Pickle Chain: a real-time Ethereum Layer 2 with native prices and shared application revenue

The Pickle Chain team
picklechain.xyz

Version 1.0, September 2026

Abstract

Pickle Chain is an Ethereum Layer 2 written from scratch in Rust around real-time execution. Transactions execute the moment they arrive; signed **mini-blocks** are sealed at a 10 ms scheduling target and serve as preconfirmations; standard Ethereum-shaped **EVM blocks** are sealed at a 250 ms scheduling target and are the confirmation wallets and explorers see; finality is Ethereum settlement. Every block's full transaction list is published so that any independent operator can re-derive the chain's state and check it. The protocol publishes prices itself: one system transaction per block carries 33 feeds, 19 of them real-world assets, with market hours respected, and two things are built on that layer: exposure to those assets, and a registry admitting third-party tokenised assets as collateral. Value flows through two fee-sharing mechanisms. Gas, paid in ETH, is split on a net basis between a PKL buyback, PKL stakers, the treasury and security. Applications that choose to be verified settle a progressive share of their own protocol revenue through an on-chain router, on a marginal schedule under which the first $100,000 of annual revenue is free and the builder keeps at least 82% at any scale. PKL is the ecosystem asset that both flows feed, with a fixed supply of ten billion, no mint function, and a holder-only burn; that buyback reduces total supply, not circulating supply during the vesting years. This paper describes how each of these mechanisms works, and states what each one does not guarantee.

1 Identity

One name across every surface. The brand, the chain, the token and the name service all say Pickle.

Surface	Name
Brand	**Pickle**, with the pickle mark
Chain name in wallets and registries	Pickle Chain, chain ID **78271**
Gas token	**ETH**
Ecosystem token	**Pickle**, ticker **PKL**
Name service	display suffix **.pkl**; the registry stores bare labels

Origin. The Pickle community originates from AlphBanX, a lending protocol on the Alephium network whose ABX token had a maximum supply of 100,000,000. Pickle recognises that community with a continuity allocation of PKL (§9.5). Pickle itself is an independent Ethereum Layer 2 implementation, built from scratch in Rust rather than forked, around immediate execution, mini-block preconfirmations, and Ethereum-compatible blocks.

2 Design principles

These are constraints rather than aspirations. Where one of them costs the project something, this paper says so at the point where the cost falls.

1. **Ethereum alignment.** Pickle is an L2; security and canonical ETH derive from Ethereum.
2. **ETH is gas, permanently.** PKL is never required in order to transact.
3. **PKL is a fee-share and coordination asset**, never equity, and never a claim on any entity's profits.
4. **Wallets and tooling work unmodified.** MetaMask, Foundry, Hardhat and the standard `eth_*` surface, with the documented gaps of §3.4.
5. **Two timescales.** Execute immediately; preconfirm at a 10 ms scheduling target; confirm at a 250 ms scheduling target; finalise on Ethereum. Both intervals are scheduling targets rather than latency guarantees, and neither is a throughput claim.
6. **The chain is derivable.** Every block's full transaction list is published so that any independent party can rebuild the state and check it.
7. **Costs are paid before value is distributed.** Nothing is distributed from a base that has not first been netted against settlement, data-availability and operating costs.
8. **Compensated capital and labour are never revenue.** Liquidity-provider fees, supplier interest, creator royalties, keeper, solver and oracle payments, and user principal sit outside every revenue base, permanently.
9. **Permissionless deployment is inviolable.** Verification is optional and affects only listing, funding and promotion. It never affects the ability to deploy a contract, transact with one, or use one.
10. **State the limits.** Every trust assumption and every open bypass is disclosed next to the benefit it qualifies, not gathered at the back where it can be skipped.

A principle the list deliberately does not contain: any claim about transactions per second. Pickle publishes no throughput figure that is not a dated benchmark result from a named machine, with client, execution and confirmed rate reported separately. Nor does Pickle describe itself as decentralised at any point in this document; §3.5 states the trust model in full and §10 names the specific properties that word would require.

3 Architecture

3.1 Real-time execution: mini-blocks and EVM blocks

A single Rust sequencer orders and executes transactions immediately on arrival, so a receipt exists before the block containing it is sealed. Two producers then run on top of that execution stream, at two timescales.

At a **10 ms scheduling target** the producer seals a **mini-block**: the transactions executed in that interval, their receipts and logs, a state-diff hash, a microsecond timestamp, and a signature by the sequencer key over the header. A signed mini-block is a **preconfirmation**: the sequencer has committed to that ordering and that outcome, and the signed header is evidence against it if a later block contradicted it. Mini-blocks are exposed over their own RPC methods and a WebSocket subscription, so a latency-sensitive application can act on a preconfirmation without waiting for a block.

