MORE ECOSYSTEM · MORETOKENS.COM

$FUEL

A mint-to-earn protocol where participation is the product,
promotion is the incentive, and every fee burns — forever.

Whitepaper v1.1 · Ethereum & Robinhood Chain · protocol live on testnet and mainnet

Abstract

$FUEL is a fair-launch mint-to-earn token whose reward function makes every participant financially long the protocol's own growth. Tokens are created only by public minting; each minter's payout scales with the logarithm of how many mints begin after theirs, converting every holder into a self-interested promoter without referral codes, trust, or off-chain accounting. Every mint action — opening AND claiming — pays an ETH protocol fee approximately equal to its own gas cost, and the built-in Claim & Remint cycle makes paying it a self-renewing habit. Fees route immutably: 45% into a 1,000-day one-way accumulator (the Pump Fund); 25% into an immediate perpetual $FUEL burn; 30% into perpetual $MORE burns. All execution is permissionless and keeper-incentivized. This paper specifies the mechanisms, quantifies the incentive structure, and states assumptions and risks plainly.

1 · Design Thesis

Most token launches purchase attention: marketing budgets, KOL allocations, liquidity incentives. $FUEL inverts this. The protocol pays for attention with math instead of treasury: the minting reward function is deliberately shaped so that the single most profitable action any existing participant can take — beyond minting again — is causing more people to mint. Growth expenditure is thereby crowdsourced to the entire holder base, and the same activity that grows the network simultaneously funds the burn engines that concentrate its value.

Three properties anchor the design:

- **Fair creation.** No presale, no team allocation, no admin mint. The public mint function is the sole source of supply.
- **Immutable routing.** Fee destinations and splits are welded at deployment; no key can redirect them.
- **Permissionless operation.** Every recurring protocol action is a public function with a caller reward, so the system runs without any operator.

2 · Minting Mechanics

A participant opens a mint by committing to a term of t days (7–100 at launch; the ceiling grows with adoption toward 1,000). The protocol records their position in the global mint counter — mint number m of all mints ever opened. At maturity, the participant claims and receives:

$$R = \log_2(\Delta m) \times AMP \times t \times EAA$$

where Δm is the number of mints opened after theirs during the term (floored at 2), AMP is a reward amplifier that starts at 3,000 and declines by 1 per day toward a floor of 1, and EAA is an early-adopter multiplier starting at +10% and decaying as global mints accumulate. Claims must occur within 7 days of maturity; lateness is penalized on an escalating curve to 99%.

The adoption-gated term ceiling:

```
maxTerm(m) = min( 100 + 15 × log2(m), 1000 ) days, for m > 5,000
```

The 100-day launch ceiling is not fixed: once the global mint counter m passes 5,000, the maximum term grows with the logarithm of adoption — roughly 349 days at 100k global mints, 449 at 10M, 549 at 1B — capped at 1,000. Commitment capacity is therefore itself a network good: the community's growth unlocks longer terms, and longer terms (linear in R) are the highest-leverage positions on further growth.

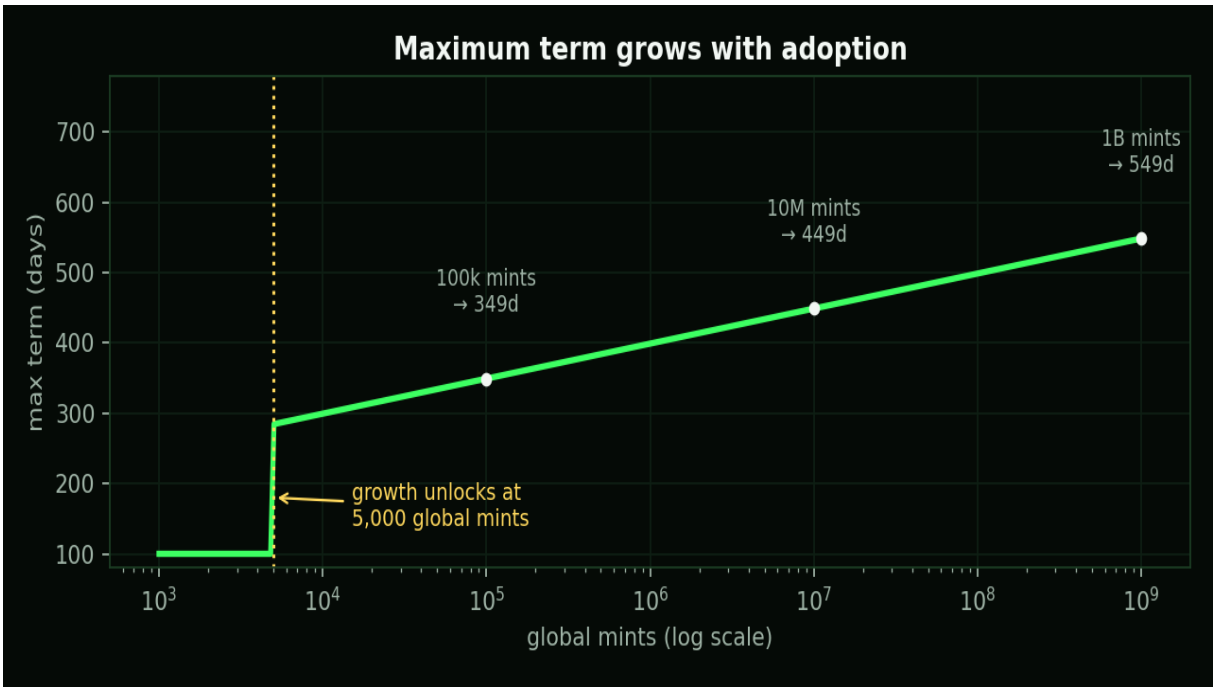

Fig. — The term ceiling as a function of global mints (log x-axis). Growth unlocks at 5,000 mints and compounds slowly: each doubling of adoption adds 15 days of maximum commitment.

