ChatGPT

Auditoría forense y hoja de ruta para adaptar SirioC hacia una fuerza tipo Reckless sin comprometer la originalidad

Resumen ejecutivo

Esta auditoría se basa en el árbol entregado de SirioC [1] (snapshot local más su repositorio público), el árbol entregado de Reckless [2] (más su repositorio y release oficial y), y la documentación oficial de Leela Chess Zero [3] / , , , , , , y la documentación/licencias oficiales de GNU. SirioC publica hoy un conjunto amplio de capacidades "de motor serio" —bitboards, alfa-beta/negamax, qsearch, TT, LMR, null move, aspiration windows, Syzygy, lazy SMP y una evaluación "NNUE" configurable—; Reckless, por su parte, declara resultados de fuerza de primer nivel, una red moderna basada en amenazas, soporte NUMA/MultiPV y desde v0.9.0 licencia AGPL-3.0. Lc0 documenta una arquitectura completamente distinta: red residual con cabezas de policy/value y búsqueda MCTS/PUCT. [4]

El dictamen principal es claro: **SirioC no alcanzará una fuerza "similar a Reckless" mediante ajustes marginales**. El cuello de botella dominante no es el tablero, ni la UCI, ni siquiera la TT: es la combinación de un evaluador neuronal demasiado simple, un pipeline de entrenamiento insuficiente y una capa de búsqueda todavía por debajo de las heurísticas modernas que Reckless ya explota. En el snapshot auditado, la "NNUE" de SirioC no es una NNUE competitiva, sino un modelo lineal de 12 features de conteo de piezas con `bias`, `scale` y 12 pesos; eso la deja varios escalones por debajo de una arquitectura bucketed/threat-based real. En paralelo, SirioC sí ofrece una base aprovechable: el proyecto compila en local, la suite `sirio_tests` pasa, la estructura de evaluación incremental existe y la TT persistente es una base respetable para evolucionar. Ese punto de partida hace viable una migración fuerte, pero exige **reemplazo de la red y ampliación de la búsqueda**, no simple tuning.

La recomendación estratégica es doble. Si el objetivo es **paridad práctica con Reckless dentro del paradigma CPU alfa-beta**, el camino correcto es **crear una NNUE propia de verdad** —o, como máximo, destilar conocimientos desde Lc0 a una NNUE propia— y elevar SirioC hacia una arquitectura más cercana a las familias Stockfish/Berserk/Reckless en histories, correction terms, reductions y selectividad. Si el objetivo fuera "usar redes Leela" literalmente, entonces SirioC dejaría de converger hacia Reckless y se acercaría más a un híbrido o a un re-diseño estilo lc0, porque las redes de Lc0 están pensadas para policy/value + MCTS/PUCT, no para incrustarse sin fricción en un evaluador NNUE CPU-friendly. [5]

Diagnóstico rápido

Área	Estado actual en SirioC auditado	Impacto en fuerza	Veredicto
Motor base, tablero, movegen, FEN/UCI	Bueno y utilizable	Medio	Conservar y refactorizar

Área	Estado actual en SirioC auditado	Impacto en fuerza	Veredicto
Búsqueda alfa-beta	Buena base, pero incompleta frente a Reckless	Muy alto	Expandir de forma prioritaria
Evaluación clásica	Rica y útil como fallback	Medio	Mantener como respaldo
“NNUE” actual	Insuficiente para competir con Reckless	Crítico	Reemplazo total
Dataset/pipeline de entrenamiento	Demasiado simple	Crítico	Rehacer
TT y persistencia	Mejor de lo que sugiere el volumen del proyecto	Medio	Mejorar, no rehacer desde cero
Tiempo y SMP	Funcional, pero menos sofisticado	Alto	Reescribir la política de tiempo y escalado
Originalidad/licencias	Riesgo alto si se porta código/weights sin disciplina	Crítico	Gobernanza estricta de procedencia

Auditoría forense del árbol de SirioC

La parte pública de SirioC ya documenta un diseño con bitboards, evaluación clásica más NNUE configurable, búsqueda iterativa con alfa-beta y utilidades UCI/Syzygy/lazy SMP. Esa descripción general coincide con lo observado en el snapshot auditado. [6]

