Chen et al., 2021). MBPP extendió la evaluación a problemas breves y mostró mejoras por escala y fine-tuning (Austin et al., 2021). Estos trabajos establecen una metodología, pero no cuestionan si el lenguaje de destino está optimizado para el modelo.

Los lenguajes generales aportan bibliotecas, depuradores y experiencia acumulada. También permiten construcciones que no son relevantes para una tarea pequeña. Cuando la especificación describe una estrategia y el programa solo será consumido por un intérprete especializado, gran parte de esa flexibilidad no tiene un valor directo. Pucará explora el extremo opuesto: un lenguaje cerrado sobre operaciones verificables.

3.2 DSL y representaciones orientadas al dominio

Los DSLs reducen la distancia entre concepto y programa. En finanzas, hay lenguajes para contratos, consultas y reglas, pero generalmente priorizan la legibilidad humana o la integración con una plataforma. La propuesta de Pucará no es solo un DSL de trading. Su diseño considera explícitamente la distribución del generador, la longitud del BPE y la posibilidad de promover abstracciones del modelo al vocabulario.

La notación prefijada completamente parentetizada tiene una historia en lenguajes funcionales y representaciones intermedias. Su principal ventaja experimental es eliminar la precedencia y hacer explícito el árbol. La desventaja es el costo de los paréntesis. Así que la economía no es una función de la apariencia, sino que se mide con el tokenizador real.

La literatura reciente comenzó a considerar al modelo de lenguaje como destinatario del diseño de DSLs. Mernik et al. (2005) sistematizaron cuándo y cómo desarrollar un

lenguaje de dominio específico con criterios pensados para programadores humanos; Pucará retoma esa metodología como antecedente, pero desplaza al consumidor principal: las decisiones de sintaxis, vocabulario y canonicidad se evalúan contra la distribución y el tokenizador del generador. En una dirección complementaria, B. Wang et al. (2023) mostraron que un modelo puede producir programas en DSLs poco frecuentes si la gramática BNF se presenta en el contexto (grammar prompting), tratando la gramática como insumo de inferencia. Ese trabajo es un antecedente directo del problema —lograr que un modelo genere un lenguaje que no domina—, aunque la estrategia difiere: en lugar de condicionar la generación con la gramática, Pucará rediseña el lenguaje para que su forma canónica sea aprendible mediante fine-tuning y verificable por un árbitro determinista.

3.3 Aprendizaje de bibliotecas

DreamCoder mostró que un sistema de síntesis puede aprender conceptos y acumularlos en una biblioteca creciente. Stitch mejora la escalabilidad de la compresión sobre grandes corpus y asigna a cada abstracción un beneficio de compresión explícito. LILO incorpora nombres y documentación generados por modelos de lenguaje. Estas líneas inspiran la capa de abstracciones de Pucará, pero el experimento añade una exigencia adicional: la biblioteca debe ser utilizable por un modelo que aprende a trabajar bajo un presupuesto igualado.

Hay diferentes propiedades de compresión y reutilización. Una abstracción puede reducir enormemente el corpus cuando un reescritor la introduce, pero el generador puede no aprender a llamarla. F0.5 hace visible la separación: la biblioteca y el tokenizador producen programas válidos más cortos, mientras que el brazo C genera muchos incompletos. El resultado no contradice la compresión de Stitch; muestra que la compresión estructural no garantiza la internalización neuronal del vocabulario.

El riesgo de que un generador no utilice una biblioteca disponible está documentado en dominios cercanos. LEGO-Prover acumula lemas verificados como habilidades para la demostración de teoremas (H. Wang et al., 2024) y TroVE induce cajas de herramientas verificables, mostrando que el uso productivo de funciones de alto nivel exige mecanismos explícitos de selección y poda (Z. Wang et al., 2024). El resultado del brazo coevolutivo de esta tesis —ninguna de las 949 salidas válidas invoca una abstracción de la biblioteca— replica ese patrón bajo un protocolo con presupuesto igualado y verificación determinista, y convierte un riesgo cualitativo reportado por la literatura en una medición con intervalos de confianza (sección 5.10).

3.4 Vocabularios extendidos y lenguajes IA-nativo

Los tokenizadores de modelos de código contienen muchos lenguajes existentes. Un nuevo lenguaje hereda segmentaciones accidentales. Agregar tokens permite que una operación o abstracción se represente como una unidad, pero también requiere entrenar nuevos embeddings y cambia la distribución de salida. Pucará mide ambos efectos con un tokenizador base y uno extendido con 14 unidades nuevas.

La noción IA-nativo empleada en esta tesis no implica opacidad como un fin. De hecho, la sintaxis primaria está adaptada para la generación automática, mientras que la auditabilidad se obtiene a través de la semántica, el printer canónico, el árbol, los informes y una futura proyección legible. Esta representación puede ser difícil de expresar a mano sin perder verificabilidad.

La adaptación del modelo sigue técnicas estándar de ajuste eficiente en parámetros, que este trabajo toma como adaptaciones y no como aportes: LoRA introduce matrices de bajo rango en las capas lineales (Hu et al., 2021) y QLoRA permite entrenarlas sobre un modelo base cuantizado a 4 bits (Dettmers et al., 2023). La extensión del tokenizador con inicialización de cada embedding nuevo en la media de los embeddings de los subtokens que

componían su nombre es una práctica establecida que aquí se fija por diseño y se verifica con pruebas unitarias (sección 5.4). La contribución original no reside en esas técnicas, sino en la medición: economía de tokens auditada de extremo a extremo con los tokenizadores serializados, bajo presupuesto de entrenamiento igualado entre brazos y con un árbitro económico determinista.

3.5 Verificación de código generado

Los modelos pueden producir código que compila, pero no cumple con el propósito previsto. Las herramientas híbridas permiten probar, analizar y sintetizar para corregir o filtrar la salida. Jigsaw, por ejemplo, utiliza un modelo y etapas que entienden la estructura del programa (Jain et al., 2022). Pucará adopta la misma idea general pero con un DSL que puede ser verificado directamente.

Además de eso, la evaluación económica es otro desafío. Una señal diferente de la que se desea podría beneficiarse del mismo mundo sintético. El backtest responde si el programa funciona bien en ese entorno, pero no responde si implementa la especificación. El criterio de fidelidad estructural se toma para cerrar esa brecha, aunque su validación externa con revisores humanos aún está por hacerse.

Una línea muy activa impone la validez sintáctica durante la propia decodificación. Synchronesh introdujo la decodificación semántica restringida, que filtra en cada paso los tokens que violarían la gramática o restricciones semánticas del lenguaje destino (Poesia et al., 2022); Geng et al. (2023) generalizaron la decodificación restringida por gramáticas a tareas estructuradas sin fine-tuning; Willard y Louf (2023) la reformularon como transiciones de una máquina de estados finitos con costo por token casi constante, y XGrammar la extendió a gramáticas libres de contexto con ejecución solapada con la inferencia (Dong et al., 2024). Estos antecedentes garantizan programas bien formados por construcción y habrían suprimido el modo de falla dominante del brazo coevolutivo (terminaciones

incompletas y colas degeneradas; sección 5.10). La versión evaluada de Pucará no restringe la decodificación: muestrea libremente y verifica después, lo que permite medir cuánto del lenguaje internalizó el modelo sin enmascaramiento. La adopción de decodificación restringida se identifica como adaptación futura (sección 8.4), con una salvedad documentada por Park et al. (2024): el enmascaramiento voraz distorsiona la distribución del modelo, de modo que imponer gramaticalidad no garantiza calidad ni fidelidad. La distinción entre validez y fidelidad que esta tesis cuantifica seguiría siendo necesaria incluso con gramática impuesta.

En síntesis de programas con LLM, la evaluación dominante usa pruebas unitarias como proxy de intención (Chen et al., 2021; Austin et al., 2021), y Jigsaw agrega posprocesamiento consciente de la estructura (Jain et al., 2022). Pucará adapta esa idea a un dominio donde el juez natural —el backtest— mide desempeño y no intención: los detectores estructurales por atributo cumplen el papel de las pruebas unitarias, y la brecha medida entre supervivencia y fidelidad ($\text{pass}@5$ held-out de 0,7726 frente a 0,0331; sección 5.8) es la versión cuantificada del fenómeno que motiva esas herramientas. Como antecedente estadístico del gate secuencial, el e-process empleado pertenece a la familia de métodos de inferencia válida en cualquier momento desarrollada por Howard et al. (2021) y a los esquemas de apuesta sobre variables acotadas de Waudby-Smith y Ramdas (2024); la sección 4.8 formaliza la construcción adaptada.

3.6 Brecha abordada por Pucará

En el fondo hay generación de código, DSL, aprendizaje de bibliotecas y verificación. La brecha es que están juntos bajo una simple pregunta económica: si una representación para un modelo puede reducir tokens sin perder un árbitro determinista. Sus contribuciones se evalúan en cuatro niveles: artefacto ejecutable, capacidad de síntesis, compactación y

fidelidad. La adaptación conjunta del modelo y el vocabulario se trata como hipótesis experimental, no como propiedad asumida.

La Tabla 3 resume la relación entre las líneas revisadas y este trabajo, distinguiendo qué se toma como antecedente, qué se adapta y qué constituye contribución original.

Tabla 3. Estado del arte: antecedentes, adaptaciones y contribución original.

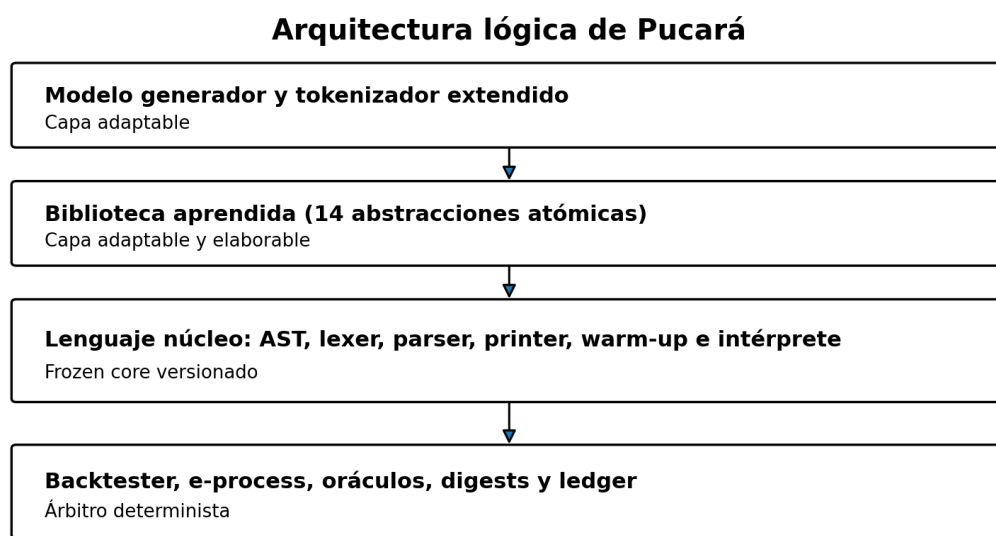
Línea de trabajo	Trabajos representativos	Relación con Pucará
Síntesis de programas con LLM y pass@k	Chen et al. (2021); Austin et al. (2021)	Antecedente metodológico: estimador pass@k y muestreo repetido. Pucará lo aplica a un DSL ausente del preentrenamiento y agrega gates deterministas y fidelidad estructural.
DSL y diseño de lenguajes	Fowler (2010); Mernik et al. (2005); B. Wang et al. (2023)	Antecedente conceptual. Contribución original: un DSL cuyo destinatario primario de diseño es el generador (canonicidad, índices de De Bruijn, economía BPE medida con el tokenizador real).
Generación restringida por gramáticas	Poesia et al. (2022); Geng et al. (2023); Willard y Louf (2023); Dong et al. (2024); Park et al. (2024)	Antecedente no incorporado en la versión evaluada; identificado como adaptación futura (8.4). La brecha validez-fidelidad medida indica que imponer gramaticalidad no bastaría.
Aprendizaje de bibliotecas y representaciones compactas	Ellis et al. (2021); Bowers et al. (2023); Grand et al. (2024); H. Wang et al. (2024); Z. Wang et al. (2024)	Antecedente y adaptación (Stitch se usa como compresor). Contribución: separación medida entre compresión estructural, compactación BPE (-34,2 % en válidos) y eficiencia extremo a extremo (8,43x tokens por superviviente).
Ajuste eficiente y extensión de tokenizadores	Hu et al. (2021); Dettmers et al. (2023)	Adaptación directa (QLoRA; mean-init de embeddings nuevos). Contribución: protocolo con presupuesto de tokens igualado y ledger exacto reconciliado.
Verificación semántica y fidelidad	Jain et al. (2022)	Antecedente. Contribución: árbitro determinista con e-process y detectores estructurales por atributo; cuantificación de la brecha supervivencia-fidelidad (0,7726 a 0,0331).
Inferencia válida en cualquier momento	Ville (1939); Howard et al. (2021); Waudby-Smith y Ramdas (2024)	Adaptación del e-process aGRAPA como gate secuencial, con validación Monte Carlo del error Tipo I (sección 4.8).

Fuente: elaboración propia.

4. Diseño e implementación de Pucará

4.1 Descripción general

Pucará se implementó como una biblioteca Rust. El artefacto incluye módulos de AST, tokenizer, parser, printer, series, kernels, intérprete, warm-up, backtesting, corpus, integración con Stitch, evaluación experimental y utilidades de reproducción. Los componentes Python administran entrenamiento, generación, scoring, agregación y oráculos independientes.



Las capas superiores pueden evolucionar; el núcleo y el árbitro se protegen mediante versiones y pruebas de integridad.

Figura 2. Arquitectura lógica de Pucará.

Fuente: elaboración propia sobre el repositorio en el commit 15ff05c.

Tabla 4. Componentes principales del artefacto.