The late-claim penalty

Maturity is not a deadline to claim by — it is the start of a 7-day claim window. Claiming on the day maturity hits costs nothing extra; waiting longer inside the window costs a small and rapidly escalating amount, and claiming after the window closes caps the penalty at its ceiling:

```
penalty(daysLate) = min( 2^(daysLate+3) / 7 − 1 , 99% )
```

The curve roughly doubles every day: 0% the first day, then 1%, 3%, 8%, 17%, 35%, 72%, capping at 99% from day 7 onward. There is no way to lose the position entirely — some reward is always claimable — but the design makes procrastination expensive fast, which keeps matured positions from sitting idle and gives the Claim & Remint flow (Section 5) a real behavioral nudge behind it, not just a convenience.

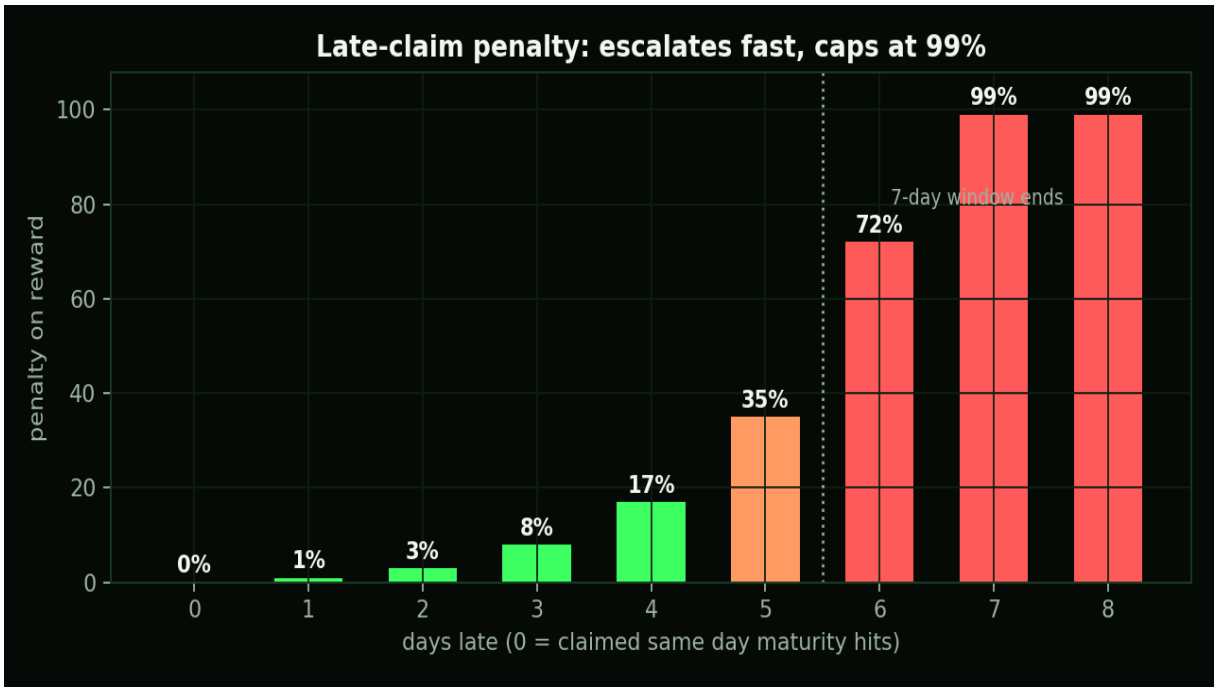

Fig. — The exact on-chain penalty curve. Claiming inside the first day is free; the cost of waiting compounds quickly enough that most rational holders claim well before the window closes.

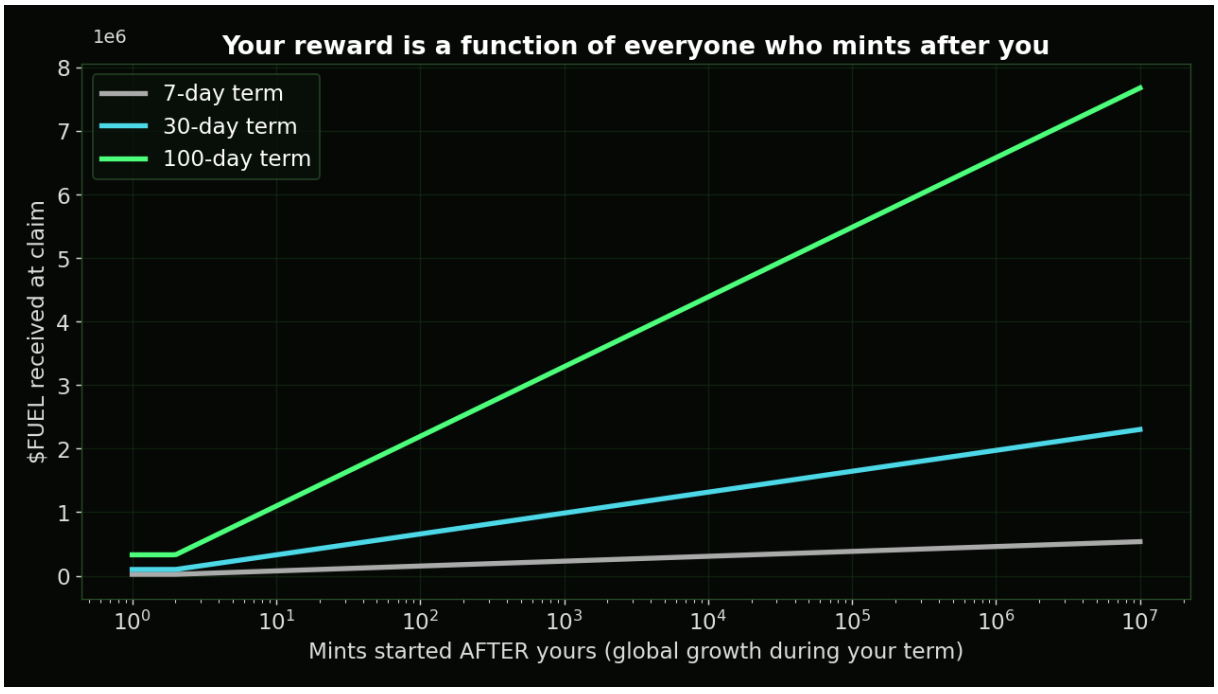

Fig. 1 — Claim payout vs. mints opened after yours, per the on-chain formula (AMP=3,000, EAA=+10%). The x-axis is the protocol's growth during your term: your reward is a direct, increasing function of adoption.