Módulos núcleo y estructura del árbol

En el snapshot local, SirioC tiene un árbol razonablemente completo para un motor independiente: `src/board.cpp`, `src/movegen.cpp`, `src/search.cpp`, `src/evaluation.cpp`, `src/tt.cpp`, `src/time_manager.cpp`, `src/uci.cpp`, `src/nnue/*`, `src/syzygy.cpp`, `tests/*`, `bench/*` y `training/nnue/*`. Los tamaños también son reveladores: `search.cpp` y `evaluation.cpp` concentran una parte muy grande de la complejidad. Ese reparto es suficiente para evolucionar, pero ya muestra el primer problema de mantenimiento: **la lógica crítica está demasiado centralizada**. Para que Codex pueda iterar con seguridad, SirioC necesita una fase de modularización ligera antes de los cambios más agresivos.

Dictamen forense: el núcleo de representación y generación de movimientos no aparece como el principal lastre. No es el área donde está la diferencia estratégica contra Reckless. La deuda real está más arriba: búsqueda, evaluación, entrenamiento y telemetría de pruebas.

Búsqueda, ordenación y heurísticas

SirioC ya incorpora, dentro de `src/search.cpp`, una base de búsqueda bastante más madura que la de un motor académico mínimo. En la auditoría local aparecen: iterative deepening, aspiration windows, quiescence, null move pruning, futility pruning, LMR, killer moves, history heuristic, SEE

(`static_exchange_score`), `MovePicker`, TT move ordering y lazy SMP compartiendo estado de parada. Eso encaja con lo que SirioC declara públicamente en su README. [6]

El problema no es la ausencia total de heurísticas, sino su **techo competitivo**. Frente a Reckless, SirioC carece —al menos en el snapshot auditado— de varias capas que hoy separan a un motor "bueno" de un motor de élite:

- no se ve una familia completa de **continuation histories**, **noisy histories** y **correction histories** como las que Reckless sí usa en su búsqueda;
- no aparece una integración fuerte de **probcut**, **singular extensions**, **dynamic reductions** y ajuste de poda condicionado por TT/eval/instabilidad;
- la política de tiempo de SirioC es utilitaria, pero todavía no parece modelar con la finura de Reckless variables como estabilidad de PV, tendencia de eval y robustez del mejor movimiento;
- el `search.cpp` de SirioC ya hace demasiado, lo que elevará el riesgo de regresiones si se "inyecta Reckless por partes" sin cortar antes en submódulos.

En el release oficial v0.9.0, Reckless destaca cuatro saltos precisamente en los campos que más importan aquí: red NNUE moderna basada en amenazas, soporte NUMA, mejoras multihilo y MultiPV; además documenta ganancias Elo notables sobre su propia versión previa. [7]

Dictamen forense: SirioC ya es un motor con búsqueda seria, pero **todavía no es una plataforma de élite en selectividad**. La brecha con Reckless no está en "tener o no tener alfa-beta", sino en la calidad de las decisiones de poda, ordenación y corrección estadística alrededor de ese alfa-beta.

Evaluación clásica, redes y capa incremental

Aquí está el hallazgo central de la auditoría.

La evaluación clásica de SirioC es relativamente rica: tapering MG/EG, PST, movilidad, seguridad del rey, estructura de peones, bonus de pareja de alfiles y tratamiento especial de finales. Eso coincide con la documentación pública del proyecto. [6]

Pero la red actual de SirioC, en el snapshot local, **no es comparable** con una NNUE moderna. La evidencia del árbol entregado es contundente:

- `FeatureState` contiene únicamente **12 contadores** de piezas (6 tipos × 2 colores).
- `NetworkParameters` contiene solo `bias`, `scale` **y 12 pesos**.
- `SingleNetworkBackend::evaluate_state()` hace esencialmente un **producto punto lineal** sobre esos 12 contadores.
- El API reporta dimensiones del tipo `PieceCounts[2x6]`.
- El entrenamiento (`PieceCountModel`) replica exactamente esa misma estructura lineal.