At a **250 ms scheduling target** the producer seals a standard Ethereum-shaped **EVM block** wrapping that interval's mini-blocks. This is the **confirmation**: what `eth_getBlockByNumber`, wallets and explorers see. The next block's timestamp is fixed when the previous one is sealed, so a transaction that reads the block timestamp sees exactly the value the sealed header commits to, which is the property that makes independent re-execution (§3.3) reproduce the same block hash.

Finality is Ethereum settlement. A block is final when the data needed to re-derive it has been committed to Ethereum and that commitment is itself final.

```
preconfirmation   signed mini-block   10 ms scheduling target
confirmation      EVM block          250 ms scheduling target
finality          Ethereum settlement of the block's data
```

Both intervals are scheduling targets rather than latency guarantees, and neither is a throughput claim. The mini-block interval is a consensus parameter of the running chain, not a per-operator tuning knob, because a replica has to agree on how the stream is cut.

3.2 Execution paths and determinism

The transaction lifecycle is short and deliberately deterministic.

1. A client submits a signed transaction over standard JSON-RPC.
2. The RPC layer recovers the signer and places the transaction on a bounded admission queue, with a per-sender cap. A full queue returns an error rather than growing memory without bound.
3. A single execution coordinator commits transactions in admission order. The same sender is never executed concurrently, which preserves nonce safety without a mempool re-ordering step.
4. Execution is immediate. The receipt exists before the containing EVM block is sealed.
5. Every accepted transaction is appended to a checksummed write-ahead log and made durable before it executes, one fsync per batch, so an acknowledged transaction survives a crash.

Three execution paths handle the work, chosen by the shape of the transaction: a native path for plain ETH transfers, a fast path for the canonical ERC-20 transfer shape that moves the balance slot directly and emits the standard event, and the general EVM for everything else. The general EVM is **revm**, pinned to the **Cancun** hardfork at compile time. The minimum gas price is likewise a compile-time consensus constant. Neither is an operator setting, because either one made tunable would let a replica reject or re-price a transaction the sequencer accepted, and derivation would halt.

3.3 Settlement, data availability, and independent verification

Each sealed EVM block produces a **derivation payload** carrying the block's full transaction list, together with the block range, parent hash, block hash and the committed timestamp. The timestamp is in the payload because it is part of the execution environment, not only of the header: a replica must execute against the same clock to seal the same hash.

Publication runs in two tiers. The full payload is stored in a content-addressed **data-availability layer**, keyed by its hash, and the sequencer verifies the acknowledgement digest before relying on it. A compact **envelope** naming that payload is then committed to an inbox contract on Ethereum. If the data-availability path fails for any reason, the full payload is posted to Ethereum as calldata instead. The chain therefore stays derivable through a data-availability outage: Ethereum always holds either the data or a commitment to data that is available elsewhere.

Every block is posted, in order, block 0 included. The inbox accepts appends only from the designated batcher key and emits an indexed event per batch; that index is the total order every verifier follows.

Batch aggregation. Ethereum does not receive one transaction per sealed block. Full data goes to the data-availability layer per block; one commitment goes to Ethereum per **aggregation window**, on a 60 s scheduling target. The instrument for that commitment, blob or calldata, is selected per submission from measured prices, because at this payload size execution gas dominates at low blob fees and plain calldata wins at high ones. Aggregation is what keeps modelled settlement cost in the low millions of dollars per year rather than the tens or hundreds of millions that a per-block posting design would incur (§12).

Independent verification. Any operator can run the same binary as a **replica**. It produces nothing. It reads the Ethereum inbox from its last applied index, fetches each payload by its content address or decodes the inline calldata fallback, re-executes every transaction in order against the committed timestamp, seals the block locally, and compares the hash to the one the sequencer committed. On any mismatch it **halts permanently**: divergence is terminal by design, since retrying would seal further blocks and walk away from the canonical chain, burying the cause. A synced replica serves the standard read RPC from its own independently derived state. Its balances, receipts and headers are locally re-derived, not trusted copies of the sequencer's answers. Transaction submission is forwarded to the sequencer; a replica accepts nothing into a local mempool.