Two structural consequences follow. First, the logarithm rewards breadth over timing games: payouts grow smoothly with adoption rather than cliff-jumping, so there is no single moment to snipe. Second, term length is a conviction lever: reward scales linearly in t, so a 100-day commitment earns roughly 14× the per-mint-growth payout of a 7-day position — but exposes the minter to 100 days of adoption, which

is precisely the protocol recruiting long-horizon promoters.

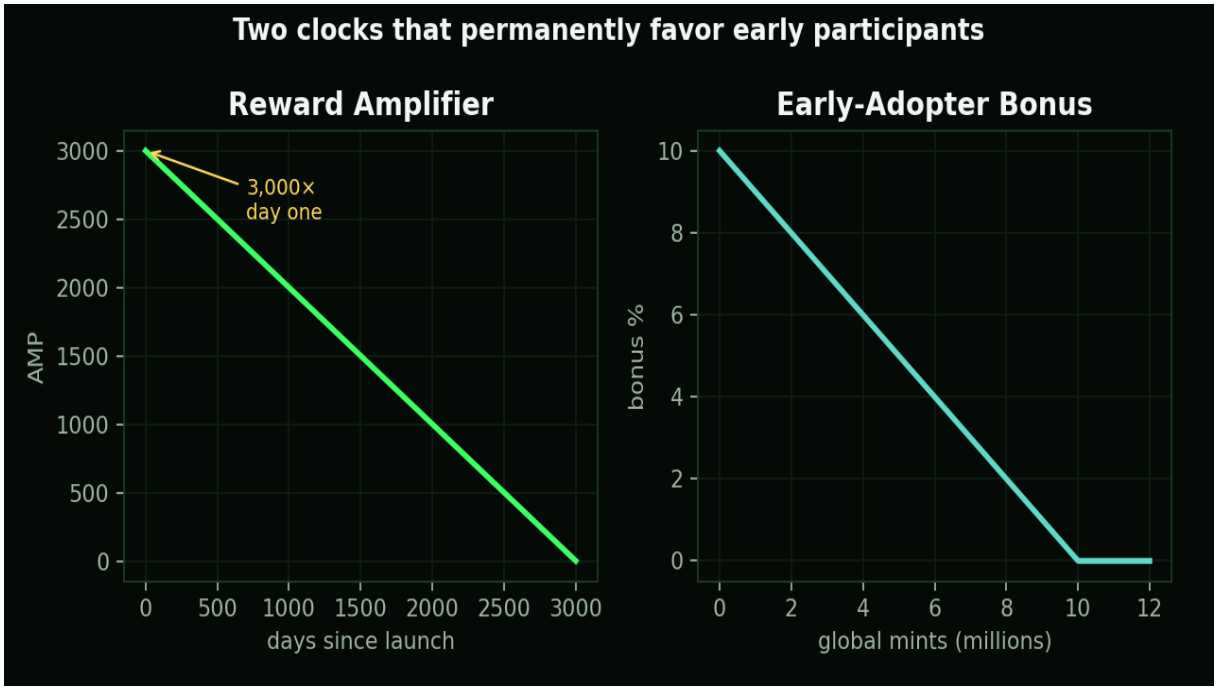

Fig. 2 — The two early-advantage clocks. AMP decays with calendar time; the Early-Adopter bonus decays with global mint count. Both are permanent: a mint opened today locks today's multipliers forever.

3 · The Marketing Engine

Consider a participant holding an open mint. Their claim value is R = log2(Δm)·AMP·t·EAA, where every variable is fixed at open except Δm — global adoption during their term. The marginal value of one additional recruited minter is ∂R/∂Δm = AMP·t·EAA / (Δm·ln2) > 0, always. Every open mint is therefore a live financial position on protocol growth, and the holder's rational behavior is promotion.

This differs from referral programs in three load-bearing ways: it is trustless (no codes, no attribution disputes — the global counter is the only ledger), uncapped (a recruit's recruit still increments Δm for you), and self-funding (recruited mints pay fees that burn supply, so promotion simultaneously raises your token count AND the scarcity behind each token). The flywheel closes on itself:

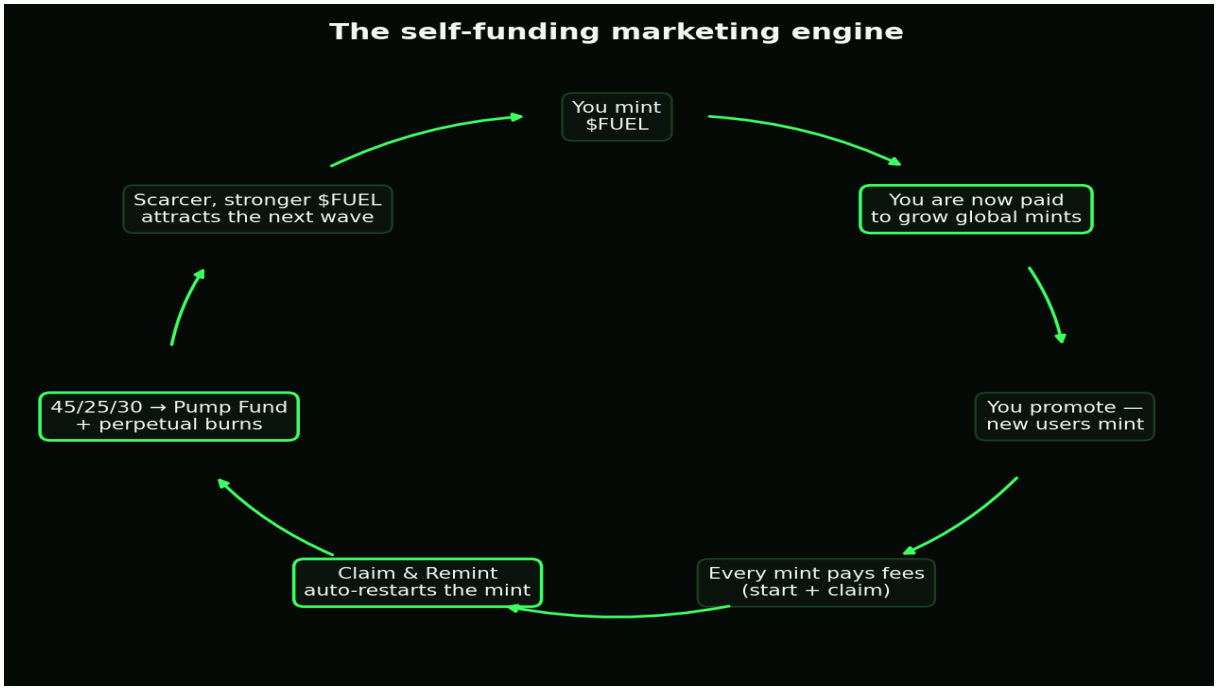

Fig. 3 — The self-funding growth loop. Note the highlighted node: Claim & Remint automatically reopens each mint at claim, so the fee-paying step recurs by default rather than by decision. Promotion is individually rational at every node.

A quantified example. A participant opens 50 mints at a 30-day term on day one (AMP=3,000, EAA=+10%). If the protocol averages 1,000 new mints/day, Δm ≈ 30,000 and each mint claims ≈ 1.47M $FUEL. If their promotion helps push the pace to 3,000/day, Δm ≈ 90,000 and each claims ≈ 1.63M — an 11% raise on their entire position for growing the network. The gradient never inverts: more adoption is always personally profitable.

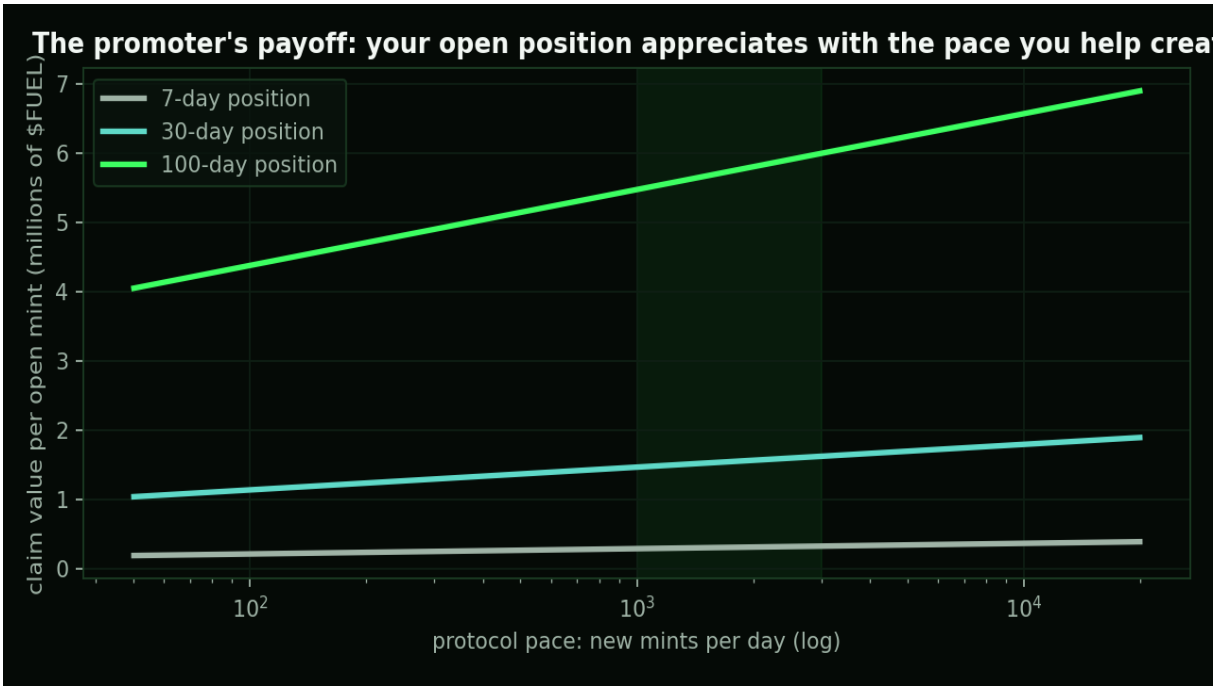

Fig. — The promoter's payoff curve: claim value per open mint as a function of the protocol's daily mint pace, for three term lengths. The shaded band is the worked example. Because holders choose the x-axis with their own promotion, every open position is a self-interested growth campaign.

4 · The Fee Engine

Every mint action pays an ETH protocol fee — once at open, once at claim:

```
fee = soloMintGas × max(tx.gasprice, minGasPrice)
```

soloMintGas benchmarks the gas of one unbatched mint, so the fee ≈ the gas the participant already paid: participation costs roughly 2× gas, with the second half captured by the protocol. The fee floats with network conditions and is floored so it cannot be gamed toward zero. Parameters are owner-tunable only inside hard-coded bounds (benchmark 50k–500k gas; floor ≤ 100 gwei) — calibration is possible, weaponization and zeroing are not.

The batcher penalty

The fee is per mint, per action, with no batch or remint discount. Batching 100 mints amortizes real gas by an order of magnitude; it amortizes fees not at all. Industrial minters — who capture disproportionate reward through scale — thereby fund the burn engines disproportionately.

5 · The Perpetual Cycle: Batch Minting & Claim-and-Remint

The BatchMinter contract lets one wallet operate up to 100 mints per transaction, each held at its own deterministic proxy address that is reusable forever. Its centerpiece is Claim & Remint: a single transaction that claims a matured batch — delivering the $FUEL to the wallet — and in the same breath reopens every position for a fresh term. One click; the claim fee and the new open fee are both paid; the participant's exposure to future adoption never lapses.