En otras palabras: hoy SirioC llama "NNUE" a algo que, en términos competitivos, es **una regresión lineal sobre material normalizado**, no una red sparse-updatable al estilo moderno descrito por la documentación oficial de Stockfish. La documentación oficial de NNUE enfatiza sparsidad muy alta, acumuladores actualizables, varias capas lineales/cuánticas y diseño CPU-friendly de baja latencia; eso está muy lejos del backend actual de SirioC. [8]

En contraste, Reckless sí monta una red bucketed moderna. El release oficial menciona una "modern threat-based architecture" que extiende inputs bucketed tipo piece-square con features adicionales de amenazas. En el snapshot local, eso aparece implementado como una red con *input buckets*, *output*

buckets, acumuladores separados para PST y amenazas, y capas densas sucesivas después de la activación del feature transformer. [7]

Dictamen forense: la evaluación clásica de SirioC puede sobrevivir como fallback, pero la actual "NNUE" debe considerarse **descartada para fines competitivos**. No es una cuestión de reentrenar mejor los mismos 12 pesos; es una cuestión de **cambiar el modelo completo**.

Entrenamiento y datos

El pipeline de entrenamiento incluido en SirioC es uno de los aspectos más valiosos de la base de código porque ya crea un "hueco" donde insertar algo serio. Pero el pipeline actual es demasiado simple:

- `prepare_dataset.py` muestrea posiciones desde PGN;
- el target combina material y resultado (`material + result_weight * outcome`);
- `train.py` entrena un modelo lineal de 12 entradas;
- la configuración por defecto usa `epochs=50`, `batch_size=512`, `learning_rate=0.01` y `device=cpu`;
- `export_to_engine.py` genera un fichero de texto `SirioNNUE1`.

Eso es excelente como prototipo reproducible, pero está muy lejos de lo que usan proyectos NNUE competitivos. La documentación oficial de `nnue-pytorch` describe un flujo con features sparse reales, factorization, cuantización e infraestructura de entrenamiento mucho más seria. Reckless, además, acredita públicamente a `bullet` como trainer NNUE y a OpenBench como marco principal de pruebas. [9]

Dictamen forense: el pipeline de SirioC **sirve como cascarón**, no como solución. Hay que conservar su ergonomía y reemplazar su corazón.

Interfaces, I/O, tiempo de búsqueda y tablas de transposición

SirioC muestra un trabajo respetable en I/O y operación práctica. Su UCI expone opciones de uso real, incluyendo activación de NNUE, ficheros de red, hash persistente y Syzygy; además, el README documenta recomendaciones por control de tiempo y operación con GUIs. También declara lazy SMP configurable y persistencia de hash. [6]

En TT, el snapshot local de SirioC es más interesante de lo esperado: clústeres alineados, aging, profundidad empaquetada, secondary hash/probe secundario, persistencia a disco y prefetch. No es una TT "toy". El problema, otra vez, no está tanto en la estructura del contenedor como en cómo la búsqueda exprime esa información frente a motores más modernos.

En tiempo de búsqueda, SirioC conserva un modelo global de parámetros (`move_overhead`, `slow_mover`, `expected_moves`, `latency_estimate`, auto-tuning opcional). Es funcional, pero sigue por detrás del refinamiento mostrado por Reckless, cuyo time manager documenta soft/hard bounds y cuya búsqueda usa estabilidad de PV y tendencia de score para modular el gasto temporal. Reckless también subraya NUMA y mejoras SMP como parte del salto v0.9.0. [7]

Dictamen forense: UCI/TT de SirioC no requieren demolición. Requieren **integración más fuerte con una búsqueda superior**. El tiempo de búsqueda sí merece rediseño parcial.

Comparación técnica entre SirioC, Reckless y Leela

SirioC y Reckless viven en el mismo universo de diseño: motor UCI de CPU con búsqueda alfa-beta. Lc0 no: Lc0 es un proyecto neural-first con MCTS/PUCT, varias cabezas de salida y backends GPU/CPU; su documentación oficial insiste en esa separación entre “binary” y “weights file”. Además, si por “Leela” se entiende estrictamente , hay una confusión importante: ese proyecto es para **Go**, no para ajedrez. Para ajedrez, la referencia operativa es . [10]

Tabla comparativa

La tabla sintetiza rasgos públicos documentados de SirioC, Reckless y Lc0; los detalles internos exactos de tamaño de red en SirioC y Reckless se complementan con la inspección local de los snapshots entregados. [11]