Componente	Función	Tecnología
Front-end	Tokenización léxica, parsing y forma canónica.	Rust, Logos y parser recursivo.
Núcleo semántico	AST, primitivas y evaluación de términos.	Rust.
Análisis temporal	Cálculo de warm-up y pruebas de causalidad.	Rust.
Backtester	P&L, costos, retornos, Sharpe y e-process.	Rust.
Biblioteca	Compresión, lift y elaboración de abstracciones.	Stitch + Rust.

Componente	Función	Tecnología
Modelo	Fine-tuning, tokenizador extendido y generación.	Qwen2.5-Coder-7B, Python/Transformers.
Evaluación	Scoring, pass@5, gates y diagnósticos.	Rust + Python.
Trazabilidad	Manifiestos, digests, tags y artefactos crudos.	Git, JSON y SHA/FNV.

Fuente: elaboración propia.

4.2 Principios de diseño IA-nativo

El diseño se rige por cinco principios. La representación debe ser única; cada elemento debe expresar una decisión del dominio; el núcleo debe permanecer pequeño y versionado; las abstracciones deben desarrollarse hasta el núcleo; y el modelo no debe ser juez de su propia salida. La economía de tokens se evalúa en el tokenizador real, no en el número de caracteres.

La opacidad no es una contribución en sí misma. El lenguaje retiene nombres reconocibles en el núcleo porque el modelo base ya tiene representaciones útiles para términos frecuentes como precio, media o desviación estándar. Las abstracciones aprendidas reciben identificadores estables y atómicos. La auditabilidad se sostiene en la expansión al núcleo, el printer y los reportes.

4.3 Representación canónica

Un programa es un término lambda en forma S-expression. La aplicación se expresa como una lista cuyo primer elemento es una primitiva y los restantes son argumentos. Las variables ligadas utilizan índices de De Bruijn, escritos como \$0, \$1 y así sucesivamente. Los números se conservan inicialmente como texto decimal para que el printer pueda fijar una forma canónica sin depender de la representación binaria.

El siguiente programa corresponde a una señal de precio estandarizada por su media y desvío de 25 barras, recortada y convertida en una posición escalada por capital, precio y valor del punto. Es una muestra real del dominio experimental.

```
(lam (set_position
  (div
    (mul
      (clip
        (mul
          (div
            (sub (price $0) (mean (price $0) 25))
            (std (price $0) 25))
          2.0)
        (neg 20.0)
        20.0)
      equity)
    (mul (price $0) (point_value $0))))))
```

La notación mantiene explícita la composición. El parser produce un árbol; el printer reconstruye una única cadena; y el round-trip exige que volver a parsear esa cadena reproduzca el mismo término. Este contrato elimina variantes de formato del corpus y simplifica deduplicación, hashing y caché.

4.4 Núcleo y vocabulario

La versión actual contiene 38 primitivas. El número difiere de etapas anteriores del proyecto porque el núcleo fue ampliado de forma aditiva y re-congelado bajo versiones explícitas. La tesis presenta la versión asociada al commit final y conserva las cifras históricas únicamente cuando describe una corrida ejecutada sobre un estado anterior.

Tabla 5. Categorías funcionales del núcleo actual.

Categoría	Primitivas	Cantidad
Acceso y proyectores	price, raw_price, volume, point_value, equity, sector, margin_rate, weight	8
Ventanas y estado	shift, ema, std, mean, sum, max, min, cov, scan	9
Aritmética y recorte	add, sub, mul, div, neg, abs, ln, sqrt, sign, clip	10
Lógica y control	gt, lt, eq, and, or, not, if_then_else	7
Agregación transversal	sum_over, mean_over	2
Construcción de posición	set_position, to_contracts	2

Fuente: elaboración propia sobre src/token.rs y src/ast.rs en el commit final.

En el tokenizer léxico de Rust cada palabra reservada constituye un token del lenguaje. En el tokenizador BPE del modelo la situación es distinta: 29 de las 38 primitivas

son atómicas y nueve se segmentan en dos o tres unidades. Esta diferencia se informa porque la tesis estudia costo BPE, no solo complejidad del parser.

4.5 Lexer, parser y round-trip

El lexer se implementó con Logos. Rechaza palabras no registradas y reconoce paréntesis, lam, primitivas, variables y números. El parser recursivo interpreta la forma prefija y valida aridades y posiciones especiales. El printer normaliza espacios y números. Las pruebas cubren programas completos, variables de De Bruijn, literales y rechazo de símbolos inexistentes.

El front-end funciona como primera puerta de control. Las salidas del modelo se extraen antes de parsearse; si no existe una expresión completa se clasifican como `no_expr`. Si existe una expresión pero el parser falla, se registra `parse_fail`. Esta distinción permitió identificar el modo de falla dominante del brazo C.

4.6 Intérprete y semántica numérica

El intérprete recorre el AST y delega las operaciones en una semántica concreta. Los valores pueden representar instrumentos, series o posiciones. Las formas `sum_over`, `mean_over` y `scan` requieren tratamiento especial porque introducen iteración o estado. La evaluación de una estrategia se realiza por instrumento y luego se agrega el P&L de cartera.

La implementación fija el orden de reducción y utiliza rutinas numéricas controladas. La portabilidad exigió reemplazar funciones matemáticas dependientes de plataforma por `libm` y re-congelar los digestos. Las operaciones que admiten igualdad exacta se comparan bit a bit; las comparaciones diferenciales de coma flotante utilizan la tolerancia documentada por la suite.

4.7 Warm-up y causalidad

Cada primitiva propaga un onset. Un shift agrega su retardo; una ventana de longitud w requiere $w-1$ observaciones adicionales; las combinaciones toman el máximo de sus entradas. Un programa cuyo onset no despeja el horizonte de evaluación se rechaza antes del backtest.

Las abstracciones aprendidas se representan con tarjetas simbólicas que expresan su inicio como una función max-plus de los intervalos. La ausencia de desplazamientos negativos garantiza que la causalidad se preserve, y esa propiedad se verifica además mediante pruebas por perturbación. El mundo sintético de evaluación refuerza la separación temporal: la ruta utilizada para la selección tiene una serie de longitud 700 y no puede indexar las 200 barras del tramo true-OOS. Un mundo canario solo introduce una tendencia en la cola y verifica que las métricas de selección no sean ciegas.

4.8 Backtester y criterios de evaluación

El pipeline evalúa a cada candidato en etapas. Primero parsea y verifica el round-trip canónico; luego calcula el warm-up; marca copias exactas del corpus de entrenamiento; exige que la expresión devuelva una posición en todos los instrumentos; calcula el Sharpe anualizado en $[\text{onset}, 450)$; y finalmente aplica el e-process en $[450, 700)$. Un superviviente no memorizado debe superar Sharpe 0,5 y e_max 20.

Definición formal del e-process. Sea r_t el retorno neto de costos de la estrategia en la barra t del tramo holdout y F_{t-1} la información disponible hasta la barra anterior. La hipótesis nula es que la estrategia no posee ventaja: $H_0: E[x_t | F_{t-1}] \leq 0$, donde x_t es el retorno estudentizado y acotado que se define a continuación. La varianza se estima de forma predecible con una media móvil exponencial de los retornos al cuadrado con span 32, calculada exclusivamente sobre $r_0, \dots, r_{t-1}$: $\sigma^2_t = \text{EWMA}(r^2; \text{span } 32)$, inicializada con r_0^2 . El resultado de la apuesta en t es $x_t = \text{clip}(r_t / \sigma_t; -5; 5) / 5$, acotado en $[-1; 1]$. La

fracción de apuesta λ_t sigue la regla aGRAPA de Waudby-Smith y Ramdas (2024): $\lambda_t = \text{clip}(\hat{\mu}_{t-1} / (\hat{v}_{t-1} + \hat{\mu}_{t-1}^2); 0; 0,5)$, donde $\hat{\mu}$ y $\hat{v}$ son la media y la media cuadrática acumuladas de los x observados desde la barra 32; durante las primeras 32 barras (warm-up del estimador de varianza) se fija $\lambda_t = 0$. La ecuación de actualización de la riqueza del proceso es $W_t = W_{t-1} \cdot (1 + \lambda_t \cdot x_t)$, con $W_0 = 1$, y la evidencia acumulada es $e_{\max} = \max_{\{s \leq t\}} W_s$.

Todos los ingredientes ($\hat{\sigma}_t$, $\hat{\mu}$, $\hat{v}$ y por lo tanto λ_t) son predecibles: dependen solo de F_{t-1} . Esa propiedad es el tratamiento de la dependencia temporal: bajo H_0 , W_t es un supermartingala no negativo cualquiera sea la estructura de dependencia del proceso de retornos, sin supuestos de independencia ni de distribución. Por la desigualdad de Ville (1939), $P(\sup_t W_t \geq 1/\alpha) \leq \alpha$ bajo H_0 ; con $\alpha = 0,05$ el umbral es $1/\alpha = 20$. El significado exacto de $e_{\max}$ es el de un factor de apuesta: $e_{\max} = 20$ indica que un apostador que partiera de una unidad de capital y apostara contra H_0 con la regla anterior habría multiplicado su capital por 20 en algún momento del tramo, evidencia equivalente a rechazar H_0 al nivel 0,05 con validez en cualquier momento y bajo paro opcional (Howard et al., 2021). El umbral 20 no se calibró sobre los datos: es la constante $1/\alpha$ implicada por el nivel prerregistrado, y su error Tipo I se validó por Monte Carlo determinista sobre tres familias nulas (ruido normal iid; GARCH(1,1) con $\sigma_t^2 = 10^{-6} + 0,90 \cdot \sigma_{t-1}^2 + 0,09 \cdot r_{t-1}^2$; y t de Student con 4 grados de libertad reescalada), con frecuencia de cruce menor o igual a 0,0546.

Como ejemplo breve de evolución, procesado con la implementación exacta (src/backtest/eprocess.rs) sobre una serie ilustrativa con retorno constante $r_t = 0,01$ por barra: durante las barras 1 a 32, $\lambda_t = 0$ y $W_t = 1$; desde la barra 33, $x_t = 0,2$ (el retorno equivale a un desvío estándar estimado y $\text{clip}(1; -5; 5)/5 = 0,2$) y λ_t alcanza el tope 0,5, de modo que cada barra multiplica la riqueza por $1 + 0,5 \cdot 0,2 = 1,1$: $W_{33} = 1,10$; $W_{40} = 2,14$; $W_{50} = 5,56$; $W_{60} = 14,42$, y el proceso cruza $e_{\max} \geq 20$ en la barra 64. En contraste,

una serie alternante $+0,01/-0,01$ (media condicional nula) termina 250 barras con $e_{\max} = 1,0$: nunca acumula evidencia. En los supervivientes reales del mundo sintético los $e_{\max}$ observados de la tarea auditada t_{20} se ubican entre $1,05 \cdot 10^6$ y $1,13 \cdot 10^8$, un exceso de seis a siete órdenes de magnitud sobre el umbral que se analiza críticamente en la sección 5.6.

La caché se indexa por el hash de la forma canónica. Reenviar el mismo programa no genera un nuevo backtest ni incrementa el ledger. El ledger cuenta candidatos únicos y sustenta las correcciones por pruebas múltiples. Los costos se fijan en un punto base por operación en el mundo sintético instrumentado.

4.9 Frozen core e integridad

El núcleo no se considera inmutable para toda la historia del proyecto, sino dentro de cada versión experimental. Toda modificación requiere un re-freeze explícito, nuevos digestos y pruebas. La suite incluye un control que falla ante la alteración de un byte de los archivos protegidos. De este modo, una biblioteca o un modelo no pueden cambiar silenciosamente la semántica del árbitro.

4.10 Corpus y biblioteca aprendida

La Etapa 1-A construyó un corpus curado de 112 programas que cubría señales EWMAC, momento, breakouts, volatilidad, combinaciones, posiciones, caps, buffers y agregación transversal. Stitch produjo 10 abstracciones jerárquicas. El corpus pasó de 102.782 a 3.502 nodos, una relación de 29,35 veces. El mayor programa pasó de 14.579 a 83 nodos. Estas cifras son compresión estructural y no deben confundirse con BPE.

Cada reescritura se elaboró nuevamente al núcleo y se comparó bit a bit contra el programa original. También se exigió preservación exacta del warm-up. Durante el ciclo F0.4 la biblioteca creció hasta 14 abstracciones y el tokenizador extendido incorporó sus nombres como unidades atómicas. La biblioteca comprime de forma demostrable; F0.5 evalúa si el generador puede aprovecharla.

4.11 Tokenizador extendido y modelo

El modelo base fue Qwen2.5-Coder-7B, un modelo especializado en código (Hui et al., 2024). El brazo B utilizó el tokenizer base y el adaptador ganador de F0.3. El brazo C utilizó el adaptador de la tercera iteración, el tokenizer base ampliado en 14 unidades y las filas de embedding correspondientes. La longitud pasó de 151.665 a 151.679 tokens.

El manifiesto fija la revisión exacta del modelo base Qwen2.5-Coder-7B utilizada en el entrenamiento y la generación mediante su hash de commit 0396a761...8b02.

La atomicidad fue comprobada antes de la corrida. No obstante, agregar unidades no garantiza su uso correcto. Los embeddings deben aprender cuándo emitir cada nombre, con qué aridad y dentro de qué estructura. La degradación observada en C se analiza en el capítulo 5.

4.12 Reproducibilidad

La reproducción se apoya en commits, Cargo.lock, toolchain, versiones Python, manifiestos, seeds, digests y JSONL crudos. La corrida F0.5 registró la revisión del modelo, la GPU A100 de 40 GB, el backend SDPA, los adaptadores, el tokenizador y los hashes de salida. Los resultados se recalcularon desde los registros crudos y scoreados.

El cierre final dejó master en 15ff05c07d281d703c7b472b4c3eee7600b60529, con los hitos históricos preservados como tags anotados. La vitrina t20 se integró en el commit ee84d663aef4b03fa40d481bbce89cafeb20d84d. Esta información se presenta en detalle en el capítulo 7 y el apéndice.