The rationality condition

```
recycle while: E[R] × price > 2 × fee + gas
```

A participant continues cycling as long as the expected claim value exceeds the round-trip cost. Because the fee is calibrated to ≈ one transaction's gas, the right side is small — on the order of a single swap — while the left side scales with AMP, term, and global adoption. For any position opened while the protocol grows, the inequality holds by a wide margin, and recycling is simply the profit-maximizing move.

The consequence is structural: rational self-interest produces a standing population of perpetually renewed mints, each paying two fees per cycle into the engines, indefinitely. The Pump Fund's 1,000-day accumulation is not fed by a launch spike that decays — it is fed by a habit that the reward function itself keeps renewing.

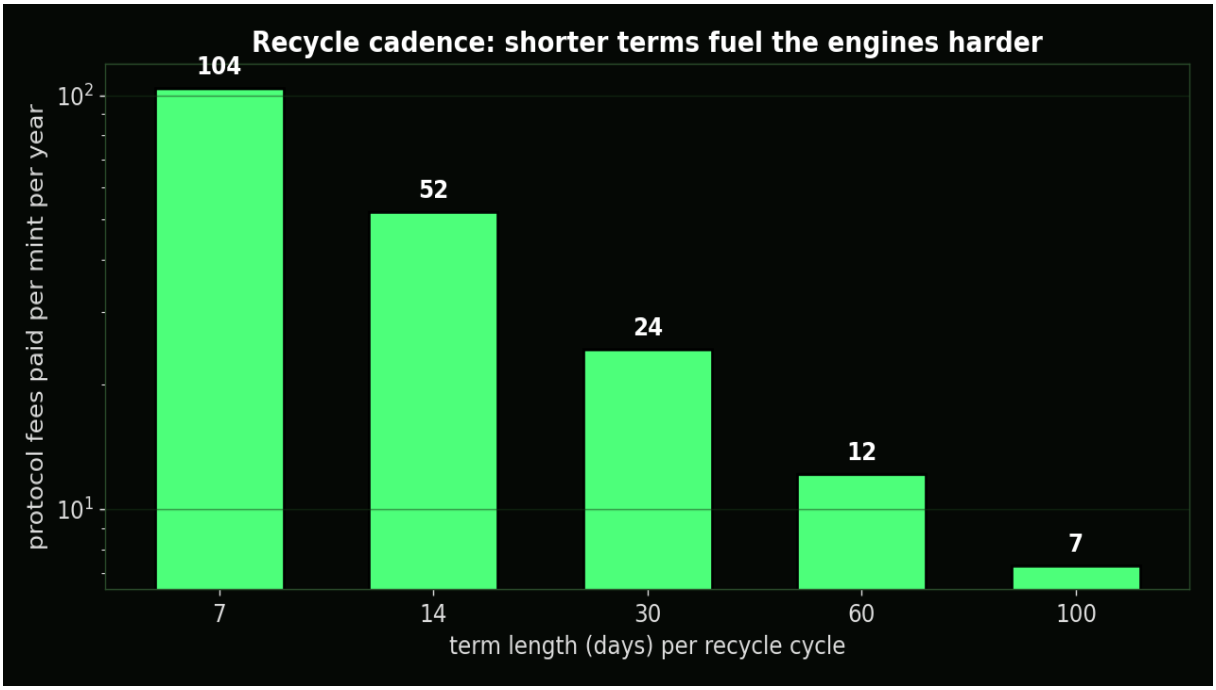

Fig. 4 — Fee throughput per perpetually recycled mint, by chosen term (log scale). A 7-day recycler contributes ~104 fees per mint per year; even a conservative 30-day recycler contributes ~24. Multiply by batch size and both fees per cycle: the engines' inflow is a function of standing participation, not launch hype.

Note the interplay with Section 3: each recycle also reopens a live position on future growth, refreshing the holder's incentive to promote. The marketing engine and the fee engine share the same crank.

6 · Flow of Funds: 45 / 25 / 30

45%	25%	30%
The 1000-Day Pump Fund	**$FUEL Buy & Burn**	**$MORE Burner**
One-way accumulator. Fills for 1,000 days; the entire balance then converts to $FUEL buy-pressure. Cycle restarts immediately, forever.	Enters the drip engine immediately; begins burning the same day.	Market-buys $MORE continuously; proceeds to 0xdEaD.

Fees pool in an ownerless FeeDistributor; anyone may trigger the split once 0.01 ETH accumulates.

7 · The 1000-Day Pump Fund

The Pump Fund is a one-way accumulator: for 1,000 days from the protocol's first mint, 45% of every fee enters and nothing exits — no owner path, no emergency valve. At cycle end, a permissionless sweep (caller reward: 0.5%) moves the entire balance into the burn drip engine, and the next cycle begins the same block.