Aspecto	SirioC auditado	Reckless auditado	Lc0 / Leela Chess Zero
Familia de búsqueda	Alfa-beta / negamax + qsearch	Alfa-beta/PVS con capa heurística mucho más profunda	MCTS/PUCT
Hardware objetivo	CPU	CPU	GPU preferente, CPU posible
Evaluación principal	Clásica + “NNUE” lineal de 12 features	NNUE moderna basada en amenazas y buckets	Red residual policy/value/moves-left
Inputs	Conteos por tipo/color	Piece-square bucketed + amenazas	Planos de tablero / representación neural
Red	Producto punto lineal	FT + capas ocultas bucketed	Torre residual SE + cabezas múltiples
Persistencia de red	Fichero texto `SirioNNUE1`	Modelo embebido/serializado en build	Pesos protobuf
Entrenamiento incluido	Sí, pero muy simple	Trainer externo acreditado (`bullet`)	Infraestructura oficial de training/self-play
Escalado SMP	Lazy SMP funcional	Mejoras SMP + NUMA documentadas	Paralelismo orientado a lotes NN
Licencia pública	MIT	AGPL-3.0 desde v0.9.0	GPL-3.0

Diagrama de arquitectura

```
flowchart LR
    A[SirioC actual] --> A1[Bitboards / Movegen]
    A --> A2[Alpha-beta + qsearch]
    A --> A3[Eval clásica]
    A --> A4["'NNUE' lineal 12 features"]
    A --> A5[TT persistente + lazy SMP]

    B[Objetivo tipo Reckless] --> B1[Bitboards / Movegen]
```

```
    B --> B2[Alpha-beta/PVS selectivo]
    B --> B3[Histories y correcciones]
    B --> B4[NNUE threat-based real]
    B --> B5[TT + NUMA + SMP fuerte]

    C[Lc0 / Leela Chess Zero] --> C1[Backend NN]
    C --> C2[Red residual policy/value]
    C --> C3[MCTS/PUCT]
    C --> C4[Batching GPU/CPU]
```

Conclusión comparativa

Si el norte es “fuerza similar a Reckless”, el camino técnicamente coherente es **seguir en CPU alfa-beta y sustituir la red de SirioC por una NNUE real**. Si el norte fuera “usar Leela”, entonces ya no estaríamos adaptando SirioC a Reckless, sino **redirigiendo SirioC hacia otra familia algorítmica**. Ese desvío es posible, pero no es el camino corto ni el más natural para este repositorio. [1][2]

Opciones para construir una red propia o adaptar redes Leela libres

Opción recomendada: NNUE propia para SirioC

La solución más sólida para fuerza + originalidad es entrenar una **NNUE propia**, con implementación propia, formato propio y total trazabilidad de datos. La arquitectura que mejor equilibra realismo y ambición para SirioC no es una copia literal del código de Reckless, sino una **arquitectura inspirada en la misma clase de problemas**:

- features **HalfKAv2 o piece-square bucketed** con dos perspectivas;
- side-channel de **amenazas**;
- *input buckets* por rey/fase;
- *output buckets* por ocupación/material;
- cuantización para inferencia CPU;
- acumuladores incrementales de verdad.

Una propuesta concreta para SirioC v2 sería:

Componente	Propuesta
Feature transformer	512 o 768 neuronas
Buckets de entrada	8–10
Buckets de salida	8
Hidden layers	32 → 32 o 16 → 32
Activación	clipped ReLU / cuantizada
Tipo de pesos	int16/int8 en inferencia
Fallback	evaluador clásico actual

Componente	Propuesta
Formato	binario `SirioNNUE2` cuantizado

La justificación es sencilla: la documentación oficial de Stockfish NNUE recalca que la fuerza práctica proviene de combinar sparsidad, actualización incremental y cuantización eficiente; y el release v0.9.0 de Reckless muestra que una arquitectura threat-based moderna sigue siendo una ruta competitiva en 2026. [13]

Dataset propuesto

Para maximizar originalidad defendible y fuerza útil, recomiendo este mix de datos:

Fracción	Fuente	Objetivo
50–60%	self-play de SirioC mejorado con libros/UHO variados	identidad propia
20–25%	posiciones etiquetadas con teacher fuerte	acelerar convergencia
10–15%	finales Syzygy y posiciones simplificadas	exactitud en endgames
10%	hard negatives: posiciones donde SirioC y teacher discrepan	aprender correcciones útiles

La etiqueta no debería ser solo una centipawn raw. Para una NNUE moderna conviene combinar:

- score estático/teacher,
- WDL o resultado,
- fase/material bucket,
- muestreo balanceado por apertura/medio juego/final.