4.13 Componentes diseñados no implementados

La definición de largo plazo contempla generación restringida por gramática y tipos, tipos refinados para riesgo, tiempo como tipo, capital lineal, efectos, compilación con Cranelift, aislamiento WebAssembly y un camino FPGA. Ninguno de esos componentes forma parte del resultado evaluado. Se incluyen como trabajo futuro porque responden a

limitaciones observadas, en particular la finalización sintáctica y la seguridad estática, pero no se utilizan para explicar métricas presentes.

5. Metodología experimental y evaluación

5.1 Formulación del problema

El experimento principal evalúa si adaptar conjuntamente el modelo, la biblioteca y el tokenizador mejora la síntesis respecto de un lenguaje estático. La comparación no pregunta si Stitch comprime el corpus; esa propiedad ya se había verificado. Pregunta si el ciclo completo produce más tareas resueltas bajo un presupuesto de cómputo igualado.

El criterio decisivo se fijó antes de observar F0.5: el brazo C debía superar al B en al menos cinco puntos porcentuales de pass@5 de supervivencia sobre tareas no vistas. Si no lo hacía, la coevolución se archivaría como línea principal. La regla evita reinterpretar un resultado desfavorable después de la corrida.

5.2 Diseño experimental

El diseño inicial contempló un baseline en Python, pero la corrida confirmatoria F0.5 comparó dos brazos Pucará. Esta tesis no presenta resultados inexistentes para una condición A. El brazo B aísla el desempeño del lenguaje estático. El brazo C agrega tres iteraciones de generación, verificación, compresión, extensión del tokenizador y reentrenamiento. Ambos parten del mismo modelo base y se evalúan sobre las mismas tareas y seeds.

Tabla 6. Comparación de los brazos experimentales.

Dimensión	Brazo B: lenguaje estático	Brazo C: adaptación conjunta
Modelo base	Qwen2.5-Coder-7B, revisión fijada.	El mismo modelo base.
Adaptador	Ganador de F0.3.	Adaptador de la iteración 3 de F0.4.
Tokenizador	Vocabulario base de 151.665 unidades.	Vocabulario base + 14 abstracciones atómicas.
Biblioteca	No se exige uso de abstracciones aprendidas.	Biblioteca iter3 de 14 abstracciones.
Tareas	40 tareas; 153 muestras por tarea.	Las mismas 40 tareas y seeds.
Generación	Batch 20; máximo 2.048 tokens.	Mismos parámetros de generación.
Propósito	Medir la síntesis sobre Pucará estático.	Medir el aporte adicional de la coevolución.

Fuente: elaboración propia

5.3 Tareas y particiones

El conjunto contiene 40 especificaciones. Las tareas t01-t10 se clasificaron como in-distribution y t11-t40 como held-out. Cada tarea registra seis atributos: familia de señal, topología de combinación, filtro, cadena de topes, amortiguación y forma de salida. Esos atributos se utilizaron posteriormente para medir fidelidad sin modificar las especificaciones.

El archivo `eval_tasks.json` que registra estos seis atributos por tarea tiene 12.600 bytes y está fechado el 10 de julio, el mismo tamaño exacto que el archivo entregado para esta evaluación; el bundle de entrenamiento correspondiente está fechado el 20 de julio. Los atributos de fidelidad quedaron, por lo tanto, registrados antes del entrenamiento y no fueron definidos con posterioridad a los resultados.

La evaluación temporal utiliza cuatro instrumentos sintéticos y 900 barras. El tramo $[0, 450)$ se emplea para el screen de Sharpe; $[450, 700)$ para el e-process; y $[700, 900)$ queda fuera de la ruta de selección. La separación de tareas y la separación temporal responden a riesgos distintos: generalización estructural y fuga temporal.

El control de fuga entre particiones de entrenamiento se registró explícitamente en la corrida: straddling OK, 0 grupos de `seed_name` presentes simultáneamente en ambas particiones, lo que descarta fuga de programas relacionados entre el conjunto de entrenamiento y el de validación del corpus de ajuste.

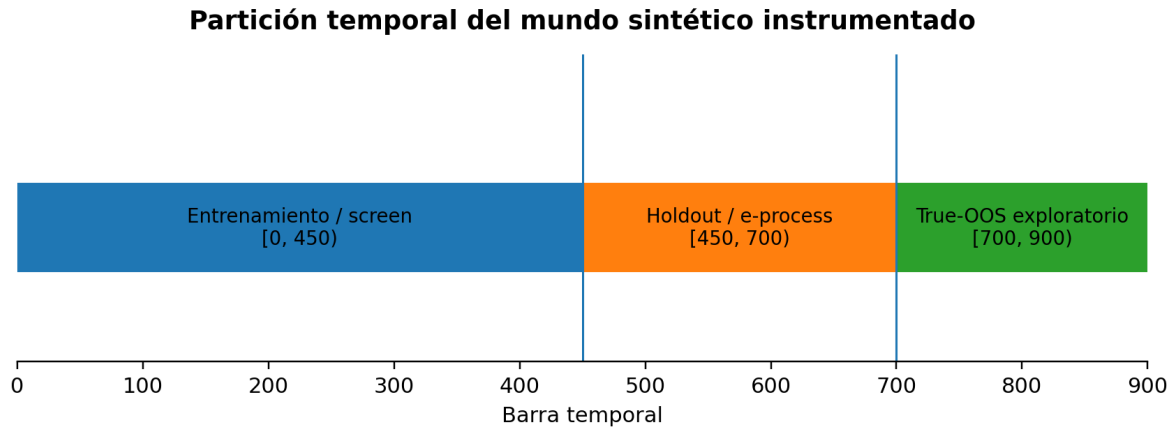


Figura 3. Partición temporal del mundo sintético instrumentado.

Fuente: elaboración propia sobre src/experiment/world.rs.

5.4 Brazos comparados

Cada brazo produjo 6.120 registros: 153 muestras por tarea. La GPU utilizada fue una NVIDIA A100-SXM4 de 40 GB. El backend de atención se fijó en SDPA y se deshabilitaron rutas incompatibles de xFormers. Los JSONL crudos conservan la salida textual y los scoreados registran el estado de cada puerta. Los digests permiten verificar que el análisis corresponde a los archivos devueltos por Colab.

Configuración de entrenamiento del brazo B. El corpus congelado de la fase F0.3 contiene 402 pares especificación-programa admitidos bajo el presupuesto $L_{\text{gen}} = 4.096$, divididos en 389 pares de entrenamiento y 13 de validación sin grupos de semillas compartidos entre particiones; los digests FNV-1a/64 de ambos archivos quedaron fijados en el manifiesto del corpus. El ajuste usó QLoRA sobre Qwen2.5-Coder-7B en la revisión fijada por el manifiesto: el modelo base se carga cuantizado a 4 bits y los adaptadores LoRA se entrenan en bfloat16 sobre las siete capas lineales de cada bloque (q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj y down_proj), con alfa igual al rango, dropout 0 y sin sesgos. Se entrenaron dos corridas idénticas salvo el rango (64 y 256) como barrido de selección. La receta prerregistrada fija: learning rate $2 \cdot 10^{-4}$, scheduler coseno con warmup del 3 %, 3 épocas con checkpoint por época, batch efectivo 16 (2 por dispositivo por 8 pasos de

acumulación), longitud máxima de secuencia 2.048, pérdida calculada únicamente sobre los tokens del programa (la especificación se enmascara), sin packing, con agrupamiento por longitud y semilla fija 0xF0035F70. El optimizador es AdamW, el valor por defecto del Trainer de Transformers, que la receta no modifica.

El checkpoint se seleccionó exclusivamente por pérdida de validación sobre los 13 pares reservados, sin observar jamás las tareas de evaluación, mediante una regla de un error estándar con bootstrap de 1.000 remuestreos (semilla 0xF003B007) y preferencia por el rango menor en caso de empate estadístico. Ganó el rango 256 en la época 3, con pérdida de validación 0,013641 y error estándar 0,002547. Cada corrida procesó 655.281 tokens de entrenamiento, cifra reconciliada de manera exacta entre la proyección determinista en CPU y el ledger real de la GPU; solo los tokens de la corrida ganadora computan para el presupuesto igualado entre brazos. El hardware fue una NVIDIA A100-SXM4 de 40 GB con torch 2.9.1+cu128; el tiempo de pared por corrida quedó registrado en el campo `wall_clock_s` de cada ledger devuelto por Colab, y la ventana completa de entrenamiento y generación se presupuestó en 2,5 a 4 horas de A100 según el runbook.

Configuración del brazo C. El brazo C utilizó el adaptador de la tercera iteración del loop de coevolución F0.4, entrenado con la misma receta base sobre el corpus densificado por la biblioteca (el programa medio pasó de 347 a 153 tokens) y con un presupuesto acumulado de 241.666 tokens de entrenamiento, aproximadamente el 37 % del consumido por B bajo la regla de presupuesto del loop. El tokenizador base se extendió con los 14 nombres de abstracciones como tokens atómicos (vocabulario de 151.665 a 151.679) y cada fila nueva de embedding —de entrada y de salida— se inicializó con la media de las filas de los subtokens en que el nombre se segmentaba sobre el vocabulario original (mean-init), regla verificada con un caso derivado a mano en la suite de pruebas. La equivalencia de conteo entre el tokenizador Rust de admisión y el de Python se comprobó sobre los 552 programas del

corpus con igualdad exacta por programa. Las tres corridas de generación comparten configuración: temperatura 0,7, top-p 0,95, tope de 2.048 tokens nuevos, lotes de 20 muestras (un lote es exactamente una tarea) y semillas derivadas por brazo y lote, lo que hace la generación reproducible e incluso reanudable. La Tabla 7 consolida los hiperparámetros.

Tabla 7. Hiperparámetros de entrenamiento y generación.

Elemento	Configuración
Modelo base	Qwen2.5-Coder-7B, revisión fijada en el manifiesto
Método	QLoRA: base cuantizada a 4 bits, adaptadores en bfloat16 (Unsloth)
Corpus (B)	402 pares especificación-programa: 389 entrenamiento / 13 validación, sin grupos de semillas compartidos; digests FNV-1a/64 en corpus_manifest.json
Épocas	3, con checkpoint al final de cada una
Learning rate / scheduler	2e-4; coseno con warmup 3 %
Optimizador	AdamW (valor por defecto del Trainer de Transformers, sin modificación)
Batch	2 por dispositivo x 8 pasos de acumulación (efectivo 16); agrupamiento por longitud; sin packing
LoRA	rango en {64, 256} (barrido), alfa = rango, dropout 0, sin bias, sobre q, k, v, o, gate, up y down proj
Longitud máxima de secuencia	2.048; pérdida solo sobre los tokens del programa (especificación enmascarada)
Selección de checkpoint	pérdida de validación, regla 1-SE con bootstrap de 1.000 remuestreos, parsimonia hacia rango 64; ganó rango 256, época 3 (val loss 0,013641, EE 0,002547)
Semillas	entrenamiento 0xF0035F70; bootstrap 0xF003B007; generación derivada por brazo y lote
Tokens de entrenamiento	B: 655.281 (ledger reconciliado exacto con la proyección); C: 241.666 acumulados (~37 % de B)
Tokens nuevos (C)	14 nombres de abstracciones, atómicos (vocabulario 151.665 a 151.679); mean-init de embeddings de entrada y salida
Hardware	NVIDIA A100-SXM4 40 GB; torch 2.9.1+cu128; entorno en pip_freeze.txt
Generación	temperatura 0,7; top-p 0,95; tope 2.048 tokens nuevos; corte en el marcador de nueva especificación; lotes de 20 (un lote = una tarea)

Fuente: config.py, train.py, ledgers y manifiestos del repositorio (commit 15ff05c0).

5.5 Métricas

La métrica sintáctica considera éxito cuando la salida contiene una expresión completa y supera parsing, round-trip y warm-up. La supervivencia exige además que produzca posiciones y pase los gates definidos en el mundo sintético experimental (screen de Sharpe y e-process). Para una tarea con n muestras y c éxitos, el estimador pass@k utilizado

es el definido por Chen et al. (2021): $1 - C(n-c,k)/C(n,k)$, siempre que $n-c \geq k$; si no, el valor es uno.

Los intervalos se obtienen por bootstrap sobre tareas, no sobre muestras individuales. La economía de tokens se recalculó con los tokenizadores serializados y 12.240 salidas crudas. La fidelidad se midió sobre el AST de los supervivientes de B mediante detectores derivados de los atributos registrados.

5.6 Resultados de generación

El funnel muestra que ambos brazos parten del mismo volumen, pero divergen antes del backtest. B produjo 3.763 candidatos válidos; C produjo 949. El número de candidatos que alcanzó backtest fue 2.805 y 780, y los supervivientes fueron 2.785 y 774. Por muestra, B alcanzó 61,5 % de validez y 45,5 % de supervivencia; C alcanzó 15,5 % y 12,6 %.

Tabla 8. Resultados globales del funnel F0.5.

Etapas	Brazo B	Brazo C	Lectura
Muestras	6.120	6.120	Mismo presupuesto de salidas.
Expresiones válidas	3.763 (61,5 %)	949 (15,5 %)	La mayor pérdida de C ocurre antes del backtest.
Backtests ejecutados	2.805	780	Solo candidatos de posición alcanzan esta etapa.
Supervivientes	2.785 (45,5 %)	774 (12,6 %)	20 fallos de screen en B y 6 en C entre los backtesteados.
Tareas con algún superviviente	37 de 40	24 de 40	En held-out: 29 de 30 y 19 de 30.

Fuente: elaboración propia.