3.4 EVM compatibility

The standard `eth_*` JSON-RPC surface is sufficient for MetaMask, Foundry and Hardhat: transactions, calls, logs, filters, fee history and gas estimation. Solidity contracts deploy unmodified. Gas is paid in ETH with no EIP-1559 base fee destroyed; the whole fee is the chain's to split as §6 describes.

An ETH bridge connects L1 and L2. A deposit locks ETH in the L1 bridge and emits a message with a domain-separated identifier; the sequencer's messenger waits for L1 confirmations and finalises the deposit on L2, where the identifier is recorded so it can never be replayed. Withdrawal is the mirror path, revalidated against a finalised canonical L2 receipt before funds are released. Both directions are journaled so a restart cannot double-pay.

Pickle does not claim EVM equivalence. The claim is narrower and measured: compatibility sufficient for the standard toolchain. Bloom filters, archive depth and the EIP-1559 fee market are documented gaps, and a paper claiming parity would be contradicted by the codebase it describes.

3.5 The trust model, stated in full

A paper that describes the mechanism without the trust model is marketing, so this section states what a user of Pickle relies on.

Ordering. One sequencer orders every transaction. It can reorder and it can censor. There is no forced-inclusion path by which a transaction refused by the sequencer can be included from Ethereum.

Liveness. If the sequencer stops, mini-blocks stop and the chain halts for new execution. Replicas continue to serve derived history. The sequencer key is held in an on-chain registry and can be rotated, effective the next block.

Correctness. There are no fraud or validity proofs. Correctness is checked by re-execution: any replica can detect a divergence between the published data and the committed block hash, and halts when it does. Nothing on-chain arbitrates such a divergence. A signed mini-block that a later block contradicted would be evidence, and the design records that evidence; it does not yet act on it.

Data. Everything needed to rebuild the chain is on Ethereum or committed to from Ethereum. This is the property that makes the previous paragraph meaningful: a user relies on the operator's honesty, plus the fact that the data needed to check it is public.

Bridge. The bridge messenger is a trusted key. It cannot forge or replay a message, because identifiers are derived from the originating event and recorded once finalised, but it can delay. Batcher and messenger keys are distinct, enforced at startup.

Pickle does not describe itself as decentralised. §10 names the properties that word would require: sequencer redundancy and rotation, forced inclusion, and a challenge or proof path.

4 The native oracle

Most chains treat prices as something applications fetch. Pickle publishes them in the protocol: the price of AAPL or of gold arrives as part of block production, at a canonical address, with market hours respected. This section describes how that works and what it does not guarantee.

4.1 What it is

Genesis predeploys place two contracts at canonical addresses, a price registry holding every feed's latest answer and an adapter exposing them in the interface most existing lending and derivatives code already consumes. **One system transaction per block** publishes the batch. That is the structural difference from an external oracle: a price is not a third-party transaction competing for inclusion and paying gas, it is part of the block, produced by the sequencer alongside the transactions it orders. Answers carry eight decimal places. The read surface is a batch method and a per-feed method over JSON-RPC, plus a per-feed HTTP endpoint, and the publishing transaction is flagged so a consumer can tell a system price from a user transaction.

4.2 What it covers

33 feeds, of which 19 are real-world assets.

Class	Feeds	Assets
Equity	9	AAPL, MSFT, NVDA, AMZN, GOOGL, META, TSLA, SPY, JPM
Foreign exchange	5	EUR/USD, GBP/USD, USD/JPY, USD/CHF, USD/CNH
Energy	3	Brent, natural gas, and a broad oil index
Metals	2	Gold, silver
Crypto	14	ETH, BTC, SOL, XRP, DOGE, LINK, ADA, XLM, BCH, TRX, BNB, HYPE, ZEC, XMR

4.3 How a price is formed

Two policies, because the two asset classes fail differently.

Crypto is aggregated from five public spot venues: Coinbase, Kraken, Bitstamp, Gemini and Bitfinex. Per-venue freshness is judged on a monotonic receive clock rather than on exchange timestamps, outliers are dropped, the answer is the median of what survives, and each feed declares a minimum number of sources below which it publishes nothing rather than publishing thinly.

Everything else, equities, foreign exchange, energy and metals, comes from an authenticated third-party aggregator, with integrity checks on each update and a pinned exponent so a scale change cannot pass silently.

Market hours are a first-class concept, and this is the part that makes the layer usable for real-world assets rather than only for crypto. Each non-crypto feed carries a schedule, resolved against a real time-zone database so that daylight-saving transitions are handled rather than approximated; a market status accompanies every answer; and a reopening gate prevents the first tick after a close from being treated as a continuous-trading price. A crypto oracle needs none of this. An equity price is meaningless without it.