Si la exigencia de originalidad es estricta, la recomendación es **no basar el net final únicamente en targets de Stockfish o Lc0**. Úselos, si acaso, como *teacher assist* documentado, pero deje que la masa principal del dataset salga de self-play propio y del pipeline reproducible de SirioC.

Pipeline de entrenamiento propuesto

La infraestructura oficial de `nnue-pytorch` y la documentación oficial de NNUE dejan claro que el salto de calidad viene con features sparse reales, data loaders rápidos, cuantización y validación continua; `bullet`, a su vez, está orientado precisamente a entrenamiento NNUE a gran escala. [14]

Un pipeline viable para SirioC sería:

```
flowchart TD
    P[PGN / self-play / FEN corpus] --> Q[generador SFEN + labels]
    Q --> R[split train/val/test]
    R --> S[trainer NNUE propio o bullet]
    S --> T[checkpoint cuantizado]
    T --> U[exportador SirioNNUE2]
```

```
    U --> V[fastchess / cutechess / ORDO]
    V --> W[promoción automática del mejor net]
```

Recursos de hardware y costes orientativos

Ruta	HW recomendado	Coste relativo	Recomendación
NNUE propia	1 GPU 16–24 GB + CPUs para self-play	Medio	**Sí**
Distilación Lc0 → NNUE	1 GPU + teacher Lc0/Stockfish	Medio	**Sí, como acelerador**
Lc0 desde cero	varias GPUs potentes o infraestructura distribuida	Muy alto	**No para este objetivo**
Leela Zero puro	irrelevante para ajedrez	Alto / desalineado	**No**

Este juicio no es ideológico sino económico y técnico. La documentación de Lc0 describe una infraestructura completa de engine + client + training + server para sostener su ciclo RL, y la línea AlphaZero/AlphaGo Zero nace en un contexto de cómputo masivo. Para llevar a SirioC hacia Reckless, eso es un desvío caro. [15]

Adaptar redes Leela libres

Aquí conviene separar tres casos.

Caso más útil: usar Lc0 como teacher, no como red embebida.
Lc0 documenta una red residual con policy/value/moves-left y backends específicos; SirioC no tiene esa interfaz. Lo más razonable es usar Lc0 para **etiquetar posiciones** o para crear señales auxiliares de política, y después destilar eso a una NNUE propia de SirioC. [16]

Caso posible pero costoso: híbrido de move ordering / root policy.
Podría añadirse a SirioC una política externa que reordene movimientos en raíz usando Lc0. Eso exige proceso externo, latencia, cache y GPU/CPU backend. Es útil como experimento, pero no es el camino principal hacia un motor “tipo Reckless”.

Caso no recomendado: importar pesos Leela y “convertirlos” directamente.
No hay una conversión natural de una red residual policy/value protobuf a un evaluador NNUE incremental de CPU. Técnicamente no es un “adaptador trivial”; es un rediseño del motor.

Dictamen: si quiere “usar redes Leela libres” sin sacrificar la identidad de SirioC, la ruta correcta es **distillation**, no “embed directo”.

Hoja de ruta paso a paso para Codex

Prioridades

Prioridad	Objetivo	Resultado esperado	Esfuerzo estimado
P0	Sustituir la falsa NNUE por una NNUE real	mayor salto de fuerza	2–4 semanas
P0	Ampliar la búsqueda con histories/ corrections/selectividad	segundo mayor salto	2–4 semanas
P1	Rediseñar tiempo y SMP	más fuerza práctica / escalado	1–2 semanas
P1	Endurecer validación y auto-promoción de nets	menos regresiones	1 semana
P2	Híbridos de policy / experimentos Lc0 teacher	valor exploratorio	1–3 semanas
P2	Limpieza legal y trazabilidad	cumplimiento y originalidad	continua

Tareas precisas por archivo y función

La siguiente lista es operativa y está pensada para ejecución por Codex sobre el árbol auditado.