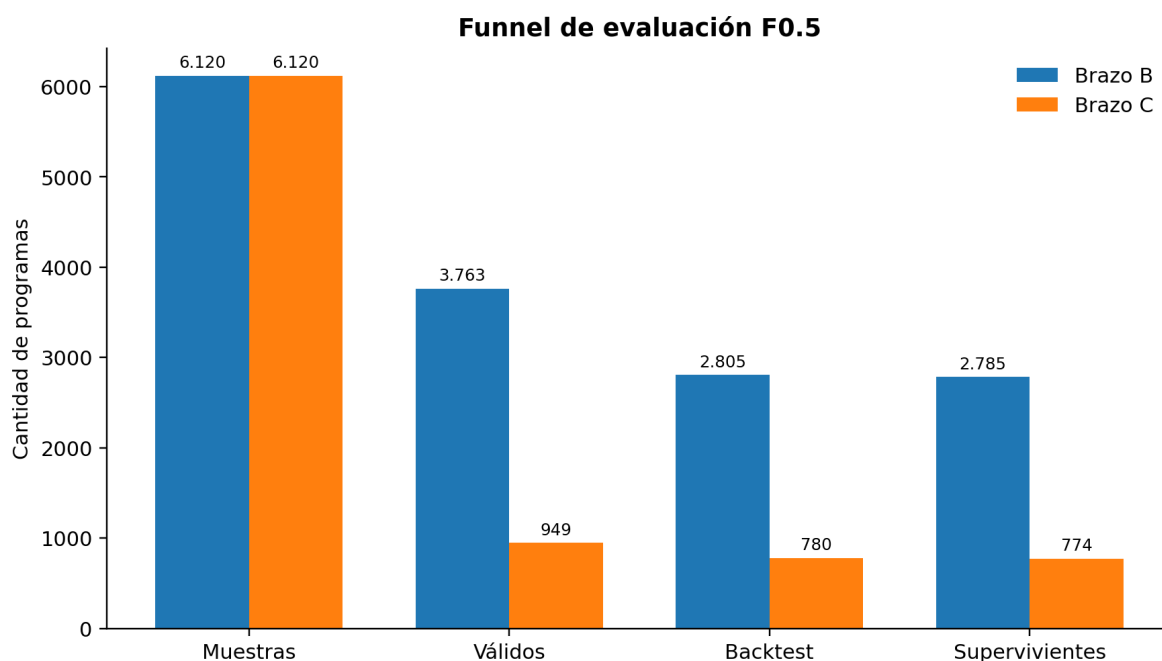


Figura 4. Funnel de evaluación F0.5.

Fuente: elaboración propia sobre report_f05a.json.

En tareas no vistas, B alcanzó pass@5 sintáctico de 0,8999 y pass@5 de supervivencia de 0,7726. C alcanzó 0,4374 y 0,3579. Los intervalos de supervivencia no se superponen. El criterio prerregistrado exigía una ventaja de C; la diferencia observada fue negativa en 41,5 puntos porcentuales.

Tabla 9. Resultados pass@5 sobre tareas no vistas.

Métrica	Brazo B	Brazo C
pass@5 sintáctico	0,8999; IC 95 % [0,8338; 0,9572]	0,4374; IC 95 % [0,3127; 0,5621]
pass@5 de supervivencia	0,7726; IC 95 % [0,6495; 0,8795]	0,3579; IC 95 % [0,2293; 0,4939]
Éxitos de supervivencia por muestra	2.275 de 4.590	661 de 4.590
Decisión Gate 0	Baseline de referencia.	No supera B por 5 puntos; Gate 0 rojo.

Fuente: elaboración propia sobre report_f05a.json; intervalos por bootstrap sobre 30 tareas.

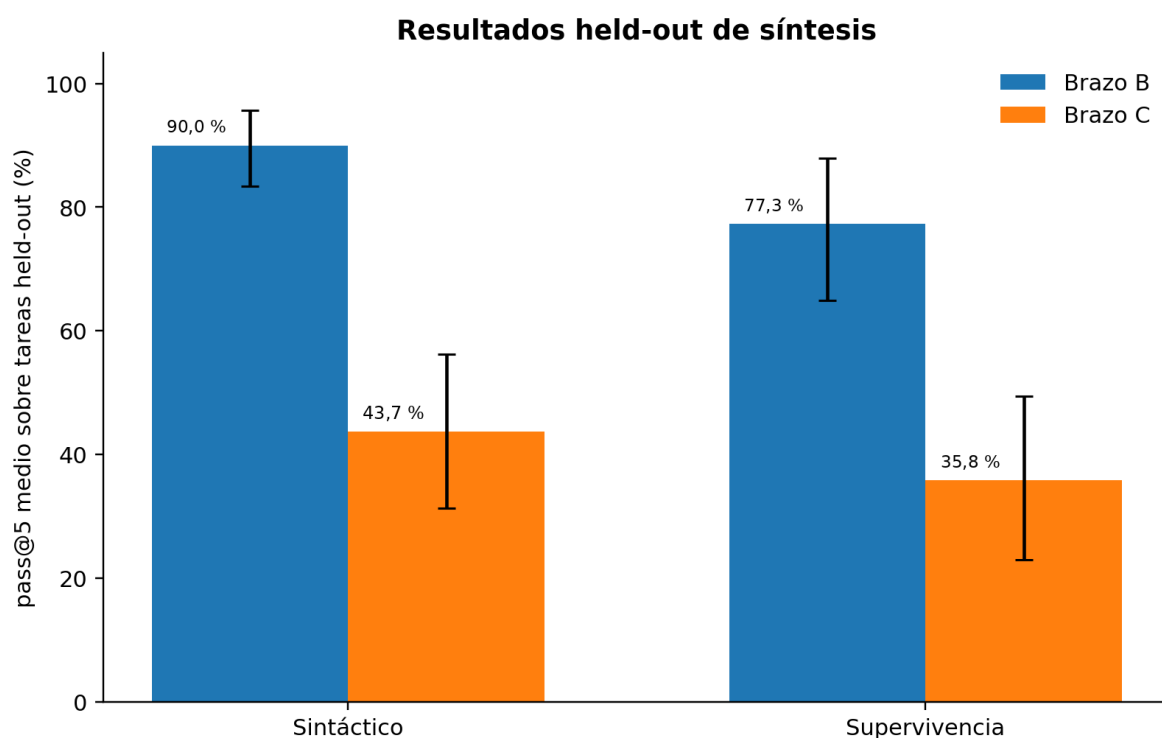


Figura 5. Resultados held-out de síntesis.

Fuente: elaboración propia sobre report_f05a.json; barras de error: IC 95 % por bootstrap de tareas.

La comparabilidad de B fue aprobada antes de leer la comparación principal. Su pass@5 sintáctico held-out fue 0,8999 frente a una referencia de 0,9028; la diferencia pareada fue -0,0029, con intervalo [-0,0363; 0,0320], dentro de la banda prerregistrada. Esta puerta confirma que el procedimiento reproduce su propia medición entre sesiones.

Análisis crítico de los gates del mundo sintético. Una vez alcanzado el backtest, casi todos los candidatos sobreviven: 2.785 de 2.805 en B (99,3 %) y 774 de 780 en C (99,2 %). El filtro efectivo del pipeline es previo: parsing, round-trip, warm-up y el requisito de producir posiciones eliminan el 54,2 % de las muestras de B y el 87,3 % de C, mientras que el screen de Sharpe y el e-process descartan en conjunto solo 20 y 6 candidatos, respectivamente. La causa es la construcción del mundo: la serie sintética combina una tendencia determinista con una oscilación periódica, de modo que casi cualquier programa que produzca posiciones correlacionadas con el precio captura la tendencia. Con esa señal, los umbrales quedan holgados por órdenes de magnitud: en la tarea auditada t20, los Sharpe

in-sample observados van de 14,285 a 29,958 frente al umbral 0,5, y los $e_{\max}$ de $1,05 \cdot 10^6$ a $1,13 \cdot 10^8$ frente al umbral 20.

Los gates operan, por lo tanto, como filtros de sanidad —descartan programas degenerados, sin posición útil o sin ventaja alguna— y no como una vara de selección exigente. Por esa razón, en este documento la palabra supervivencia debe leerse siempre como supervivencia bajo los criterios definidos en el mundo sintético experimental: describe una propiedad del pipeline de evaluación, no una propiedad financiera. La tendencia fuerte introduce además una limitación direccional: infla la proporción de supervivientes y comprime la diferencia entre estrategias buenas y mediocres. Ninguna cifra de este capítulo demuestra rentabilidad ni validez financiera real; establecerla exigiría datos de mercado, validación walk-forward, más instrumentos y costos representativos (sección 8.4).

5.7 Economía de tokens

La auditoría distingue tokens persistidos, tokens generados reconstruidos, candidatos válidos y supervivientes. La reconstrucción agrega el EOS cuando corresponde y respeta el límite de 2.048. Se validó el round-trip de los 12.240 registros sin discrepancias.

Entre candidatos válidos, B utilizó en promedio 115,9 tokens BPE y C 76,3. La reducción media fue 34,2 %; la mediana pasó de 96 a 73, una reducción de 24,0 %. La biblioteca y el tokenizador, por lo tanto, sí producen una representación más compacta cuando la generación se completa.

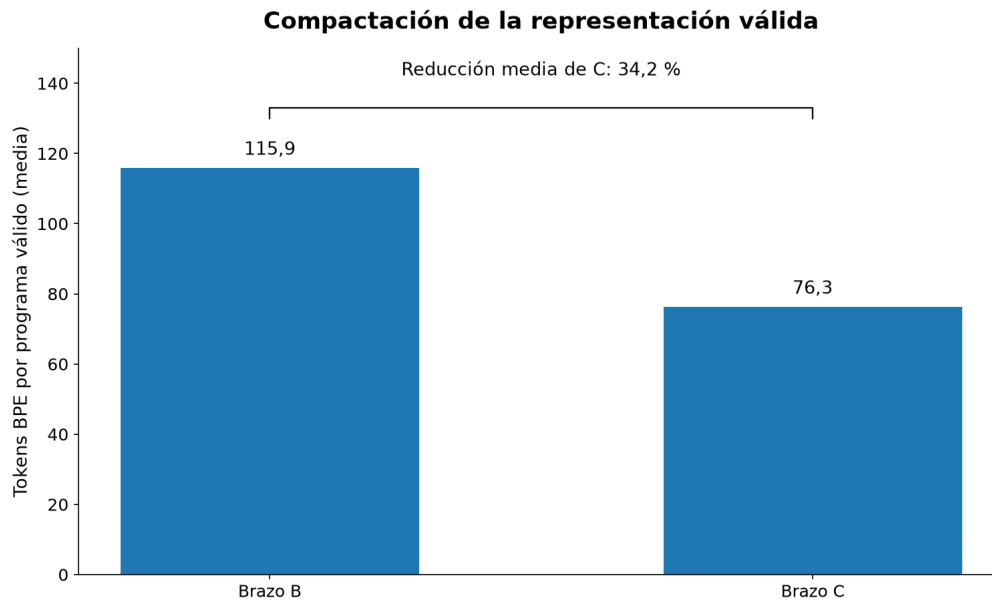


Figura 6. Compactación de la representación válida.

Fuente: auditoría exacta sobre raw_f05-arm-b.jsonl, raw_f05-arm-c.jsonl y el tokenizador serializado.

El resultado extremo a extremo tiene signo opuesto. B generó aproximadamente 1,20 millones de tokens y C 2,80 millones. C alcanzó 1.098 veces el límite de generación, frente a 160 en B. El costo reconstruido fue 430 tokens por superviviente en B y 3.623 en C. La compactación válida no compensó la cola de salidas degeneradas.

Tabla 10. Auditoría de economía de tokens.

Métrica	Brazo B	Brazo C	Relación o diferencia
Tokens generados reconstruidos	1.197.588	2.804.432	C/B = 2,34x
Media por muestra	195,7	458,2	C/B = 2,34x
Media por candidato válido	115,9	76,3	C reduce 34,2 %
Mediana por candidato válido	96	73	C reduce 24,0 %
Tokens por válido	318	2.955	C/B = 9,29x
Tokens por superviviente	430	3.623	C/B = 8,43x
Muestras en el límite 2.048	160	1.098	C/B = 6,86x

Fuente: f05_token_audit.json; conteo BPE con tokenizadores serializados y archivos crudos verificados por digest.

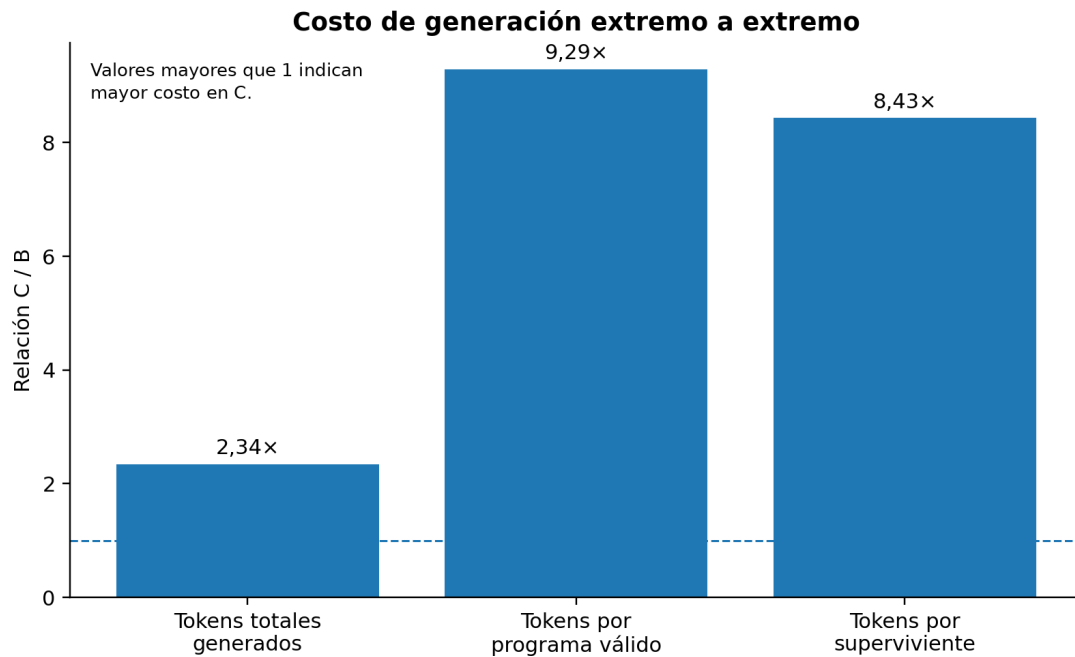


Figura 7. Costo de generación extremo a extremo.