4.4 The trust boundary

The publisher is the sequencer. A price therefore inherits exactly the trusted-operator assumption that already governs ordering (§3.5), and no more: a dishonest or compromised sequencer can publish a false price, and nothing on-chain arbitrates it. Any application consuming these feeds inherits that assumption too, and should say so to its own users.

The third-party dependency is load-bearing. Without a valid credential for the upstream aggregator, every feed sourced from it reports as unauthorised. That is 19 of 33 feeds, which is to say the entire real-world set. The crypto feeds, sourced from public venues, continue.

The aggregating service holds no signing key of any kind. The sequencer polls it over an internal interface for a batch authenticated with a shared secret, and publishes during block production; the service then reads its own result back off the chain. That division means compromising the price service does not by itself let anyone sign anything.

5 The RWA layer

The native oracle is the hard part of real-world assets. What this section adds is what is built on it, and, at least as importantly, what Pickle deliberately does not build.

5.1 Three layers

One: the native oracle, as described in §4. The claim here is narrow: a price for a real-world asset, published inside the protocol by block production, with market hours respected.

Two: exposure to those assets. Collateral is crypto. A position is long or short against a feed from §4, with a margin engine, liquidation, and positions frozen outside market hours using the same schedule and status machinery the feeds already carry. No real asset is ever held, and settlement is always in crypto.

Three: a collateral registry for third-party tokenised assets. Tokenised real-world assets issued by regulated third parties can be admitted as collateral, with risk parameters, valuation haircuts and circuit-breakers. Pickle issues nothing in this arrangement; the issuer's obligations stay the issuer's. Admitting such a token imports the credit, custody and legal standing of whoever issued it, and the risk parameters can limit exposure but cannot make a bad issuer good.

5.2 What is deliberately out of scope, and why

Pickle does not issue tokenised real-world assets. Not issuance, not custody, not reserve attestation, not redemption, and not KYC-gated transfer restrictions on its own tokens.

The reason is that backed issuance requires things a chain cannot supply: a custodian under contract, attested reserves, and a legal entity willing to act as issuer. Those belong to issuers, and layer three is the rail by which their tokens reach Pickle as collateral.

5.3 Naming, stated plainly because it matters

Layer two is **exposure** to a real-world asset, not the asset. It is not tokenised AAPL and this paper will not call it that. A holder of such a position owns a claim against a margin system denominated by a price feed. They do not own a share, they have no shareholder rights, and there is nothing to redeem. Layers one and three are the parts that carry the real-world-asset label honestly: a price of a real thing, and a rail for tokens somebody else has properly issued.

The classification position, stated in full so that it does not have to be inferred: the feeds are prices, not assets. A separate layer creates exposure measured against those prices. Pickle does not tokenise the underlying instrument.

6 Transaction fee sharing

6.1 Who pays gas

Every transaction pays gas used times gas price, in ETH, from the sender. Failed and reverted calls consume gas and pay for it, as on Ethereum. There is no base-fee burn: the entire fee is credited to the block beneficiary and then split inside the same transaction, on all three execution paths, with the shares summing back to the fee exactly so that no wei is created or stranded.

6.2 The net basis

What the chain distributes is what remains after the chain has paid for itself:

```
R_net = max(0, gas fees - settlement - data availability - operations - security)
```

Distributing from a gross base would pay out money the chain owes to Ethereum. The netting is therefore structural, not a policy that a treasury applies afterwards.

6.3 The split

That net amount is split four ways.

Share	Destination	Paid in
40%	PKL buyback (§8)	ETH, converted to PKL and burned
30%	PKL stakers	ETH
20%	Treasury	ETH
10%	Security and insurance	ETH

The split is a set of protocol constants rather than operator settings, and its destination addresses are resolved once per process, so that the split cannot change under a running chain and fork a replica. The buyback share reduces PKL's total supply, not its circulating supply during the vesting years; §8.4 gives the arithmetic.

6.4 The staking contract

Staking is a **fee-share vault**. It does not produce blocks, does not secure the chain, has no slashing, and is not a validator bond. A holder stakes PKL; incoming ETH is accounted with a per-share reward index; the holder claims ETH; unstaking starts a **7-day unbonding period** after which the PKL can be withdrawn. All value-moving entry points are non-reentrant. If a reward payout fails, the amount moves to a deferred balance for that staker rather than blocking anyone else, and if ETH arrives while nothing is staked it is forwarded to the treasury rather than stranded.