Paquete de red y evaluación

Archivo	Función / bloque	Tarea
`include/sirio/nnue/backend.hpp`	`FeatureState`, `NetworkParameters`, `SingleNetworkBackend`	reemplazar conteos lineales por índices sparse, acumuladores y pesos cuantizados
`src/nnue/backend.cpp`	`compute_state`, `apply_move_to_state`, `evaluate_state`, `evaluate_batch`	implementar FT real + update incremental + hidden layers
`src/nnue/api.cpp`	`build_info`, `init`	exponer nuevo formato `SirioNNUE2`, tamaños, buckets y checksum
`src/evaluation.cpp`	`make_nnue_evaluation`, `ensure_thread_backend`, `initialize_evaluation`	hacer de la nueva NNUE la ruta principal y dejar clásica como fallback
`training/nnue/scripts/features.py`	`encode_piece_counts`	**eliminar**; sustituir por encoder HalfK/Threats

Archivo	Función / bloque	Tarea
`training/nnue/scripts/prepare_dataset.py`	pipeline completo	pasar de PGN→12 features a SFEN/FEN→features sparse + labels enriquecidas
`training/nnue/scripts/train.py`	`PieceCountModel`	sustituir por modelo NNUE real con cuantización-aware training
`training/nnue/scripts/export_to_engine.py`	`export()`	emitir binario cuantizado, no texto lineal
`training/nnue/configs/default.yaml`	todo	nueva configuración de lotes, optimizer, scheduler, mixed precision

Paquete de búsqueda

Archivo	Función / bloque	Tarea
`src/search.cpp`	`SearchContext`	sacar histories y parámetros de la estructura gigante actual
`src/search.cpp`	`MovePicker`	añadir noisy/continuation histories y score de quiets más moderno
`src/search.cpp`	`negamax`	añadir correction history, singular extensions, probcut, reverse futility, move count pruning
`src/search.cpp`	`quiescence`	SEE pruning más agresiva y TT-aware qsearch
`src/search.cpp`	`publish_best_result` / root loop	introducir estabilidad de PV y heurística de tiempo estilo Reckless
`include/sirio/search.hpp`	API pública	exponer controles más finos para configuración y test
Nuevo `include/sirio/history.hpp`	—	mover allí quiet/noisy/continuation/correction histories
Nuevo `src/history.cpp`	—	implementación y tests dedicados
Nuevo `include/sirio/search_params.hpp`	—	centralizar márgenes, factores y tuning knobs

Tiempo, TT, hilos e I/O

Archivo	Función / bloque	Tarea
`src/time_manager.cpp`	estado global	reemplazar por estructura por búsqueda con soft/hard bounds y multiplicadores por estabilidad
`include/sirio/transposition_table.hpp`, `src/tt.cpp`	`store`, `probe`	añadir TT PV flag explícito, score mate normalizado por ply y criterios de reemplazo más duros
`src/uci.cpp`	opciones	añadir `EvalBackend`, `NNUEFormat`, `SearchProfile`, `MultiPV`, `PolicyGuideFile` solo si se activa híbrido
`src/engine/work_queue_watchdog.cpp`	watchdog / colas	revisar para evitar overhead y mejorar escalado en análisis largo

Cronograma sugerido

```
timeline
    title Hoja de ruta SirioC -> fuerza tipo Reckless
    week 1 : congelar snapshot auditado
           : crear ramas P0/eval y P0/search
           : extraer histories y search params
    week 2 : implementar SirioNNUE2 backend
           : definir encoder sparse + exportador binario
    week 3 : preparar nuevo dataset y trainer
           : primer net funcional
    week 4 : integrar net en motor
           : match gauntlet contra baseline
    week 5 : correction/noisy/continuation histories
           : probcut + singular + LMR tuning
    week 6 : time manager nuevo
           : pruebas SMP/TT
    week 7 : promoción automática de nets
           : ORDO/SPRT y suites tácticas
    week 8 : limpieza legal
           : documentación de procedencia y release candidate
```

Comandos, scripts y snippets

Los comandos de SirioC ya existentes en el árbol local son útiles como punto de partida; las variantes marcadas como “propuestas” presuponen los cambios de arquitectura descritos arriba.