Fuente: elaboración propia sobre f05_token_audit.json.

La afirmación respaldada es acotada: el vocabulario coevolutivo compacta los programas válidos, pero la configuración evaluada no reduce el costo total por solución útil. Esta formulación evita confundir una propiedad del lenguaje con la fiabilidad del modelo.

Las cifras de costo 2,34 veces el total y 8,43 veces por superviviente se calculan con el modelo de costo prerregistrado de COST.md, que estima el tiempo por paso de generación en régimen de lote máximo ($s_{\text{step}} = 0,032831$ s por paso, derivado de 864,6725 s sobre 26.337 pasos) y no con el tiempo de reloj de pared observado. Como verificación empírica independiente, el tiempo de reloj de pared (wall_s ; sección 7.1) da un costo total 2,394 veces mayor y 8,61 veces por superviviente (4,26 s contra 36,66 s); la diferencia frente a 2,34/8,43 se debe a que uno y otro modelo miden magnitudes distintas del mismo proceso, y COST.md se mantiene como la referencia prerregistrada.

5.8 Fidelidad especificación-programa

La inspección de ejemplos mostró que un programa podía superar el backtest sin implementar la estructura solicitada. Se aplicaron detectores sobre los 2.785 supervivientes de B. Cada detector busca una condición deliberadamente amplia asociada con uno de los seis atributos registrados. El procedimiento es post-hoc, pero utiliza etiquetas previas a la generación y no modifica resultados económicos.

La forma de salida fue el eje mejor preservado, con 86,9 %. La cadena de topes y la topología fueron los más débiles, con 40,6 % y 43,6 %. Solo 106 programas, 3,8 %, cumplieron simultáneamente los seis ejes. En held-out, combinar supervivencia y fidelidad redujo pass@5 de 0,7726 a 0,0331 y las aceptaciones de 2.275 a 97.

Esta caída proviene en su totalidad de una única tarea de las treinta held-out. Una partición de tarea única reproduce exactamente 0,0331; dos tareas producirían 0,0570, tres 0,0694, cinco 0,0828 y diez 0,0941, de modo que el valor observado solo es consistente con una tarea. El intervalo de confianza bootstrap con semillas 0, 1 y 20260804 es [0,0000; 0,0994]. El hallazgo para esta sección es, por lo tanto, que de las 30 tareas held-out exactamente una — t20, analizada en la sección 5.9 — produjo programas que satisfacen los seis ejes.

Tabla 11. Fidelidad estructural por eje.

Eje	Programas que satisfacen el eje
Familia de señal	1.453 de 2.785 (52,2 %)
Topología de combinación	1.213 de 2.785 (43,6 %)
Filtro	1.955 de 2.785 (70,2 %)
Cadena de topes	1.132 de 2.785 (40,6 %)
Amortiguación	2.087 de 2.785 (74,9 %)
Forma de salida	2.419 de 2.785 (86,9 %)
Seis ejes simultáneamente	106 de 2.785 (3,8 %)

Fuente: análisis estructural sobre los supervivientes del brazo B. El valor conjunto es un techo bajo los detectores utilizados; la auditoría manual de la sección 5.8 valida cuatro de los seis ejes con concordancia total.

La debilidad de los ejes topología (43,6 %) y cadena de topes (40,6 %) tiene una explicación estructural verificable en el corpus. Un censo de primitivas sobre los 499 programas distintos supervivientes held-out muestra que la primitiva `if_then_else` no aparece en ningún programa del corpus, lo que elimina por construcción las tareas que requieren una zona muerta condicional; `to_contracts` aparece en un único programa de los 499, lo que explica la falla sistemática en las tareas que exigen forma de salida en contratos; y `mean` aparece en solo 17 de los 499 programas, lo que explica la falla dominante en tareas que requieren estandarización tipo z-score. El modelo, en cambio, concentra su repertorio en `ema 8 / ema 32` y en el retorno logarítmico escalado por `sqrt 252`, y los aplica incluso cuando la especificación pide una familia distinta.

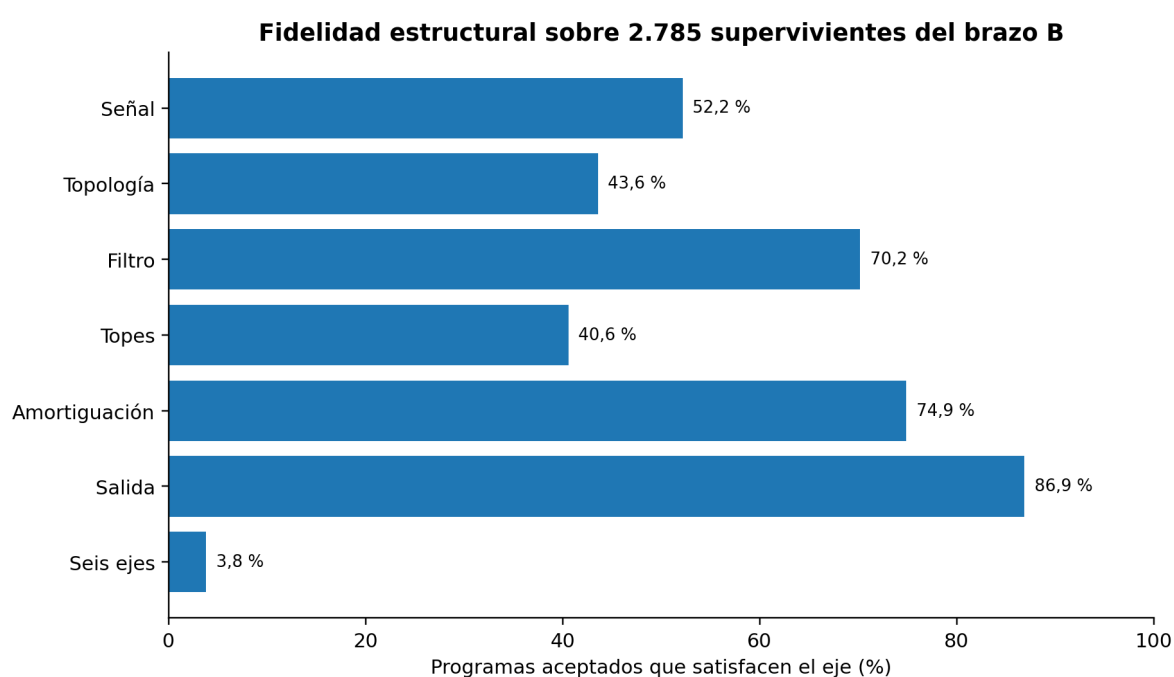


Figura 8. Fidelidad estructural sobre los programas supervivientes del brazo B.

Fuente: elaboración propia sobre el análisis de AST y los `structure_tags` registrados.

El hallazgo no implica que el backtester sea incorrecto. El backtester responde a una pregunta distinta: si el programa supera los umbrales del mundo de evaluación. La fidelidad responde si la estructura coincide con la especificación. Ambas preguntas deben reportarse por separado en un trabajo de síntesis automática. La validación de los detectores frente a un

revisor humano se realizó sobre una muestra estratificada (sección 5.8) y arrojó concordancia total en los cuatro ejes contrastables; su extensión a un segundo evaluador y a los dos ejes sin contraparte automática queda como trabajo futuro.

Auditoría manual de los detectores. Para contrastar los detectores estructurales con el criterio de un revisor humano se ejecutó una auditoría manual sistemática sobre una muestra estratificada por tarea de los supervivientes del brazo B: 57 programas distribuidos en 29 de las 30 tareas held-out, dos por tarea salvo t31, que aportó uno, y con t34 excluida por no registrar supervivientes distintos. El revisor trabajó sobre una planilla ciega que contenía, para cada programa, la especificación en lenguaje natural, la forma canónica expandida y los seis atributos que la especificación exige, sin acceso al veredicto del detector, y registró para cada eje un juicio fiel o no fiel junto con una nota escrita que justifica la decisión; 56 de las 57 filas cuentan con nota, de 250 caracteres en promedio.

El resultado de esa auditoría es el siguiente. Sobre los cuatro ejes que cuentan con veredicto automático reconstruido —topología, filtro, amortiguación y forma de salida— la revisión humana produjo 228 juicios y coincidió con el detector en los 228: cero falsos positivos, entendidos como un eje que el detector acepta y el revisor rechaza, cero falsos negativos, entendidos como el caso inverso, y un kappa de Cohen de 1,000 en cada uno de los cuatro ejes. Los ejes de familia de señal y cadena de topes no admiten ese contraste porque su implementación original no pudo recuperarse, de modo que para ellos el veredicto humano es la única fuente y no es posible informar falsos positivos ni negativos; esa asimetría se declara en la sección 5.11.

Los veredictos humanos por eje sobre los 57 programas fueron: familia de señal 33 fieles (57,9 %), topología 31 (54,4 %), filtro 35 (61,4 %), cadena de topes 14 (24,6 %), amortiguación 41 (71,9 %) y forma de salida 42 (73,7 %). Un solo programa de los 57

satisface los seis ejes simultáneamente, y pertenece a la tarea t20. La auditoría manual, realizada a ciegas y sin acceso a los resultados agregados, converge así de forma independiente con el hallazgo de esta sección: t20 es la única tarea held-out de la muestra que produce programas fieles a su especificación.

El protocolo aplicado fue el siguiente: (i) muestra estratificada por tarea sobre los supervivientes de B; (ii) evaluación de cada programa contra los seis atributos registrados de su tarea, sobre la forma canónica y con los mismos criterios por eje que motivan los detectores; (iii) registro del veredicto por eje y en conjunto, acompañado de una nota justificativa; (iv) conteo de falsos positivos y falsos negativos tomando la revisión humana como referencia; y (v) medición de la concordancia mediante kappa de Cohen. No participó un segundo evaluador, de modo que no se informa concordancia entre anotadores y esa ausencia se declara como limitación en la sección 5.11. Las limitaciones conocidas de los detectores acotan la lectura de la Tabla 11: cada condición es deliberadamente amplia, por lo que puede aceptar coincidencias superficiales y rechazar implementaciones equivalentes pero sintácticamente inusuales. La concordancia perfecta observada en los cuatro ejes contrastables indica que, en el rango de programas efectivamente generados, esas condiciones amplias no produjeron discrepancias con el criterio humano; el 3,8 % conjunto sigue reportándose como un techo bajo los dos ejes sin contraparte automática.

5.9 Caso auditado t20

La tarea t20 fue estandarizar el precio contra su promedio móvil y desviación, y convertir la señal en posición. B produjo 146 muestras sobrevivientes. Los seis atributos son cumplidos por ocho programas, representando 97 muestras. Los programas se reejecutaron sobre las 900 barras y reprodujeron el veredicto original.

El criterio que separa los ocho programas fieles de los trece no fieles entre los 21 programas distintos de t20 es estructural y reproduce el veredicto completo: los programas

fieles emiten (mean (price \$0) 25) y (std (price \$0) 25), mientras que los no fieles sustituyen ese cálculo por (shift (price \$0) 25) o (shift (price \$0) 1), una forma que aprovecha la tendencia determinista del mundo sintético sin implementar el z-score solicitado. Esta única regla discrimina correctamente los 21 de 21 programas distintos de la tarea.

La vitrina conserva los tres tramos y muestra una observación metodológica. Los programas no fieles pueden obtener mejores métricas económicas que los fieles; el máximo Sharpe IS de los no fieles fue 29,958, mientras el grupo fiel se ubicó alrededor de 15,635. El mundo contiene una tendencia determinista y una oscilación periódica, por lo que esas magnitudes no son alcanzables como evidencia de rendimiento financiero. La figura demuestra ejecución end-to-end y la separación entre backtest y fidelidad.

Ejemplo completo de extremo a extremo. El recorrido de una muestra de t20 fija el ciclo entero. La especificación en lenguaje natural indica estandarizar el precio en un z-score contra su propia media y desviación móviles y convertir esa señal, con objetivo de volatilidad, en una posición. Un programa fiel generado por B es:

```
(lam (set_position (div (mul (clip (mul (div (sub (price $0)
(mean (price $0) 25)) (std (price $0) 25)) 2.0) (neg 20.0) 20.0)
equity) (mul (price $0) (point_value $0)))))
```

Su AST aplica, de adentro hacia afuera: el z-score del precio contra su media y desvío móviles de ventana 25; una ganancia $K = 2,0$; un recorte simétrico en ± 20 ; el escalado por capital y la conversión a contratos mediante el valor de punto. El análisis temporal deriva el warm-up del programa de su ventana de 25 barras y lo admite al backtest, que solo consume posiciones posteriores al onset. En el backtest, los 21 programas distintos supervivientes de t20 pasaron todos los controles de sanidad al reejecutarse sobre las 900 barras: Sharpe in-sample entre 14,285 y 29,958, e_max de holdout entre $1,05 \cdot 10^6$ y $1,13 \cdot 10^8$, multiplicación de capital entre 11,0 y 142,8 veces y drawdowns entre 0,03 % y 43,99 %. La evaluación de fidelidad acepta 97 de las 146 muestras supervivientes (66,4 %), correspondientes a ocho programas distintos que difieren solo en la ganancia K, con valores entre 0,8 y 5,3; los trece

programas distintos restantes superan el backtest sin implementar el z-score solicitado. El mejor Sharpe in-sample (29,958) pertenece a un grupo no fiel con desplazamiento temporal, mientras que el mejor fiel alcanza 15,635: el ejemplo materializa, dentro de una única tarea, la separación entre generar un programa válido, superar los gates del mundo sintético e implementar lo especificado.