Stakers never receive PKL emissions. A yield paid in the staked asset is a transfer from non-stakers, not revenue. When the chain earns little, stakers earn little; there is no floor, and interfaces display realised historical distributions only, never a projected rate.

6.5 Scale

At the modelled Mid activity level of 2M transactions per day at $0.003 average fee, gross gas revenue is $2,190,000 against a modelled cost base of $1,840,000, leaving $350,000 net: of which $140,000 to buyback and $105,000 to stakers.

The chain needs roughly 1.7M transactions per day at that cost base to cover its own costs. Below it, the treasury subsidises operations. That figure is published because a chain unwilling to state its break-even is asking to be taken on faith.

7 Application fee sharing

7.1 The idea

A general-purpose chain captures value only from gas. The businesses built on it, exchanges, lending markets, marketplaces, games, keep all of their revenue, and each launches its own token in order to monetise, which fragments attention and liquidity across the ecosystem it sits in.

Pickle inverts that. An application that wants ecosystem support asks to be **verified**, and a verified application settles a published share of its **protocol take** through an on-chain router, where that share feeds the single ecosystem asset. The claim is not that this is generous; it is that application revenue on Pickle is *verifiable on-chain* rather than estimated by third parties.

7.2 Verification

The fee share applies to verified applications only, and verification is the application's choice. Any application deploys and runs on Pickle without asking anyone. An application that wants more applies for verification on the dApp directory, Pickle's own listing of applications on the chain.

Verification means Pickle reviews the application's code. That access is what makes the fee share workable: the review identifies the contracts, the fee recipients and the revenue base under the category adapter, and wires the router in with the builder rather than guessing at the integration from the outside. It is also what the verified mark on the directory stands for: an application whose code has been read and whose revenue is visible through the router.

What a verified application receives:

- the **verified mark** on the dApp directory, where unverified applications are not listed;
- **featured placement** on the directory and on Pickle's own site, and promotion through Pickle's social channels;
- eligibility for grants, liquidity support and incentives from the allocated pools of §9.3.

What it gives in return is the schedule below, on protocol take only. An application that does not apply keeps all of its revenue and receives none of the above. That is the whole of the difference, and it is stated here so that nobody reads the fee share as a tax on the chain.

7.3 What counts as protocol take, and what never does

Protocol take is the application's own revenue. It is never any of the following, at any tier, permanently:

Never in any revenue base	**Why**
Liquidity-provider fees, supplier interest	Compensation to capital
Creator royalties	Payment to creators
Liquidator bounties; keeper, solver, oracle and relayer payments	Compensation to service providers
User principal, trader profit and loss, margin, funding payments, seller proceeds, collateral	Counterparty money, not revenue
Pass-through infrastructure costs	Costs

A chain that taxes its own liquidity does not have liquidity to tax. The arithmetic behind the rule rather than the sentiment: taxing an automated market maker's *total* fee instead of its protocol take would cut provider returns by roughly 40% to 48%, and capital compares returns across chains within days. This exclusion list is one of the parameters that no simple token vote can change (§10).

7.4 How it works

Component	**Responsibility**
AppRegistry	The verified applications: tier, published rate schedule, category adapter, bond, suspension. Governed and timelocked
FeeRouter	Receives protocol take, splits builder retention from ecosystem share, emits a public revenue event
RevenueVault	Multi-asset accumulation and allocation, per epoch
Category adapters	Define the revenue base per category: exchange, perpetuals, lending, marketplace, game, names, subscription, generic

Rates are a published schedule per tier, **never negotiated per application**. Every settlement and allocation emits a public event, so the revenue dashboard is a read over those events rather than a spreadsheet somebody maintains.

Integration is deliberately small, and it is done together with the builder during verification. For an automated market maker it is one line: point the pool's fee recipient at the router.

7.5 The schedule

Brackets are **marginal**, applying only to the revenue that falls inside them. Cliff rates were rejected deliberately: a rate that jumped on the whole base at a threshold would reward an application for holding its reported revenue just underneath one.

Annual protocol take	Marginal ecosystem share	Builder keeps
First $100,000	0%	100%
$100,000 to $500,000	6%	94%
$500,000 to $1,000,000	12%	88%
Above $1,000,000	18%	82%

Cumulative ecosystem share at the bracket edges: nothing at $100,000, $24,000 at $500,000, and $84,000 at $1,000,000.

Stated as a headline rather than a table: **the builder keeps at least 82% at any scale, and everything below $100,000 is free.**

7.6 Effective rates

Because the brackets are marginal, the effective rate is always below the top marginal rate and approaches it only slowly.