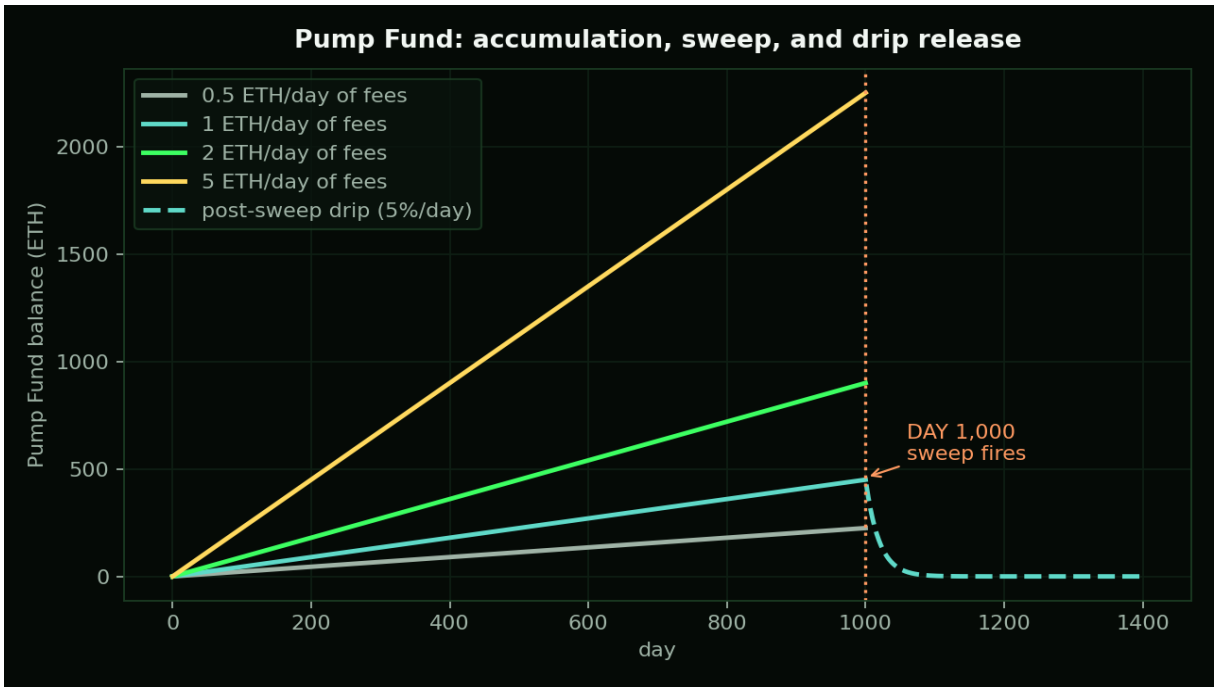

Fig. 5 — Accumulation under several fee-inflow scenarios, and the post-sweep release: swept ETH deploys through the drip engine (dashed: 5%/day), converting day 1,000 into a months-long buying campaign rather than a single absorbable spike.

The convergence thesis. The sweep arrives precisely when $FUEL is at maximum distribution: three years of fair minting, thousands of independent wallets, zero insider overhang. The fund's size is a pure mirror of cumulative usage — at a mean inflow of just 1 ETH/day of fees it exceeds 450 ETH; the date is public from block one and the market may price it accordingly.

This is design description with illustrative arithmetic — not a projection. Inflows depend entirely on adoption; price depends on far more than buy pressure.

8 · Burn Engines & Keeper Economics

Each burn engine drips its ETH pool at 1% per day (owner-tunable 1–10%), sliced into 144 ten-minute intervals. Any address may execute the current interval — swapping ETH for $FUEL through the protocol-owned pool (or for $MORE through its market) and burning the proceeds — earning 1.5% of the interval in ETH. Swaps are guarded by a 15-minute TWAP with bounded slippage; at drip scale, sandwich extraction is unprofitable. Missed intervals accumulate and execute together, so the schedule batches rather than falls behind.

The instant leg: steady-state burning

The 25% leg deserves its own analysis. It receives fee inflow continuously and drips out at a fixed percentage — a classic leaky-bucket system with a provable equilibrium: the pool grows until rate × pool = inflow, at which point daily burn equals daily inflow exactly. At 1 ETH/day of protocol fees and the default 1%/day drip, the pool converges toward 25 ETH and thereafter burns 0.25 ETH of $FUEL every day — 100% of its inflow — forever, while retaining a 25-ETH war chest that smooths burning through any lull in activity. The drip rate does not change how much ultimately burns (everything does); it only tunes the size of the buffer and the speed of convergence.

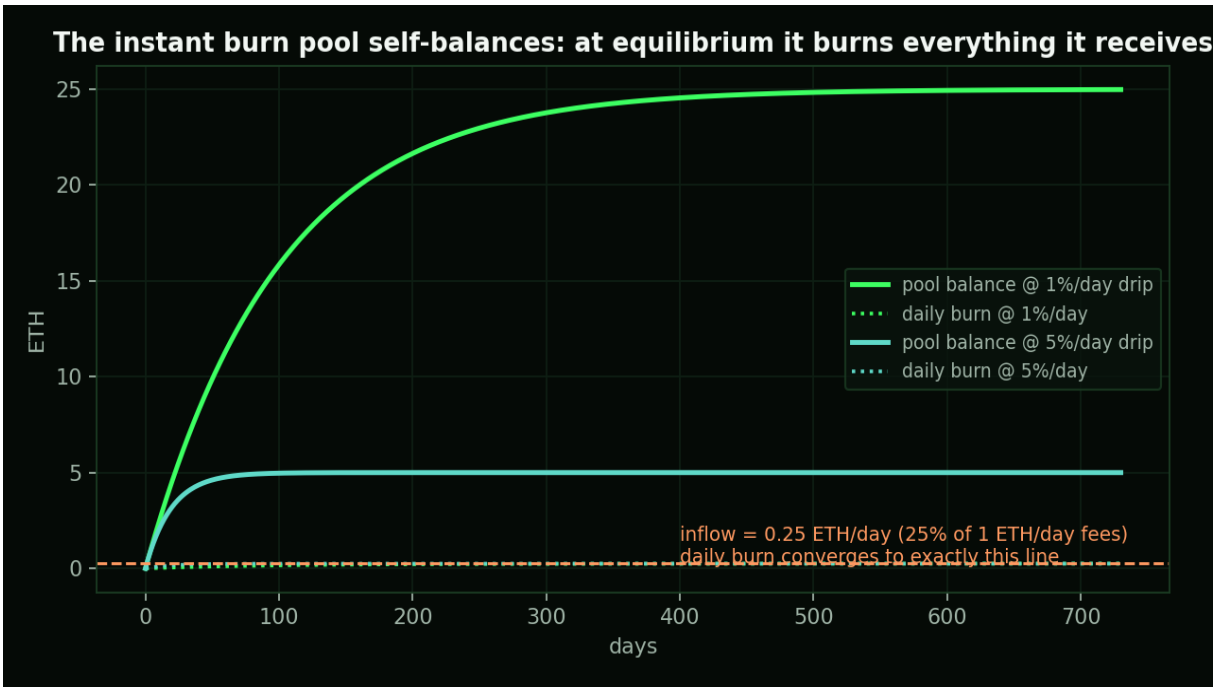

Fig. — The instant pool converging to equilibrium under constant inflow, at two drip settings. Daily burn (dotted) asymptotes to the inflow line in both cases: the leg burns everything it is fed, with the drip rate setting only the buffer size.