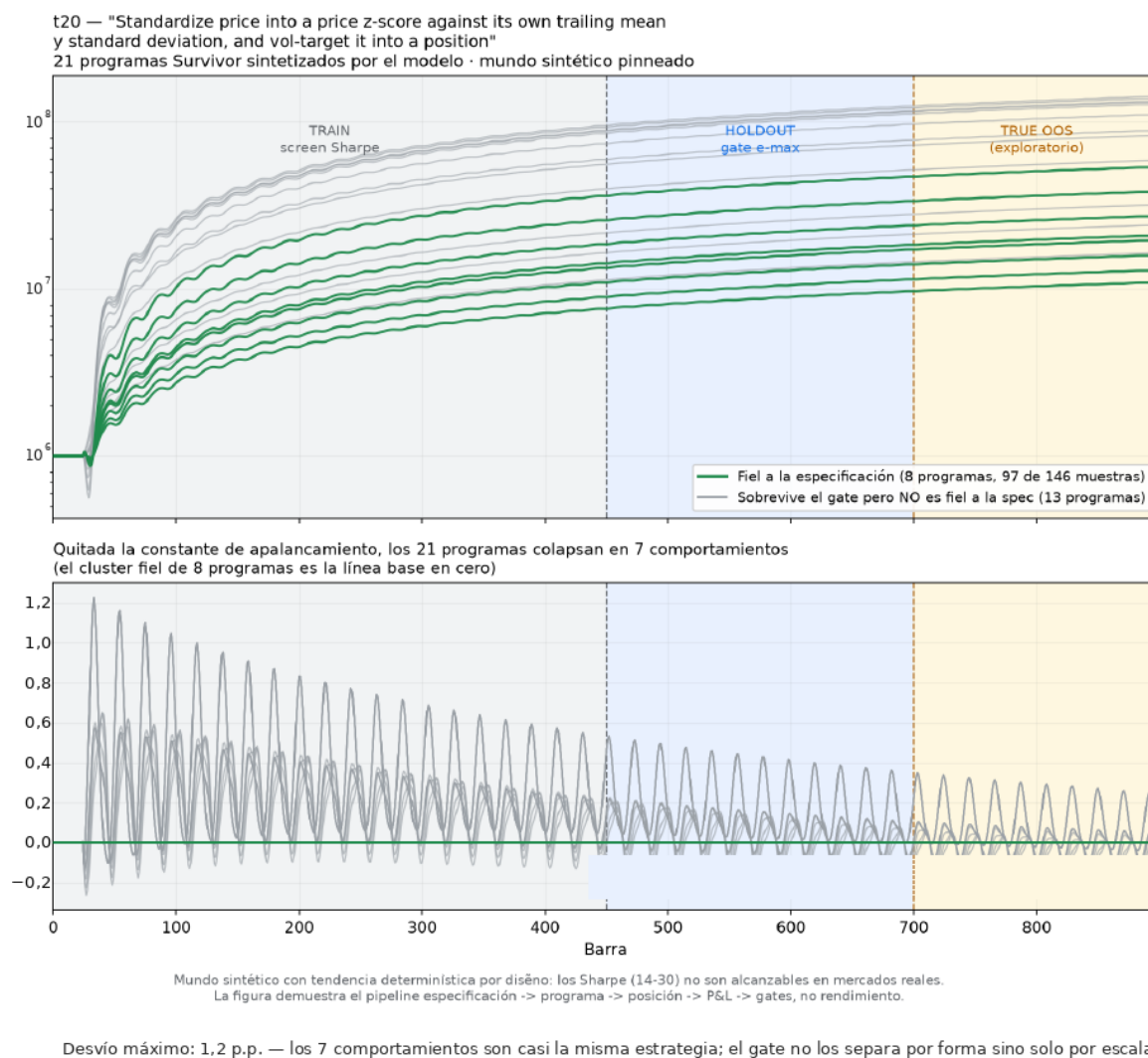


Figura 9. Curvas de capital de los 21 programas distintos de la tarea t20.

Fuente: elaboración propia sobre la vitrina reproducible integrada en el commit final. True-OOS es exploratorio y no participa del gate.

5.10 Diagnóstico del brazo coevolutivo

El modo de falla dominante de C fue `no_expr`: 3.599 de 6.120 salidas no contenían una expresión completa. El diagnóstico separó terminación temprana y repetición. Un grupo amplio (2.492 salidas, el 69 %) finalizó por EOS con programas cercanos a completarse; el resto (1.107 salidas, el 31 %) alcanzó el límite de 2.048 tokens mediante patrones repetitivos; en B, el mismo modo de falla apareció en 812 salidas. Las colas degeneradas incluyeron tanto tokens del vocabulario añadido como fragmentos inexistentes en el lenguaje, de modo que la falla dominante no fue la invención aislada de nombres, sino la incapacidad de cerrar expresiones.

Un control con preámbulo enumeró las 14 abstracciones, sus aridades y la convención de uso. Aumentó las invocaciones, pero redujo el `pass@5` de supervivencia held-out a 0,1967. Mostrar la biblioteca no resolvió el problema. La evidencia apunta a una internalización incompleta de la sintaxis de llamadas y a un entrenamiento sobre corpus densificado que alteró la distribución de longitud.

La conclusión de Gate 0 se mantiene sin reinterpretación: C no supera B y la coevolución evaluada no constituye el camino principal para esta versión. La biblioteca permanece como optimización verificable; su selección futura puede trasladarse al compilador o combinarse con generación restringida.

Un control adicional (Anexo A, informe generado el 26 de julio de 2026 a partir de un cuaderno del 25 de julio de 2026) probó si la ausencia de biblioteca en el contexto explicaba la falla del brazo C: se repitió la generación del brazo C con las 14 abstracciones descritas en el prompt mediante un preámbulo de biblioteca. El resultado fue negativo: la supervivencia held-out cayó de 0,3579 (661 aciertos) a 0,1967 (348 aciertos), y las tareas held-out con algún acierto pasaron de 19 a 14; el modo de falla dominante también se agravó (`parse_fail` pasó de 969 a 4.834 sobre el histograma sintáctico expandido). Mostrar la biblioteca no resolvió el

problema —lo empeoró—, lo que descarta que la ausencia de biblioteca en contexto sea la causa de la falla de C. Debe declararse un desvío respecto del resto del trabajo: esta corrida del Anexo A se ejecutó en una GPU NVIDIA A100-SXM4 de 80 GB, mientras que el resto de las corridas declaradas en esta tesis usó una A100-SXM4 de 40 GB.

5.11 Limitaciones

La evaluación utiliza un mundo sintético de tendencia fuerte. Permite probar causalidad, gates y reproducibilidad, pero no inferir rentabilidad. Las 40 tareas fueron usadas para desarrollo y diagnóstico, de modo que nuevas afirmaciones confirmatorias requerirían otro conjunto. El corpus presenta baja cobertura de primitivas de cartera, lo que afecta fidelidad. Respecto de la validación de los detectores estructurales, la auditoría manual sistemática descrita en la sección 5.8 se ejecutó sobre 57 programas de 29 tareas y arrojó concordancia total en los cuatro ejes contrastables, pero subsisten dos limitaciones: no participó un segundo evaluador, por lo que no se informa concordancia entre anotadores, y los ejes de familia de señal y cadena de topes carecen de veredicto automático reconstruido, de modo que para ellos la revisión humana es la única fuente y no admite conteo de falsos positivos ni negativos. Finalmente, los resultados de C dependen del modelo, la configuración y el procedimiento de entrenamiento utilizados.

A esta limitación se agrega una segunda, de procedencia: la implementación original de los seis detectores estructurales no se encontró en ningún artefacto entregado, y una reconstrucción independiente sobre los 2.785 supervivientes del brazo B reprodujo de forma exacta cuatro de los seis ejes (filtro 1.955, amortiguación 2.087, forma de salida 2.419 y topología 1.213), pero no reprodujo familia de señal (entre 1.448 y 1.458 según la regla de desempate, contra 1.453) ni cadena de topes (máximo alcanzable de 1.102 contra 1.132). Por lo tanto, la cifra conjunta de 106 supervivientes que cumplen los seis ejes (3,8 %) se reporta

tal como fue producida por el detector original, sin verificación independiente completa de su implementación.

La factibilidad del lenguaje base impone además un límite que no depende del modelo. Bajo el presupuesto de generación `l_gen` de 4.096 tokens, los envoltorios de dimensionamiento con tope nocional (`notional_cap`) requieren 4.511 tokens en el lenguaje base, la cadena completa de dimensionamiento (`full_chain`) requiere 27.290, y la cadena completa amortiguada (`full_chain_buffered`) requiere 109.090; las tres categorías exceden el presupuesto entre 1,10 y 26,63 veces. Diecisiete de las treinta tareas held-out eran, por lo tanto, imposibles por construcción en el lenguaje base y no por una falla del modelo. Restringido a las trece tareas held-out factibles, la misma aceptación de `t20` produce una estadística secundaria de 0,0765 (0,9942 dividido 13) sobre el grupo `held_out_feasible`, prerregistrado en GROUPS; esta cifra no reemplaza al 0,0331 prerregistrado sobre `held_out`, pero se reporta como verificación adicional. La factibilidad tampoco garantiza supervivencia: de las trece tareas held-out factibles, `t34` tuvo cero programas distintos supervivientes y `t31` tuvo uno solo, ambas con un costo de 737 tokens frente al presupuesto de 4.096; `t20`, la única tarea que produjo programas fieles, es factible con 487 tokens.

6. Pruebas y validación

6.1 Estrategia de validación

La validación del software se separa de la evaluación científica. Las pruebas comprueban que el parser, el intérprete, el warm-up, el backtester y la infraestructura implementen sus contratos. El experimento, en cambio, mide el desempeño de dos configuraciones del generador. Un resultado experimental negativo no convierte en incorrecto el artefacto; una suite verde tampoco valida la hipótesis de coevolución.

6.2 Suite automatizada

El cierre final ejecutó `cargo test --quiet -- --test-threads=1`: se descubrieron 426 pruebas, 423 pasaron, ninguna falló y tres quedaron ignoradas. Cuatro pruebas específicas de la vitrina pasaron. `cargo clippy --all-targets --all-features -- -D warnings` y `cargo fmt --check` finalizaron con código cero. Dos pruebas de integridad del frozen core pasaron y el diff directo de archivos protegidos quedó limpio.

Tabla 12. Verificación del artefacto final.

Control	Resultado
Suite Rust	426 descubiertas; 423 pasadas; 0 fallidas; 3 ignoradas.
Pruebas de vitrina	4 pasadas; 0 fallidas.
Clippy	Sin advertencias con <code>-D warnings</code> .
Formato	<code>cargo fmt --check</code> con salida 0.
Frozen core	2 pruebas pasadas; diff protegido limpio.
Python	Compilación y validador de procedencia con salida 0.
Regeneración	CSV, JSON y PNG generados dos veces con igualdad byte a byte.
Inventario vitrina	21 programas únicos y 146 muestras validadas.

Fuente: close-out del commit final y artefactos de reproducción.

6.3 Oráculos y pruebas diferenciales

La suite contiene comparaciones contra implementaciones independientes, pruebas property-based, casos borde, digestos de corpus y mutaciones deliberadas. Las abstracciones de Stitch se expanden y se evalúan contra el programa original. El warm-up se compara con tarjetas simbólicas y sondas. Las pruebas diferenciales reducen el riesgo de que una única implementación se valide a sí misma.

El backtester canónico vive en Rust. Los scripts Python agregan una segunda implementación para métricas y agregación. Los manifiestos fijan las entradas y los oráculos recalculan resultados desde archivos crudos. La existencia de dos rutas no elimina toda posibilidad de error, pero vuelve observables las divergencias.

6.4 Integridad y portabilidad

La reproducibilidad entre macOS y Linux exigió controlar funciones matemáticas. Una diferencia de último bit en logaritmos modificaba digestos y se resolvió mediante libm. El cambio se documentó como re-freeze porque alteraba resultados numéricos. Esta disciplina evita declarar equivalencia donde la implementación cambió.

6.5 Validación de la vitrina

La rama t20 fue endurecida antes de integrarse. Se extrajo una función compartida del evaluador canónico y se añadieron pruebas de equivalencia de prefijo, métricas y splits. El script de extracción valida 21 programas y 146 muestras. La generación del CSV, el JSON de estadísticas y el PNG fue determinista en dos ejecuciones consecutivas. True-OOS permanece rotulado como exploratorio y no se utiliza para selección.

6.6 Alcance de las pruebas

Dos pruebas de Stitch quedaron ignoradas porque el binario compress pineado no estaba disponible localmente; la tercera genera una fixture v11 documentada. Esta limitación se informa sin convertir los tests ignorados en pasados. La suite demuestra consistencia del artefacto bajo sus fixtures, no ausencia absoluta de defectos. Tampoco sustituye una evaluación sobre datos reales o una auditoría externa del criterio de fidelidad.

7. Infraestructura experimental y reproducibilidad

7.1 Entornos de ejecución

La implementación y las verificaciones locales se realizaron en Rust y Python. La generación F0.5 se ejecutó en Google Colab sobre una NVIDIA A100-SXM4 de 40 GB. El modelo se cargó sin cuantización de 4 bits, con bfloat16 y backend SDPA. El entorno Python quedó registrado mediante `pip_freeze.txt`. El toolchain Rust y Cargo.lock forman parte del repositorio.

El tiempo de generación quedó registrado en el campo `wall_s` de los checkpoints, no documentado hasta esta versión: el brazo B insumió 11.852,0 s (3,292 h), el brazo C 28.374,7 s (7,882 h), y el total de generación fue 11,174 h. Debe además distinguirse la precisión de entrenamiento de la de generación: el ajuste del modelo (sección 5.4) se realizó con QLoRA, es decir, con el modelo base cuantizado a 4 bits mediante bitsandbytes, mientras que la generación F0.5 aquí descrita cargó el modelo sin cuantización de 4 bits, en bfloat16. Ambas afirmaciones son correctas y no son contradictorias: describen etapas distintas del mismo flujo de trabajo.