Annual protocol take	Ecosystem share	Effective rate	Builder keeps
$50,000	$0	0.00%	$50,000
$250,000	$9,000	3.60%	$241,000
$500,000	$24,000	4.80%	$476,000
$1,000,000	$84,000	8.40%	$916,000
$5,000,000	$804,000	16.08%	$4,196,000
$20,000,000	$3,504,000	17.52%	$16,496,000

At twenty million dollars of annual revenue the effective rate is 17.52%, still under the 18% top bracket, and it never reaches it at any scale.

Two commitments harden the schedule. The builder keeps the majority at every bracket, permanently. And rates change only by a governed, timelocked vote on the whole tier, **never per application**, because per-application negotiation is how ecosystems get captured.

7.7 Worked examples

What each example excludes matters as much as what it includes.

Exchange. Ten million dollars of daily volume at a 0.30% swap fee, split 0.25% to liquidity providers and 0.05% to the protocol. The provider share is never touched. Annualised protocol take $1,825,000: ecosystem share $232,500, an effective 12.74%, and the builder keeps $1,592,500.

The daily flow, at the top marginal rate:

```
$30,000/day total fees
|- $25,000  ->  liquidity providers, inside the pool   (never touched)
`-  $5,000  ->  protocol take, through the FeeRouter
        |- $4,100  ->  the application's treasury  (82%)
        `-   $900  ->  ecosystem share:
                       $540 buyback / $180 treasury
                       $90 security / $90 incentives
```

That buyback reduces PKL's **total** supply. It is not a reduction in circulating supply during the vesting years, and §8.4 gives the arithmetic.

Perpetuals. Fifty million dollars of daily volume at a 0.05% taker fee with 70% to the liquidity vault, plus two hundred thousand dollars of daily liquidations at a 1% penalty split evenly between liquidator and protocol. Never in the base: trader profit and loss, margin, funding payments, the vault's share, the liquidator's bounty. The adapter measures the protocol's liquidation share *as actually transferred*, never as an assumed ratio. Annualised $3,102,500: ecosystem $462,450, an effective 14.91%, builder keeps $2,640,050.

Lending. Fifty million dollars borrowed at 6% with a 15% reserve factor. Supplier interest is not revenue. Annualised $510,000 from reserve accrual and the protocol's liquidation share: ecosystem $25,200, an effective 4.94%, builder keeps $484,800.

Marketplace. Five hundred thousand dollars of daily volume at a 2% fee. Seller proceeds and the creator royalty are not revenue. Annualised $3,650,000: ecosystem $561,000, an effective 15.37%, builder keeps $3,089,000.

Fee levels differ across categories because applications price their own fees. Pickle defines only what counts as protocol take per category, never what an application may charge.

7.8 What this does not enforce

Enforcement is economic rather than physical, and the difference is worth stating rather than glossing.

Deployment is permissionless, and a sequencer-level tax is deliberately rejected. On a general-purpose EVM the transfer patterns such a tax would have to intercept are indistinguishable from account-abstraction settlement, marketplace payouts and vault withdrawals; and changing token semantics to catch them would invalidate every audit on the chain.

What Pickle offers instead is conditional: the verified mark, the listing, the promotion, grants, liquidity support and incentives all depend on verification, and verification depends on routing revenue. A bond-and-challenge mechanism prices under-reporting: a verified application posts a bond in PKL, and a successful challenge showing unrouted protocol take forfeits it and the verified mark with it. The code review at verification narrows the room for under-reporting, because the fee recipients are known rather than declared. The first-party venues make the flagship activity transparent because their own books are on-chain.

An application can decline verification and keep everything. It then forgoes the listing and the support, and that is the whole of the consequence. Spread-based revenue and off-chain revenue remain undetectable even in a verified application. This layer makes honest revenue sharing cheap, verifiable and rewarded; it does not make dishonest revenue sharing impossible, and a paper claiming otherwise would be wrong.

7.9 Application tokens

Applications at any tier may launch their own token and may distribute their retained share to its holders. Pickle is agnostic about what a builder does with the builder's share. The only rule is settlement order:

> *The ecosystem share is settled at the router, on gross protocol take as defined by the category adapter,* ***before*** *the application's treasury receives or distributes anything. It is never computed on an application-declared profit after its own distributions.*

And the verified mark means *code reviewed and revenue routed through the router*, not *token endorsed*. That distinction is stated on the directory, in the explorer and in the documentation as well as here, because it is the one a reader is most likely to blur.