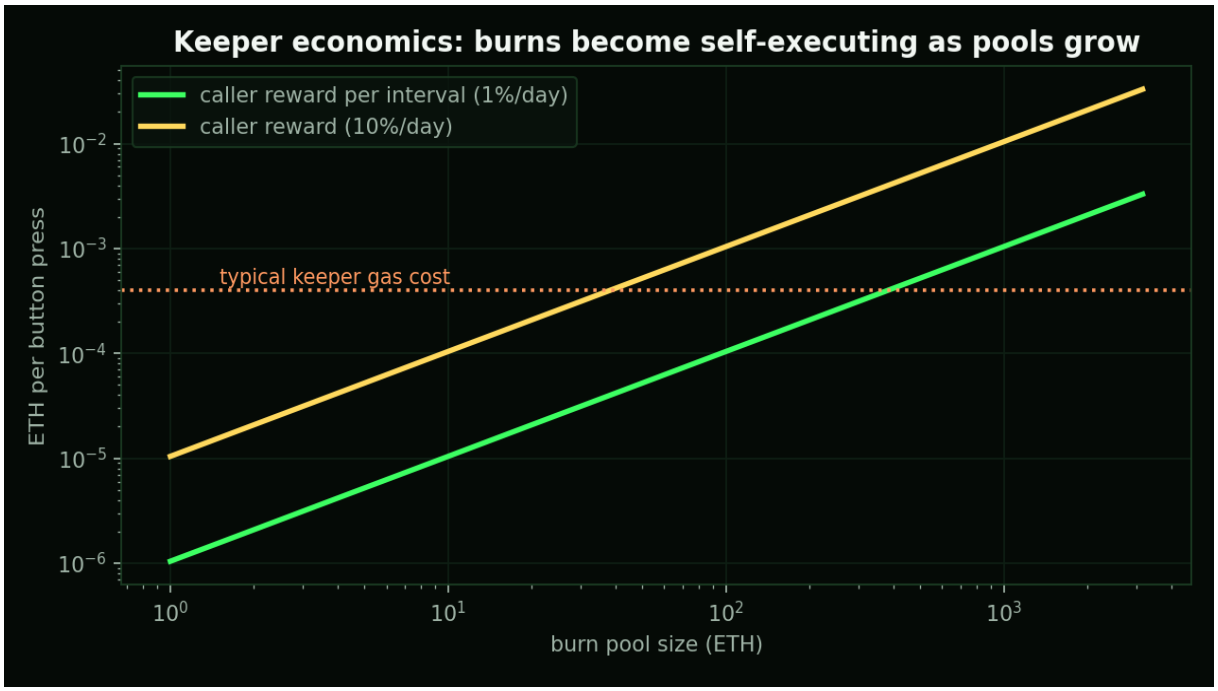

Fig. 6 — Caller reward per interval vs. pool size (log-log), against typical keeper gas. Below breakeven, intervals stack until a batch clears it; above, bots compete every 10 minutes. The system is self-executing at scale and self-batching below it.

Protocol-owned liquidity

The $FUEL buy-and-burn creates and permanently holds the $FUEL/WETH pool, seeded from early fee ETH against a one-time LP mint. LP fees earned in $FUEL are burned; the WETH side funds ecosystem operations. Pairing against WETH — not $MORE — is deliberate: $FUEL's supply is inflationary by design, and a $MORE pairing would transmit that inflation into $MORE sell pressure. WETH pairing quarantines the inflation, while the 30% leg hands $MORE pure, supply-blind buy pressure.

10 · Architecture & Trust Model

Contract	Role & trust surface
FUEL (token)	ERC-20; mint/stake/burn mechanics; collects the fee on every open and claim. (On-chain, the open function r
BatchMinter	Deterministic proxy factory; up to 100 mints/tx; full fee per proxy; Claim & Remint.
FeeDistributor	Ownerless 45/25/30 splitter; destinations immutable.
Pump Fund	Ownerless 1,000-day accumulator (on-chain: MintVault); exit only via sweep, only to the burn engine.
FuelBuyAndBurn	Drip engine + permanent protocol-owned $FUEL/WETH liquidity.
MoreBurner	Drip engine; $MORE to 0xdEaD.

Owner powers (bounded): fee benchmark and gas floor inside hard limits; drip rate 1–10%/day; swap slippage tolerance. **Owner non-powers:** redirecting fees, altering the split, withdrawing the Pump Fund, minting outside public rules, pausing, or rewiring — the one-time configuration validates all cross-references, then welds.

9 · Staking

Independent of minting, any $FUEL holder may stake their tokens directly with the Token contract for a fixed term and a protocol-wide APY. Staking is fee-free — no protocol fee is charged to open or close a stake, only ordinary network gas — and collateral-simple: staked $FUEL is burned on entry and re-minted (principal plus reward) on exit, so the position requires no token approval and cannot be rehypothecated elsewhere while locked.

Mechanics

A wallet holds one stake position at a time. Opening a new stake while one is active is not permitted; the existing position must be withdrawn first. Reward accrues only if the stake is withdrawn at or after maturity:

$$\text{reward} = \text{principal} \times (\text{APY} \times \text{term\_days}) / 365$$

where APY is fixed at the protocol-wide rate in effect the moment the stake opens, not the rate on the day it matures. There is no early-exit penalty in the punitive sense — withdrawing before maturity simply returns principal only, with zero reward, since the reward formula above evaluates to zero before the term completes. Capital is never at risk from staking itself; only the yield is conditional on patience.