Build y tests actuales de SirioC

```
cmake -S . -B build -DCMAKE_BUILD_TYPE=Release
cmake --build build -j
./build/sirio_tests
./build/sirio_bench
```

Preparación y entrenamiento actual del pipeline incluido

```
python -m venv .venv
source .venv/bin/activate
pip install -r training/nnue/requirements.txt

python -m training.nnue.scripts.prepare_dataset \
  --pgn data/games.pgn \
  --output-dir training/nnue/datasets/processed \
  --name sirio_example \
  --sample-stride 2 \
  --result-weight 400

python -m training.nnue.scripts.train \
  --config training/nnue/configs/default.yaml

python -m training.nnue.scripts.export_to_engine \
  --checkpoint training/nnue/weights/sirio_example.pt \
  --output training/nnue/weights/sirio_example.nnue
```

Pipeline propuesto para SirioNNUE2

```
python -m tools.generate_sfen_corpus \
  --pgn data/*.pgn \
  --selfplay-dir runs/selfplay \
  --teacher stockfish \
  --out data/sfen/train/

python -m training.nnue.scripts.train \
  --config training/nnue/configs/sirio_nnue2.yaml

python -m training.nnue.scripts.export_to_engine \
  --checkpoint training/nnue/weights/sirio_nnue2.pt \
  --output nets/sirio_nnue2.bin
```

Gauntlet propuesto

```
fastchess \
  -engine cmd=./build/sirio name=SirioC-new option.Hash=256 option.Threads=8
\
  -engine cmd=./build/sirio name=SirioC-base option.Hash=256
option.Threads=8 \
```

```
  -each tc=10+0.1 proto=uci \
  -rounds 500 -repeat -recover \
  -openings file=books/UHO_XXL_+0.90_+1.19.epd format=epd order=random
```

Snippet de dirección para el nuevo encoder

```
struct SparseFeature {
    uint16_t idx;
    int16_t value;
};

struct FeatureState {
    std::array<SparseFeature, MAX_ACTIVE_FEATURES> white;
    std::array<SparseFeature, MAX_ACTIVE_FEATURES> black;
    uint16_t white_count = 0;
    uint16_t black_count = 0;
};
```

Pruebas, métricas de fuerza y métricas de originalidad

Cómo medir fuerza

Reckless destaca públicamente OpenBench como su marco principal de pruebas, y su release v0.9.0 publica ganancias Elo explícitas por suite. SirioC ya trae benchmarks reproducibles y un `match_orchestrator.py`. Eso sugiere una integración natural: mantener los benches y elevar el sistema de matches hasta un flujo ORDO/SPRT sostenido. [17]

La batería que recomiendo es:

Capa	Métrica	Criterio
Correctitud	perft, reglas de tablas, Syzygy	cero regresiones
Velocidad	NPS, nodos útiles, hashfull, SMP scaling	no perder velocidad neta por Elo marginal
Táctica	suites tipo STS/WAC/ECM	mejora estable
Práctica	fastchess/cutechess gauntlets	SPRT favorable
Rating	ORDO/BayesElo	CI al 95%
Robustez	UHO/DFRC/SMP	no sobreajustar a un único régimen

Cómo medir originalidad

“Originalidad” no es una sola métrica; requiere una **matriz de evidencia**.

Riesgo	Prueba recomendada	Evidencia a conservar
Copia de código	`git diff`, `jscpd`, AST-diff contra baselines públicos	reporte de similitud por archivo
Copia de pesos	correlación/coseno por capa, histogramas y fingerprints de activación	informe por net y hash SHA-256
Dependencia excesiva de teacher	correlación de outputs en 100k FEN	curva SirioC vs teacher
Falta de procedencia	manifiesto de dataset y scripts	`DATASET.md`, `MODEL_CARD.md`, checksums

Una política razonable de originalidad exigible sería:

- **todo dataset versionado y con hash**;
- **todos los checkpoints con metadata reproducible**;
- **ningún peso redistribuido sin licencia clara**;
- **ningún port de código sin relicenciar conscientemente**;
- **release notes con procedencia exacta**.