7.10 The comparison a builder actually makes

	A general-purpose L2	**Pickle**
Protocol revenue retained	100%	100%, falling to 82% marginal
Revenue below $100,000	100%	100%
Grants	Competitive, discretionary	An allocated share of supply for builder incentives
Liquidity support	Competitive, discretionary	Allocated incentives plus protocol-owned liquidity
Distribution	Compete with thousands of applications	Verified listing and featured placement on the dApp directory and the site, promotion on Pickle's social channels
Own token	Yes, as the only route to monetise	Yes, optional: revenue-backed and verifiable through the router

The honest statement of the trade: Pickle offers revenue plus ecosystem support in exchange for a progressive share of protocol take. Whether that beats going it alone depends on how much distribution and capital Pickle actually supplies, an empirical question this paper cannot settle in advance, and does not pretend to.

8 Buyback and burn

Two flows fund the buyback: 40% of net gas revenue (§6) and 60% of the ecosystem share collected from applications (§7). The ecosystem share accumulates per epoch and is allocated 60% to buyback, 20% to treasury, 10% to security and 10% to incentives. Everything in this section reduces **total** supply; it does not reduce circulating supply during the vesting years, and §8.4 gives the arithmetic.

8.1 Execution

The buyback is designed around three constraints. PKL liquidity may be thin, so any mechanism assuming depth would execute badly. Pickle runs a single sequencer, so a predictable on-chain buyback is front-runnable *by the operator*, which has to be designed around rather than disclosed away. And revenue arrives in many assets, some too small to convert economically.

Primary mechanism: a sealed-bid, batch-cleared auction. The revenue vault offers a fixed amount of ETH or USDC for PKL. All bids within a window clear at a single price, so the power to order transactions confers no advantage, and there is no market order for anyone to sandwich. The execution price is public and comparable to a reference by construction. If nothing clears above the floor, no PKL is acquired and the funds stay in the vault. The auction sets no price. Filling is refused for the sequencer, batcher and messenger keys, the treasury, the vault, the market-making reserve and every vesting contract, through a timelocked on-chain denylist, and no team, contributor or treasury address trades PKL in a published window around each auction.

Secondary mechanism: time-weighted execution on the first-party exchange, under hard caps: at most 100 basis points of price impact and 0.5% of pool depth per slice, slices timed at random within a published window rather than on a fixed block, execution reverting above 102% of a 24-hour on-chain reference, and no execution at all in an epoch below $25,000 of accumulated value. The keeper is paid gas plus a fixed fee, never a proportion of size, which would pay it to execute at bad times.

Small and illiquid fee assets accumulate to a per-asset threshold before conversion under the same caps. An asset still below threshold after four epochs is marked dormant, disclosed and left in place, neither written off nor dumped. Every conversion emits input, output and reference price.

Until PKL has a market with meaningful depth, the vault accumulates and does not buy. That period is disclosed as accumulation, and no burn is claimed for it.

8.2 Burn semantics

Burn means **total supply decreases**. The burner contract holds only its own balance, calls the token's holder-only `burn()`, and emits the amount, the cumulative total and the epoch. It has no privileged role and no upgrade path. There is no `burnFrom` anywhere: an allowance-gated burn would let any approved spender, including routers holding routine unlimited approvals, destroy a holder's balance, which on a token with no mint is irreversible in a way theft is not. Nobody can burn tokens they do not hold, and nobody can create more.

Three words are never used interchangeably. **Burned** means total supply reduced, verifiable from the token's supply. **Removed from circulation** means held at an inaccessible address or locked, with supply unchanged. **Accumulated** means held in a vault under governance, which is not a reduction of anything.

8.3 Scale at the application layer

Because the first bracket is free, the ecosystem share depends on the *distribution* of application sizes and not only on the aggregate: applying the brackets to an aggregate figure would silently assume a single dominant application and overstate the result. The Mid scenario's aggregate $3,000,000 of annual protocol take is therefore modelled as an explicit distribution of eight applications, and the share is computed per application and summed.

Cohort	Applications	Take each	Ecosystem share each	Total
Large	1	$1,500,000	$174,000	$174,000
Medium	2	$500,000	$24,000	$48,000
Small	5	$100,000	nothing	nothing
Total	**8**	**$3,000,000**		**$222,000**

That is an aggregate effective rate of 7.40%, and $133,200 flowing to buyback. The same total revenue collects more from a concentrated ecosystem than from a fragmented one.