A protocol-wide, decaying rate

Unlike the mint reward's per-position amplifier, APY is a single global value that decays on a fixed calendar, independent of any individual stake: 25% at launch, stepping down by 1 percentage point every 90 days, floored at 2%. Every stake opened at a given moment locks that moment's rate for its full term — so, as with minting, early stakers are structurally favored: the same 90-day commitment opened on

day one earns materially more than the identical commitment opened two years later.

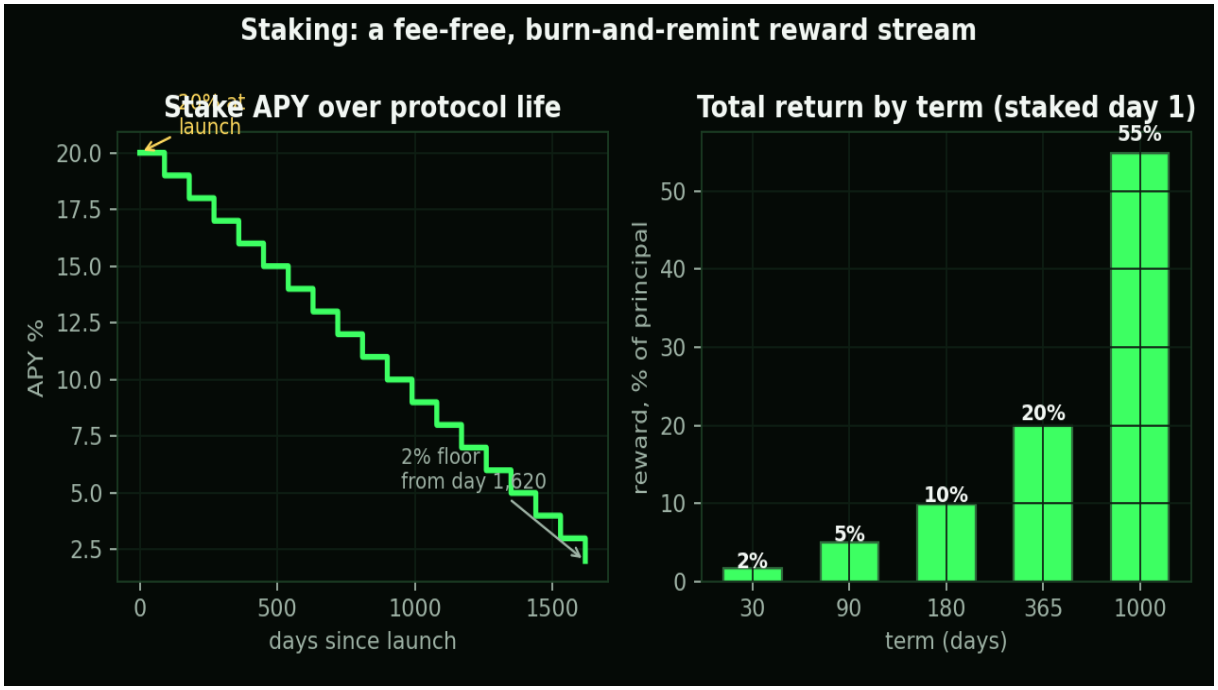

Fig. — Left: the APY step-down schedule (20% → 2% floor over ~4.4 years). Right: total return by term length for a stake opened on day one, at the launch APY — a 1000-day commitment returns roughly 55% of principal in $FUEL, paid only at maturity.

Relationship to the fee engine. Staking does not itself route ETH into the 45/25/30 split — it is a $FUEL-denominated yield mechanism, not a fee event. Its economic role is complementary: minting and Claim & Remint generate the ETH that fuels the burn engines, while staking gives holders a reason to remove freshly minted $FUEL from circulating supply and hold it through a lock, tempering the sell pressure that a purely inflationary mint schedule would otherwise create.

11 · Risk Disclosures

- **Unaudited code.** No third-party audit has been performed. Contract bugs can permanently destroy funds.
- **Unbounded supply.** $FUEL supply grows with every claim, indefinitely. Burn engines are a counterweight, not a promise of price behavior.
- **Reflexivity cuts both ways.** The same loop that amplifies growth amplifies contraction: falling adoption shrinks rewards, fees, and burns together — and the recycle inequality of §5 can flip, pausing fee flow until conditions recover.
- **Thin early liquidity.** The protocol-seeded pool starts small; early prices are trivially movable, and illustrative figures assume prices that large sales would themselves invalidate.
- **Long horizons.** The Pump Fund's first sweep is ~3 years out. Nothing guarantees any outcome on any timeline.
- **Testnet status.** Deployments exist on Sepolia, PulseChain, and Robinhood Chain testnets with accelerated cycles; mainnet parameters and addresses are not final everywhere.

Nothing herein is financial advice or an offer of securities. Participate only with capital you can lose entirely.

Appendix · Formula Reference

Quantity	Formula / value
Claim payout	`R = log2(max(Δm,2)) × AMP × t × EAA`
Reward amplifier	`AMP = max(3000 − days_since_launch, 1)`
Max term	`min(100 + 15×log2(m), 1000) days, once m > 5,000`
Early-adopter mult.	`EAA = 1 + max(0.10 − mints/1,000,000 × 0.01, 0)`
Protocol fee	`soloMintGas × max(tx.gasprice, minGasPrice); at open AND claim`
Recycle condition	`continue while E[R] × price > 2×fee + gas`
Fee split	`45% Pump Fund / 25% $FUEL burn / 30% $MORE burn`
Drip per interval	`pool × rate/day ÷ 144 (rate ∈ [1%,10%])`
Caller rewards	`1.5% per burn interval · 0.5% per Pump Fund sweep`
Late-claim penalty	`min(2^(daysLate+3)/7 - 1, 99%) — 0/1/3/8/17/35/72%, 99% from day 7`
Minimum mint term	`7 days`
Pump Fund cycle	`1,000 days (mainnet) · clock starts at first mint`
Stake reward	`principal × (APY × term_days) / 365, paid only at/after maturity`

Stake APY schedule	`20% at launch, −1pt per 90 days, floored at 2%`