Table of Contents

Gold/Silver Ratio Perp — Contract Spec & Launch Runbook

Instrument: a perpetual future on the XAU/XAG ratio (spot gold ÷ spot silver) **Venue:** Hyperliquid, via HIP-3 (builder-deployed perpetuals) **Prepared for:** Daniel · May 2026

> **Disclaimer.** This is a technical build guide, not legal, tax, or investment advice. Listing a derivative on precious-metals prices has regulatory implications (commodity-derivatives, and potentially securities/benchmark rules) that vary by jurisdiction. Engage qualified counsel before any mainnet deployment. All parameters reflect Hyperliquid's documentation as of May 2026 and are subject to change by validator governance.

---

1. The key design decision: this is a perp, not a token

It is worth being precise about *what* you are listing, because it determines the cost and the controls.

A **HIP-1 spot token cannot track the gold/silver ratio.** A HIP-1 token is a capped-supply asset whose price is simply whatever its own order book clears at. There is no protocol mechanism that pegs a spot token to an external value like XAU ÷ XAG. To make a spot token "track" the ratio you would have to run a market maker that continuously re-quotes around the true ratio and absorb arbitrage every time you drift — a perpetual subsidy with no protocol support, and a soft peg that breaks under pressure.

The instrument you actually want behaves like a **perpetual future on the ratio**, and on Hyperliquid that is an **HIP-3 builder-deployed perpetual**. The reason it works is that the protocol does the tethering for you: you publish the ratio as the market's **oracle price** every ~3 seconds, and the **funding mechanism** continuously pulls the contract price toward that oracle. Traders who push the contract away from the ratio pay funding to those on the other side, which is exactly the economic force that keeps the contract honest. You define the contract; the protocol enforces solvency, margining, and liquidation.

This document specifies that contract and the steps to launch it, and compares the two ways to get it live — **self-deploy** (you stake and operate) versus **partner / HIP-3-as-a-Service** (a provider stakes and operates; you supply the spec and share fees).

---

2. The most economical implementation

The headline cost fact is counterintuitive and important: **the protocol charges essentially nothing to list the market itself.** The first three markets on any HIP-3 perp DEX skip the deploy auction entirely, so listing the gold/silver ratio market costs only trivial gas. There is no per-listing fee.

The real gate is the **deployer stake of 500,000 HYPE (~$25M at an assumed $50/HYPE)**. This is *locked, slashable capital — not a fee*; you get it back if you wind the market down cleanly, but it is exposed to HYPE's price and to slashing while live. For a single niche market, tying up ~$25M is rarely justified, which is why the economical route for most builders is to **list through an existing HIP-3 deployer or a HIP-3-as-a-Service provider** (for example, Pyth runs a managed service that combines the metals oracle, the staking/operations, and even capital support). You hand them the contract spec, they operate it, and you split the fees.

The companion spreadsheet (GSR_Cost_Revenue_Model.xlsx) computes both paths from your own inputs. With the default assumptions (HYPE at $50, $10M/day volume), the comparison is roughly:

Capital locked			
One-time cost			
Annual operating cost			
You keep			

The economic crossover is volume-driven: below roughly $25–50M/day, the partner path usually wins because you avoid the $25M lockup; above that, self-deploy's full 50% fee share starts to dominate. Run your own numbers in the model.

Recommendation: prototype on **testnet** with the provided scaffold (no stake required), then launch on mainnet via a **HIP-3-as-a-Service partner**, and only graduate to self-deploying your own DEX once volume justifies the capital lockup.

---

3. Contract specification

These are the parameters you (or your partner) set when registering the market. The Hyperliquid action is perpDeploy → registerAsset with a schema that initializes the DEX.

Parameter			
DEX name (dex)			
Asset ticker (coin)			
Full name			

Underlying
Collateral / quote
szDecimals
Initial oracle price
Margin table / max leverage
Margin mode
Open-interest cap
Funding multiplier
Funding interest rate
Fee scale (SetFeeScale)
Growth mode

A note specific to metals: precious metals are not 24/7 spot markets, so Hyperliquid applies HIP-3 trading constraints for non-24/7 asset classes to control risk and price deviation when the underlying spot market is closed. Your oracle/operations must account for weekend and holiday gaps in the underlying (the ratio of two metals is more stable than either leg, which works in your favor, but the gap risk is real). Discuss the exact constraints with your partner or read the current HIP-3 docs before going live.