7.2 Artefactos y digests

Cada brazo produce un checkpoint, un JSONL crudo y un JSONL scoreado. `digests.json` registra FNV-1a/64; la auditoría de tokens agregó SHA-256. Los archivos crudos contienen 6.120 registros por brazo. El orden (`prompt_id`, `sample_idx`) coincide uno a uno con los scoreados. El tokenizador extendido tiene 11.424.633 bytes y hash SHA-256 3828632a...5517f.

7.3 Repositorio y versionado

El repositorio privado del proyecto conserva la historia de ingeniería. Después del cierre, `master` y `origin/master` apuntan a 15ff05c07d281d703c7b472b4c3eee7600b60529. Las ramas de hitos se reemplazaron por tags anotados: `milestone/etapa1a`, `milestone/f02-bakeoff`,

milestone/f04a, milestone/f04c, milestone/f04f y milestone/vitrina-t20. La política final establece una sola rama de larga vida.

7.4 Procedimiento de reproducción

La reproducción completa requiere el repositorio, el modelo base en la revisión indicada, los adaptadores, el tokenizador, los JSONL y el binario Stitch cuando se pretende regenerar la biblioteca. La reproducción de tablas y gráficos de esta tesis no requiere nueva generación: parte de los JSONL y manifiestos congelados. Los comandos y hashes principales se incluyen en el Apéndice E.

Cuando se incluye la planilla concordancia_t20_ciego.csv como material de apoyo, debe leerse como una verificación de autoconsistencia interna del detector automático de fidelidad: ambos lados de la comparación —el orden barajado y la clave de desciframiento— fueron generados por la misma herramienta de detección, por lo que el porcentaje de acuerdo no constituye una validación independiente por una persona distinta del detector.

Versión congelada para el jurado. El commit 15ff05c07d281d703c7b472b4c3eee7600b60529 fija el código de la versión evaluada. El paquete de retorno olaair_f05_return.zip, versionado dentro del repositorio en f05_eval/, contiene los JSONL crudos y scoreados de ambos brazos (6.120 registros por brazo y por tipo), gate_d1.json, report_f05a.json, digests.json y pip_freeze.txt. El tokenizador extendido (prepared-tokenizer.json) y los adaptadores LoRA se entregan con el paquete de evaluación junto con sus SHA-256 (Tabla 17). Las dependencias quedan fijadas por Cargo.lock, rust-toolchain.toml y pip_freeze.txt. Los scripts de la vitrina (extract_t20_survivors.py, el binario t20_equity y plot_t20.py) reconstruyen la tabla y la figura del caso auditado desde los artefactos congelados. El procedimiento reducido ejecutable durante la defensa consiste en los seis comandos del Apéndice E: no requiere GPU ni red, verifica la suite completa y la integridad del núcleo congelado, y reejecuta la vitrina t20 de punta a punta.

7.5 Restricciones de acceso

El repositorio y los pesos de adaptadores son privados. Para la instancia de evaluación se ofrece al jurado acceso temporal de lectura al repositorio en el commit congelado o, de manera equivalente, la entrega del paquete completo de evaluación descrito en la sección 7.4 (código, artefactos, tokenizadores, adaptadores, manifiestos y hashes); una reproducción externa completa fuera de ese marco debe ser aprobada por el autor. El manifiesto de la exhibición declara nulos los campos de procedencia cuyo JSONL autorizado no estaba disponible en el momento del cierre. Esta ausencia se registra y no se compensa por inferencia.

Dos archivos adicionales, `f03c_phase_b.json` y `outcomes_eval.jsonl`, no pudieron recuperarse pese a la búsqueda y se declaran ausentes de forma permanente; ningún resultado de esta tesis depende de ellos, y su ausencia se documenta aquí por completitud del inventario de artefactos.

8. Conclusiones

8.1 Síntesis del trabajo

Se diseñó e implementó Pucará, un lenguaje para estrategias de trading cuyo principal consumidor es una inteligencia artificial. El trabajo produjo un núcleo ejecutable, un intérprete determinista, análisis temporal, backtesting, aprendizaje de biblioteca, un tokenizador extendido, un modelo ajustado y una infraestructura de evaluación reproducible. La versión final del software superó 423 pruebas automatizadas y preservó la integridad del núcleo.

El brazo estático demostró que un modelo local de 7B puede aprender un lenguaje inexistente en su preentrenamiento y resolver tareas no vistas. Su pass@5 sintáctico held-out fue 0,8999 y su pass@5 de supervivencia 0,7726. Estos valores se interpretan por tarea y no como tasas individuales. Por muestra, B produjo 61,5 % de expresiones válidas y 45,5 % de supervivientes.

El brazo coevolutivo produjo programas válidos 34,2 % más cortos en promedio. Esa es evidencia positiva sobre la representación. Sin embargo, generó demasiadas salidas incompletas y repetitivas: consumió 2,34 veces más tokens totales y 8,43 veces más tokens por superviviente. El experimento muestra que un lenguaje puede ser más compacto sin que el generador sea más eficiente.

La evaluación de fidelidad aportó la conclusión central del trabajo. Solo 106 de 2.785 supervivientes de B cumplieron simultáneamente los seis atributos de la especificación, y el pass@5 held-out cayó de 0,7726 a 0,0331 al exigir fidelidad además de la supervivencia bajo los criterios del mundo sintético experimental. La distancia entre generar un programa sintácticamente válido, superar los gates del mundo sintético e implementar fielmente lo especificado es la brecha que este trabajo cuantifica. El backtester verificó ejecución y

desempeño bajo el mundo, pero no intención. La síntesis automática necesita una puerta estructural adicional.

8.2 Decisiones técnicas clave

La elección de una forma canónica prefija permitió unificar generación, parsing, hashing, compresión e interpretación. Los índices de De Bruijn evitaron nombres de variables y simplificaron sustitución. La separación entre núcleo congelado y biblioteca elaborable protegió la semántica durante la evolución. La truncación física del true-OOS convirtió una convención metodológica en una restricción estructural.

Otra decisión central fue medir tokens con los tokenizadores serializados. Contar caracteres habría mostrado compactación, pero no el costo real del modelo. La auditoría BPE permitió distinguir programas válidos más cortos de una generación total más cara. Finalmente, mantener el resultado prerregistrado de Gate 0 evitó convertir un límite experimental en una afirmación positiva artificial.

8.3 Contribuciones

La contribución de ingeniería es un DSL operativo con semántica determinista, causalidad, backtesting, integridad versionada y tooling de reproducción. La contribución empírica es la medición de un modelo de 7B sobre 12.240 generaciones compute-matched. La contribución metodológica es la separación explícita de compactación, eficiencia extremo a extremo y fidelidad a la especificación.

El aporte positivo no depende de afirmar que C funcionó. Pucará muestra que una representación centrada en la IA puede ser compacta y verificable y que un modelo local puede lograrlo. El resultado negativo muestra que no fue suficiente: el reentrenamiento en un corpus densificado de nuevos tokens no garantizó llamadas correctas ni una finalización estable.

El resultado negativo del brazo coevolutivo constituye en sí mismo un hallazgo: una representación más compacta no produce necesariamente una generación más eficiente de extremo a extremo. La compactación medida sobre programas válidos (-34,2 % de tokens BPE) coexistió con un costo total 2,34 veces mayor y 8,43 veces más tokens por superviviente, porque la patología de terminación dominó la economía. Debe subrayarse además el alcance de las comparaciones: ninguna comparación con Python, Pine Script u otro lenguaje convencional formó parte del diseño experimental, por lo que esta tesis no afirma que Pucará sea más eficiente que esos lenguajes. Las afirmaciones sostenidas se limitan a comparaciones internas entre brazos y a propiedades verificadas del artefacto, y se consolidan junto a su evidencia en la Tabla 13.

Tabla 13. Afirmaciones principales y evidencia que las sostiene.

Afirmación	Evidencia experimental	Fuente
Un modelo local de 7B aprende un DSL ausente de su preentrenamiento	pass@5 sintáctico held-out 0,8999; 3.763/6.120 salidas válidas en B	report_f05a.json; Tablas 8 y 9
Los programas generados superan los gates del mundo sintético	pass@5 de supervivencia held-out 0,7726; 2.785 supervivientes	report_f05a.json; Tablas 8 y 9
Superar los gates no implica implementar la especificación	pass@5 con fidelidad 0,0331; 106/2.785 fieles en los seis ejes (3,8 %)	Detectores sobre AST; Tabla 11
La biblioteca y el tokenizador compactan los programas válidos	-34,2 % de tokens BPE medios por programa válido (115,9 a 76,3)	f05_token_audit.json; Tabla 10
La compactación no reduce el costo extremo a extremo	C consumió 2,34x tokens totales y 8,43x tokens por superviviente	f05_token_audit.json; Tabla 10
La coevolución evaluada no superó al lenguaje estático	Diferencia de -41,5 puntos en supervivencia held-out; 0/949 salidas válidas de C usan abstracciones	report_f05a.json; sección 5.10
Los gates discriminan poco tras el backtest	Supervivencia post-backtest de 99,3 % (B) y 99,2 % (C)	JSONL scoreados; sección 5.6
La medición es reproducible	Comparabilidad de B: diferencia pareada -0,0029 dentro de la banda prerregistrada; digests y reejecución t20	gate_d1.json; vitrina

Fuente: elaboración propia sobre artefactos congelados del repositorio.

8.4 Trabajo futuro

Las líneas de continuidad identificadas en este proyecto se presentan como un programa de evolución técnica, no como componentes ya implementados. Se clasifican según su relación con las limitaciones observadas y cuál debería hacerse primero para el desarrollo futuro de Pucará.

A corto plazo, se propone completar la orientación centrada en la IA del lenguaje mediante la tokenización atómica del vocabulario central estable, congelando la gramática y limitando la generación para evitar cierres prematuros, paréntesis incompletos y símbolos inválidos. Sobre esta base, se podrían incluir contratos estructurales y tipos mínimos en la decodificación. También se debe abordar la normalización de la escala de riesgo, la cobertura del sublenguaje de la cartera y el modo de falla del brazo coevolutivo. La evaluación futura debería utilizar tareas retenidas, medir tokens por programa fiel y complementar el verificador automático con revisión humana ciega.

Una segunda línea corresponde al sistema de garantías estáticas: tipos refinados para límites de riesgo, tiempo como tipo para expresar causalidad, tipos lineales o afines para impedir la duplicación accidental de capital y un sistema de efectos que separe simulación y ejecución real. En una etapa posterior, la latencia podría formar parte del contrato de tipos, de modo que el presupuesto temporal se compruebe durante la compilación.

La biblioteca aprendida podría mantenerse como optimización del compilador, sin exigir que el modelo memorice identificadores opacos. Esta evolución incluye una teoría ecuacional versionada, canonicalización mediante e-graphs, huellas conductuales, mecanismos de renta y recolección de abstracciones, re-canonización periódica, control de diversidad mediante un archivo de élites, actualización automática de nombres y descripciones, y reconstrucción del modelo desde el checkpoint base cuando cambie el vocabulario.

En materia de auditabilidad y validación externa, se prevé una proyección legible del programa, certificados de compilación y un segundo backtester independiente. La evaluación podría ampliarse con validación walk-forward, costos de transacción más representativos, datos de mercado reales, mayor cantidad de instrumentos y una separación estricta entre desarrollo, holdout y verdadero fuera de muestra. Estas extensiones deberán distinguir siempre la corrección de la implementación de la rentabilidad económica.

Finalmente, el entorno de ejecución podría evolucionar hacia una representación intermedia común, compilación nativa sobre Rust mediante Cranelift y ejecución aislada y portable mediante WebAssembly y Wasmtime. Las modificaciones operativas podrían resolverse mediante parámetros, tablas o máquinas de estados actualizables sobre un datapath estable. La integración con el motor existente y la eventual compilación de caminos críticos a FPGA quedan como objetivos de largo plazo, condicionados a que antes se consoliden la generación, la fidelidad y la verificación.

De acuerdo con la evidencia obtenida, el orden recomendado es avanzar primero en generación restringida y tipos mínimos; luego en saneamiento de la biblioteca y validación con datos reales; y solo después en compilación y aceleración especializada.

8.5 Supuestos y dependencias

El trabajo depende de la disponibilidad del modelo base, Transformers, Colab para la corrida histórica y el binario Stitch para regenerar bibliotecas. La reproducción de cifras finales depende de los JSONL y tokenizadores cuyas sumas de verificación se registraron. El dominio supone series discretas y rebalanceo barra a barra.

8.6 Restricciones y limitaciones

Tabla 14. Principales limitaciones y efecto sobre las conclusiones.

Limitación	Efecto
Mundo sintético con tendencia determinista.	No permite afirmar rentabilidad ni generalización financiera.
Un único modelo base de 7B.	La conclusión sobre C no se generaliza a todos los modelos.
40 tareas reutilizadas para diagnóstico.	Nuevas afirmaciones confirmatorias requieren otro benchmark.
Baja cobertura del sublenguaje de cartera.	Reduce la fidelidad en tareas que exigen agregación transversal.
Detectores de fidelidad sin validación humana.	La medición es estructural y debe interpretarse como techo bajo esas reglas.
Nueve primitivas no atómicas en BPE.	El objetivo de una operación por token no está completo.
Pesos y repositorio privados.	La reproducción externa requiere acceso autorizado.

Fuente: elaboración propia.

9. Declaración sobre el uso de inteligencia artificial

9.1 Herramientas utilizadas

Durante el desarrollo se utilizaron ChatGPT, de OpenAI, y Codex como asistentes de programación, análisis y documentación. Notion se utilizó como repositorio de decisiones y evidencia. Las herramientas de inteligencia artificial no se consideran autoras ni fuentes científicas; su función se declara para transparentar el proceso.