Hitos cuantitativos aconsejados

Hito	Señal mínima
Hito A	nueva NNUE > baseline clásica en 2+0.02
Hito B	nueva búsqueda > baseline NNUE en 10+0.1
Hito C	SMP 8 hilos no colapsa y gana Elo neto
Hito D	DFRC/UHO no muestran regresión severa
Hito E	informe de similitud de código y pesos archivado

Riesgos legales, éticos y recomendaciones de cumplimiento

SirioC publica licencia MIT en su repositorio. Reckless v0.9.0 declara que el proyecto pasa a AGPL-3.0. Lc0 y lc0.org documentan GPL-3.0. La FAQ oficial GNU recuerda que, cuando se combinan obras con código GPL/AGPL o se presta software AGPL modificado a usuarios a través de red, entran en juego obligaciones de reciprocidad y de oferta de código fuente correspondiente. [18]

Recomendaciones de cumplimiento

1. **No porte código de Reckless a SirioC esperando conservar una distribución "puramente MIT".** Si copia o combina código sustancial, trátelo como trabajo derivado y asuma el régimen copyleft que corresponda. [19]

2. **No incruste código de lc0 o de `nnue-pytorch` sin revisar la licencia del resultado global.** Ambos proyectos publican GPL-3.0; combinar código con SirioC cambia la naturaleza de la distribución. [20]

3. **Si usa redes “Leela libres”, verifique la licencia de cada artefacto concreto de pesos antes de redistribuirlo.** La forma segura es tratarlas como teachers externos o como fuentes de distilación, no como pesos empaquetados sin análisis.

4. **Separe inspiración de copia.** Arquitecturas, ideas y parámetros de alto nivel pueden inspirar; el código exacto y los pesos concretos exigen revisión de licencia y procedencia.

5. **Documente la genealogía del net.** Un `MODEL_CARD.md` por release con: datasets, teachers usados, scripts, commit, checksum y licencia.

Recomendación final de cumplimiento

La vía más limpia es esta:

- SirioC sigue siendo **MIT** en sus partes originales.
- Todo código nuevo del backend NNUE, histories y trainer se escribe **desde cero** en SirioC.
- Lc0 o Stockfish pueden usarse como **herramientas externas** para etiquetado/distilación, con documentación explícita.
- Si en algún momento decide portar código directo desde Reckless o lc0, conviértalo en una decisión deliberada de licencia, no en una mezcla accidental.

Preguntas abiertas y limitaciones

Hay tres límites que conviene dejar por escrito.

Primero, esta auditoría sí inspeccionó localmente los snapshots entregados de SirioC y Reckless, pero **no pudo respaldar con fuentes web línea por línea** cada detalle interno de esos árboles; varios hallazgos finos —sobre todo los exactos del backend actual de SirioC y los tamaños concretos de la red de Reckless— provienen de esa inspección local directa del código suministrado.

Segundo, **no se ha medido aquí Elo absoluto de SirioC contra Reckless** mediante un match largo controlado; por eso el informe emite un dictamen arquitectónico, no un número Elo cerrado.

Tercero, en licencias de pesos y distilación, este informe ofrece una **lectura técnica de cumplimiento**, no asesoría jurídica formal. Si SirioC va a redistribuir binarios, pesos o servicios comerciales, conviene una revisión legal específica sobre el pipeline final.

---

[1] [9] [14] https://github.com/official-stockfish/nnue-pytorch
https://github.com/official-stockfish/nnue-pytorch

[2] [3] [4] [6] [11] [18] https://github.com/jordiqui/SirioC
https://github.com/jordiqui/SirioC

[5] [15] [16] https://lczero.org/dev/overview/
https://lczero.org/dev/overview/

[7] [12] [19] https://github.com/codedeliveryservice/Reckless/releases/tag/v0.9.0
https://github.com/codedeliveryservice/Reckless/releases/tag/v0.9.0

[8] [13] https://official-stockfish.github.io/docs/nnue-pytorch-wiki/docs/nnue.html
https://official-stockfish.github.io/docs/nnue-pytorch-wiki/docs/nnue.html

10 https://github.com/leela-zero/leela-zero
https://github.com/leela-zero/leela-zero

17 https://github.com/codedeliveryservice/Reckless
https://github.com/codedeliveryservice/Reckless

20 https://github.com/LeelaChessZero/lc0
https://github.com/LeelaChessZero/lc0