---

4. The controls: how the contract stays pinned to the ratio

This is the heart of your operational responsibility, and the thing that makes the instrument "correct."

The oracle. As the deployer (or oracle updater) you publish prices via the setOracle action. For each tick you submit three things for the GSR asset: the **oracle price** (your computed XAU/XAG ratio), one or more optional **mark-price inputs**, and an **external perp price** (required for every asset, used to bound deviations). The protocol then sets the market's mark price as the median of your submitted marks plus the *local* mark (the median of best bid, best ask, and last trade). The reference scaffold publishes the same ratio into all three fields, so when the book is thin the mark tracks the ratio, and when the book is liquid the local mark blends in real trading.

Cadence. Publish about every 3 seconds (minimum 2.5 seconds between calls). If you stop, stale marks fall back to the local mark after 10 seconds — but you should never rely on that, because **oracle downtime is a slashing condition** for a self-deployer. Run the publisher redundantly with monitoring, alerting, and a failover updater key.

Protocol safety clamps (enforced on-chain). You cannot move the market arbitrarily even if you wanted to: oracle prices are clamped to 10x the start-of-day value, mark-price moves are clamped to 1% per update, and the external perp price bounds sudden

deviations. These protect users and, by extension, protect you from accidentally triggering a slashable event.

The data source. For a continuous 24/7 perp you need a continuous feed for each leg. The cleanest source is **Pyth's XAU/USD and XAG/USD price feeds** (institutional-grade, 120+ publishers), from which you compute the ratio. Chainlink also publishes both. If you later want to anchor to an official benchmark (e.g., the LBMA gold and silver fixings), note those are *once-or-twice-daily* reference prints, suitable as a settlement/anchor reference but not as the continuous feed a perp needs — so you would use the continuous feed for the live oracle and the official fix only as a periodic sanity check or settlement anchor.

The economic tether. With the oracle published, funding does the work. If the contract trades above the ratio, longs pay shorts (and vice versa), capped at 4%/hour, computed against the oracle price. Your funding multiplier (0–10) tunes how hard that pull is; the zeroed interest component means there is no built-in drift, so the contract has no reason to sit persistently off the ratio.

The reference implementation of this control loop is in gsr_perp_deploy.py (the run_oracle function): it fetches XAU and XAG from Pyth Hermes, computes the ratio, rounds to five significant figures, and calls perp_deploy_set_oracle every ~3 seconds with staleness and fallback handling.

---

5. Launch runbook

Phase 0 — Rehearse on testnet (both paths)

1. Fund a **testnet** wallet and install the SDK: pip install hyperliquid-python-sdk eth-account requests.
2. Verify the data feed: python gsr_perp_deploy.py --check-feed prints the live XAU/XAG ratio.
3. Confirm the USDC collateral token index on testnet via the spotMeta info endpoint.
4. Register the market: python gsr_perp_deploy.py --register (this initializes the GSR DEX and registers the asset).
5. Start the oracle: python gsr_perp_deploy.py --run-oracle and watch it publish every ~3s.
6. Place test orders through the testnet UI, confirm funding behaves, and confirm liquidation/mark behavior. Iterate on szDecimals, leverage, and OI caps until the market feels right.

Phase 1a — Mainnet via partner (recommended first launch)

1. Engage a HIP-3-as-a-Service provider (e.g., Pyth). Share this spec.
2. Negotiate the fee split (their share of your 50% deployer fees), any setup/platform fee, oracle SLAs, and whether they provide market-making capital support.

3. They register the asset under their staked DEX and run the oracle to your spec; you review on testnet/mainnet.
4. You (or they) seed liquidity, then open trading. Proceed to Phase 2.

Phase 1b — Mainnet self-deploy (once volume justifies it)

1. Acquire and stake **500,000 HYPE** from the deployer address. Understand it is slashable and remains so for 30 days after halting plus the 7-day unstaking queue.
2. Separate keys: keep the high-value staking key cold; authorize a dedicated **oracle-updater / sub-deployer** key (hot) that can only setOracle.
3. Register the DEX + GSR asset (first 3 markets free).
4. Stand up redundant oracle infrastructure with monitoring, alerting, and failover; begin publishing.
5. Set the fee scale, fee recipient, OI caps, and (optionally) growth mode.
6. **Do not enable cross margin** at launch — leave it isolated until the oracle and liquidity are proven.