9.2 Finalidad de uso

La asistencia abarcó generación y revisión de código a partir de especificaciones del autor, exploración de alternativas, organización de documentación, revisión de consistencia y apoyo en la redacción. Las corridas de entrenamiento, generación, scoring, tests y auditorías produjeron los datos presentados. Los números no se obtuvieron mediante estimación textual del modelo.

9.3 Aporte propio

Corresponden al autor la formulación del problema, la decisión de diseñar un lenguaje IA-nativo, la definición del dominio, los criterios de freeze, la supervisión de la implementación, el diseño experimental, los umbrales prerregistrados, la ejecución de las corridas, la interpretación de los resultados y la responsabilidad académica. La asistencia de código operó bajo instrucciones y revisión del autor.

9.4 Salvaguardas y responsabilidad

Toda afirmación cuantitativa fue contrastada contra archivos o código. Las referencias bibliográficas se verificaron contra fuentes primarias. Las decisiones de alcance y presentación se registraron en una bitácora. El autor asume la responsabilidad por el contenido y deberá estar en condiciones de explicar el diseño, los experimentos, las limitaciones y cada resultado expuesto.

Referencias

- Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q. V., & Sutton, C. (2021). *Program synthesis with large language models* [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.2108.07732>
- Bailey, D. H., Borwein, J. M., López de Prado, M., & Zhu, Q. J. (2014). Pseudo-mathematics and financial charlatanism: The effects of backtest overfitting on out-of-sample performance. *Notices of the American Mathematical Society*, 61(5), 458–471. <https://doi.org/10.1090/noti1105>
- Bailey, D. H., & López de Prado, M. (2014). The deflated Sharpe ratio: Correcting for selection bias, backtest overfitting, and non-normality. *The Journal of Portfolio Management*, 40(5), 94–107. <https://doi.org/10.3905/jpm.2014.40.5.094>
- Bowers, M., Olausson, T. X., Wong, L., Grand, G., Tenenbaum, J. B., Ellis, K., & Solar-Lezama, A. (2023). Top-down synthesis for library learning. *Proceedings of the ACM on Programming Languages*, 7(POPL), 1182–1213. <https://doi.org/10.1145/3571234>
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 1877–1901. <https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfc4967418bfb8ac142f64a-Abstract.html>
- Carver, R. (2015). *Systematic trading: A unique new method for designing trading and investing systems*. Harriman House.
- Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. de O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H.,

- Sastry, G., Mishkin, P., Chan, B., Gray, S., ... Zaremba, W. (2021). *Evaluating large language models trained on code* [Preprint]. arXiv.
<https://doi.org/10.48550/arXiv.2107.03374>
- de Bruijn, N. G. (1972). Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church–Rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5), 381–392.
[https://doi.org/10.1016/1385-7258\(72\)90034-0](https://doi.org/10.1016/1385-7258(72)90034-0)
- Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). QLoRA: Efficient finetuning of quantized LLMs [Preprint]. arXiv.
- Dong, Y., Ruan, C. F., Cai, Y., Lai, R., Xu, Z., Zhao, Y., & Chen, T. (2024). XGrammar: Flexible and efficient structured generation engine for large language models [Preprint]. arXiv.
- Ellis, K., Wong, C., Nye, M., Sablé-Meyer, M., Morales, L., Hewitt, L., Cary, L., Solar-Lezama, A., & Tenenbaum, J. B. (2021). DreamCoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (pp. 835–850). Association for Computing Machinery.
<https://doi.org/10.1145/3453483.3454080>
- Fowler, M. (2010). *Domain-specific languages*. Addison-Wesley Professional.
- Geng, S., Josifoski, M., Peyrard, M., & West, R. (2023). Grammar-constrained decoding for structured NLP tasks without finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.
- Grand, G., Wong, L., Bowers, M., Olausson, T. X., Liu, M., Tenenbaum, J. B., & Andreas, J. (2024). LILO: Learning interpretable libraries by compressing and documenting code.

International Conference on Learning Representations.

<https://openreview.net/forum?id=TqYbAWKMle>

Howard, S. R., Ramdas, A., McAuliffe, J., & Sekhon, J. (2021). Time-uniform, nonparametric, nonasymptotic confidence sequences. *The Annals of Statistics*, 49(2), 1055–1080.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021). LoRA: Low-rank adaptation of large language models [Preprint]. arXiv.

Hui, B., Yang, J., Cui, Z., Yang, J., Liu, D., Zhang, L., Liu, T., Zhang, J., Yu, B., Lu, K., Dang, K., Fan, Y., Zhang, Y., Yang, A., Men, R., Huang, F., Zheng, B., Miao, Y., Quan, S., ... Lin, J. (2024). *Qwen2.5-Coder technical report* [Preprint]. arXiv.
<https://doi.org/10.48550/arXiv.2409.12186>

Jain, N., Vaidyanath, S., Iyer, A., Natarajan, N., Parthasarathy, S., Rajamani, S. K., & Sharma, R. (2022). Jigsaw: Large language models meet program synthesis. In *Proceedings of the 44th International Conference on Software Engineering* (pp. 1219–1231). Association for Computing Machinery.
<https://doi.org/10.1145/3510003.3510203>

Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4), 316–344.

Park, K., Wang, J., Berg-Kirkpatrick, T., Polikarpova, N., & D'Antoni, L. (2024). Grammar-aligned decoding. *Advances in Neural Information Processing Systems*, 37.

Poesia, G., Polozov, O., Le, V., Tiwari, A., Soares, G., Meek, C., & Gulwani, S. (2022). Synchronesh: Reliable code generation from pre-trained language models. *International Conference on Learning Representations.*

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information*

Processing Systems, 30, 5998–6008.

<https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html>

Ville, J. (1939). Étude critique de la notion de collectif. Gauthier-Villars.

Wang, B., Wang, Z., Wang, X., Cao, Y., Saurous, R. A., & Kim, Y. (2023). Grammar prompting for domain-specific language generation with large language models. *Advances in Neural Information Processing Systems*, 36.

Wang, H., Xin, H., Zheng, C., Li, Z., Liu, Q., Huang, Q., Lu, J., Yin, Z., Yin, Z., Liu, J., et al. (2024). LEGO-Prover: Neural theorem proving with growing libraries. *International Conference on Learning Representations*.

Wang, Z., Fried, D., & Neubig, G. (2024). TroVE: Inducing verifiable and efficient toolboxes for solving programmatic tasks. In *Proceedings of the 41st International Conference on Machine Learning*.

Waudby-Smith, I., & Ramdas, A. (2024). Estimating means of bounded random variables by betting. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 86(1), 1–27.

Willard, B. T., & Louf, R. (2023). Efficient guided generation for large language models [Preprint]. arXiv.

Apéndices

Apéndice A. Glosario

Tabla 15. Glosario de términos.

Término	Definición
Abstracción	Función aprendida que resume un patrón del corpus y se expande a primitivas del núcleo.
AST	Árbol de sintaxis abstracta que representa la estructura del programa.
Backtest	Ejecución histórica o sintética de una estrategia para obtener posiciones y retornos.
Candidato	Salida generada por el modelo antes de completar todas las puertas de evaluación.
Causalidad	Propiedad por la cual la salida en t no depende de datos posteriores a t .
e-process	Proceso no negativo utilizado para acumular evidencia secuencial en el tramo holdout.
Frozen core	Conjunto versionado de archivos y semántica protegido mediante digests y pruebas.
Onset / warm-up	Primera barra en la que un programa posee información suficiente para producir una salida válida.
Pass@k	Probabilidad estimada de obtener al menos una solución correcta entre k muestras.
Superviviente	Programa que supera las puertas sintácticas, de posición, screen y e-process.
Token BPE	Unidad del vocabulario neuronal del modelo.
Token léxico	Unidad reconocida por el lexer del lenguaje.
True-OOS	Tramo temporal oculto a la selección y reservado para análisis final.

Fuente: elaboración propia.

Apéndice B. Acrónimos

Tabla 16. Acrónimos y abreviaturas.

Acrónimo	Significado
AI / IA	Artificial Intelligence / Inteligencia Artificial
AST	Abstract Syntax Tree
BPE	Byte Pair Encoding
CI	Continuous Integration
DSL	Domain-Specific Language
EOS	End of Sequence

Acrónimo	Significado
FNV	Fowler-Noll-Vo hash
GPU	Graphics Processing Unit
IC	Intervalo de confianza
IS	In-Sample
LLM	Large Language Model
OOS	Out-of-Sample
P&L	Profit and Loss
SHA	Secure Hash Algorithm
SFT	Supervised Fine-Tuning

Fuente: elaboración propia.

Apéndice C. Gramática y vocabulario

La gramática conceptual del prototipo puede resumirse como sigue. La implementación concreta agrega validación de aridades, posiciones especiales y clases de valores.

```

programa    ::= término
termino     ::= número | variable | (lam término) | (primitiva término*)
variable    ::= "$" dígitos
número      ::= dígitos | dígitos "." dígitos
primitiva   ::= una de las 38 palabras reservadas del núcleo
               | una abstracción registrada en la biblioteca

```

El vocabulario completo del núcleo se lista en la Tabla 5. La implementación histórica conserva nombres en inglés porque el modelo base se entrenó sobre código y esos lexemas poseen segmentaciones favorables. El nombre Pucará identifica el lenguaje y no exige renombrar las primitivas ni los artefactos de reproducibilidad.

La gramática anterior se completa con las restricciones estáticas que la implementación valida: cada primitiva posee una aridad fija verificada por el parser; los índices \$i deben quedar ligados por un lam envolvente; el round-trip parse-print-parse debe reproducir el texto byte a byte para que el programa sea admisible; y las 14 abstracciones de la biblioteca se expanden a primitivas del núcleo antes de la interpretación, con preservación exacta del warm-up verificada por pruebas diferenciales.

Apéndice D. Métricas y fórmulas

Para una tarea con n muestras y c éxitos, el estimador utilizado para $\text{pass}@k$ es:

$$\text{pass}@k = 1 - C(n - c, k) / C(n, k).$$

El Sharpe anualizado se calcula sobre retornos válidos del tramo IS. El e-process se evalúa sobre holdout y el gate exige un máximo de al menos 20. La economía extremo a extremo se calcula como tokens generados reconstruidos divididos por programas válidos o supervivientes. La compactación válida compara las medias BPE condicionadas a que exista un candidato válido.

Ecuaciones del e-process (sección 4.8): $\sigma^2_t = \text{EWMA}(r^2; \text{span } 32)$ sobre $r_0, \dots, r_{t-1}$, inicializada con r_0^2 ; $x_t = \text{clip}(r_t / \sigma_t; -5, 5) / 5$; $\lambda_t = \text{clip}(\mu_{t-1} / (\hat{v}_{t-1} + \mu_{t-1}^2); 0, 0.5)$, con $\lambda_t = 0$ para $t < 32$; $W_t = W_{t-1} \cdot (1 + \lambda_t \cdot x_t)$, $W_0 = 1$; $e_{\max} = \max_{s \leq t} W_s$. El gate exige $e_{\max} \geq 20 = 1/\alpha$ con $\alpha = 0.05$ (desigualdad de Ville), válido en cualquier momento bajo $H_0: E[x_t | F_{t-1}] \leq 0$.

Apéndice E. Artefactos y comandos de reproducción

Tabla 17. Inventario de artefactos y digests principales.

Artefacto	Digest o versión
Commit final de master	15ff05c07d281d703c7b472b4c3eee7600b60529
Commit de vitrina	ee84d663aef4b03fa40d481bbce89cafeb20d84d
raw_f05-arm-b.jsonl	SHA-256 77f88583398d88cd201a1f48fcfbdf0cb936921e8792a7158823f377afe6da8e
raw_f05-arm-c.jsonl	SHA-256 b6e8bf2811da583743ff32f4d9f79441c28b6f45873b59a3351a9cdbb28a4974
scored_f05-arm-b.jsonl	SHA-256 5ce751b5e5e013061a0e4ced5292c4376c9b4eb38c3fb3fdc451e8f415a60235
scored_f05-arm-c.jsonl	SHA-256 b973898a9d6c03649e85686aa9f71a18297a6b63cf498be0cd734c4a03560644
prepared-tokenizer.json	SHA-256 3828632a1614f2620e9f453ae92bbf029c994954bf9c22475f687a196dc5517f
f05_token_audit.json	SHA-256 6f122842a48ed7e880861bc8ba124e070f4c9a1378f86b0af70e025efb46d4ad
t20_equity.csv	SHA-256 62fc70ba7708c61a2eefbd73ebe8ef14e192b9b04e318495ca3f1644a4979a61

Artefacto	Digest o versión
t20_equity_stats.json	SHA-256 73edd0e030db3344e75a7985c93a6c20bad8fd698e8fad92f776de1cfb0b4b13
vitrina/t20_equity.png	SHA-256 14a376b4250ab3b6c31d200af251b438ad029e6c1c2ee6acfd603f5c1c2844e9

Fuente: manifiestos, auditoría de tokens y close-out del repositorio.

Comandos principales de verificación local:

```
cargo test --quiet -- --test-threads=1
cargo clippy --all-targets --all-features -- -D warnings
cargo fmt --check
python vitrina/extract_t20_survivors.py --validate
cargo run --release --bin t20_equity
python vitrina/plot_t20.py
```

Los archivos de generación F0.5 se analizan sin ejecutar nuevamente el modelo. La auditoría BPE utiliza prepared-tokenizer.json para C y el vocabulario base para B, verifica el round-trip de las 12.240 salidas y produce f05_token_audit.json.

Los seis comandos anteriores constituyen también el procedimiento reducido para la defensa: no requieren GPU ni conexión de red, verifican la suite completa, la integridad del núcleo congelado y el formato, y reconstruyen la tabla y la figura de la vitrina t20 desde los artefactos congelados. El paquete mínimo de reproducción entregado junto con esta versión documenta el inventario completo, los hashes y el orden de ejecución.