Phase 2 — Liquidity, distribution, and getting "listed"

On Hyperliquid, "listing on exchanges" is different from a traditional CEX listing. Your market lives on **one shared order book (HyperCore)** and is automatically tradable through **any frontend that chooses to surface HIP-3 markets** — there is no separate listing on each venue. Practically, getting traded means:

1. **Seed liquidity.** HLP will *not* market-make your book. Either run a market maker (the protocol still provides backstop liquidation and inherits HyperCore's solvency math) or have your partner provide it. A tight, continuous quote around the ratio is what makes the market usable.
2. **Get surfaced by frontends.** Reach out to the major frontends (Phantom, pvp.trade, and others) so they display the GSR market; note that frontends run their own checks before listing a market (see Hyperliquid's "Frontend checks" guidance).
3. **Run your own frontend and register a builder code.** This is both a distribution channel and a second revenue stream (below). Builder codes require ≥100 USDC of perps account value and let you charge up to 10 bps on routed perp flow.
4. **Get into the data layer.** Market-data providers index HIP-3 contracts; once volume is real, the market shows up in analytics and aggregators, which drives organic flow.

---

6. How you make money

There are three independent revenue streams; you can stack all three.

1. **Deployer fee share.** You keep up to **50%** of trading fees on the GSR market (at fee scale 1.0, where users pay 2x the base fee and the protocol keeps the other half). This is the primary stream and it scales directly with volume.

2. **Builder-code fees.** If you run the frontend that routes the flow, you additionally earn up to **10 bps** on the perp volume you originate — on top of the deployer share.
3. **Market-making PnL.** Whoever provides liquidity earns the spread plus maker rebates. If that is you (or a JV with your partner), it is a third stream — though it carries inventory risk.

Using the model's default assumptions ($10M/day volume, fee scale 1.0, 30% of flow through your own frontend at 5 bps):

Stream
Deployer fee share
Builder-code (own frontend)
Gross (self-deploy)
Gross (partner, net of 30% share)

Net of costs, the self-deploy path nets ~$0.62M in year 1 (dragged down by the ~$1M opportunity cost of the locked $25M), while the partner path nets ~$0.96M — which is why, at this volume, partnering is more economical. The partner path breaks even at roughly **$213k/day** of volume. All of these figures are driven by the blue input cells in the spreadsheet; change the volume, fee scale, or HYPE price and everything updates.

---

7. Risk and regulatory checklist

- **Slashing (self-deploy only).** Oracle downtime, manipulation, or protocol-harming inputs can cost up to 100% of the 500k-HYPE stake by validator vote; slashed stake is burned, not refunded. Treat oracle uptime and key security as production-critical.
- **Oracle integrity.** Your market is only as good as the XAU/XAG feed. Use redundant institutional feeds, validate inputs, and alert on divergence between sources. A bad feed is both a user-harm and a slashing risk.
- **Metals are not 24/7.** Plan for weekend/holiday gaps in the underlying spot markets; respect Hyperliquid's HIP-3 constraints for non-24/7 assets.
- **Cross-margin is irreversible.** Launch isolated; only enable cross after liquidity and oracle quality are proven, knowing >50% daily moves invite a manipulation review.
- **Capital and market risk.** The stake is HYPE-denominated (price risk); market-making capital is at risk to inventory; both have an opportunity cost.
- **Regulatory.** A perpetual on metals prices is a commodity derivative in most regimes (e.g., CFTC oversight in the US; MiFID II in the EU), with possible benchmark-licensing, AML/KYC, sanctions, and market-surveillance obligations. Confirm whether referencing third-party metal indices requires a license. **Get qualified counsel in every jurisdiction you intend to serve before mainnet.**

---

8. Appendix — using the code scaffold

gsr_perp_deploy.py (testnet) exposes three commands:

pip install hyperliquid-python-sdk eth-account requests

```
python gsr_perp_deploy.py --check-feed     # print the live XAU/XAG ratio
python gsr_perp_deploy.py --register       # initialize the GSR DEX + asset (run once)
python gsr_perp_deploy.py --run-oracle     # publish the ratio every ~3 seconds
```

All parameters live in the Config dataclass at the top (network, DEX/coin names, szDecimals, margin table, collateral token, Pyth feed IDs, publish interval). It defaults to **testnet** and a placeholder key — set a real testnet key via --secret-key or an environment variable. The Pyth feed IDs are included as defaults but should be re-verified against Pyth's price-feed-ID list before use.