8.4 The float caveat

Combined buyback at the Mid scenario is **$273,200 per year**: $133,200 from applications and $140,000 from the chain layer. That is a real number and a small one, published because a mechanism whose scale is concealed cannot be evaluated.

The sentence that goes with it, unsoftened: **the 0% first bracket means a majority of applications contribute nothing to the buyback. That is a deliberate choice in the builder's favour, and this is its cost.**

The float caveat, which appears everywhere the burn appears. During the vesting years, scheduled unlocks add more PKL to circulation than the modelled buyback retires. The buyback reduces **total** supply. It must never be described as reducing **circulating** supply during vesting, and

cumulative burn is always published alongside float added over the same period. A buyback denominated in a fixed currency retires more PKL when the price is lower; that is a mechanical property and **not a price floor**.

9 PKL

9.1 The token

Property	Value
Ticker and decimals	PKL, 18
Supply	**10,000,000,000**, minted once at deployment
Minting after deployment	None. No mint function exists
Burning	Holder-only `burn()`. No `burnFrom`, so no approved spender can destroy a holder's balance
Owner, pause, upgrade	None. Immutability by omission
Home	The Pickle L2; any L1 representation only through the canonical bridge

PKL is a plain ERC-20 with two deliberate omissions. The absence of a mint function fixes the supply more firmly than any governance rule could, and the absence of an owner means there is nobody who can pause, upgrade or confiscate. The consequence is stated plainly: a defect in the token contract could not be patched in place; it would require a migration.

9.2 What PKL is for

Fee share. Staked PKL receives 30% of net gas revenue, paid in ETH, never in PKL emissions (§6.4).

Buyback. A standing, revenue-funded bid, at the scale published in §8.4 and subject to the float caveat there: it reduces total supply, not circulating supply during vesting.

Utility. `.pkl` names payable in PKL at a published discount, the ETH schedule being 0.05, 0.025 and 0.005 ETH per year for three-, four- and five-or-more-character names; application registry bonds posted in PKL; priority-lane access keyed to staked PKL.

Governance. Staked PKL votes on rates, tiers, incentives and treasury spend (§10).

An index over application revenue. Under the settlement-order rule of §7, every verified application's revenue, whatever token that application launches, routes a share into PKL before its own distributions. PKL's position as that index depends on applications actually choosing verification; unverified activity contributes gas alone.

9.3 Allocation

Category	Share	PKL	Launch unlock	Cliff	Vesting
Community and user incentives	17.0%	1,700,000,000	0%	-	60 months linear
Core contributors	15.0%	1,500,000,000	0%	12 months	36 months linear
Builder incentives and grants	13.0%	1,300,000,000	0%	-	60 months linear
Ecosystem treasury	12.0%	1,200,000,000	0%	6 months	54 months linear
Private round investors	10.0%	1,000,000,000	0%	12 months	24 months linear
Liquidity incentives	8.0%	800,000,000	0%	-	48 months linear
Public sale, community round	5.0%	500,000,000	50%	-	6 months linear
Security, audit and insurance	5.0%	500,000,000	0%	-	60 months linear
Protocol-owned liquidity	5.0%	500,000,000	100%	-	-
ABX migration, direct	4.0%	400,000,000	10%	-	12 months linear
Market-making and listing	4.0%	400,000,000	50%	-	24 months linear
ABX loyalty pool, earned	2.0%	200,000,000	0%	12 months	12 months linear
Total	**100.0%**	**10,000,000,000**			

Grouped by who receives it: 28% to users and the ABX community, 38% to building the ecosystem, 25% to insiders, and 9% to liquidity and market operations.

Custody, in brief. The migration sits in an audited Merkle claim contract with a twelve-month window, and unclaimed PKL returns to the treasury. Incentive pools sit behind per-epoch caps and timelocks. Contributor and investor PKL is non-transferable until vested, with schedules and addresses disclosed. **Treasury, incentive and unvested balances cannot vote**; investor tokens vote only once vested and staked.

Insiders hold 25% of supply, 15% core contributors and 10% private-round investors, and that figure is published as the upper bound rather than presented as modest. The investor allocation was funded from a former strategic reserve, from community incentives and from builder grants, never from the migration. The two rounds are laddered so that community buyers are never the exit liquidity for the private round: the private round is locked for a year and then vests over two more with nothing liquid at launch, while the community round receives half its tokens at launch and the rest over six months.

Nothing vests to contributors or investors in year one. There are no PKL emissions to stakers anywhere in this design.

9.4 The unlock schedule

	M0	M6	M12	M18	M24	M36	M48	M60
Unlocked, could circulate	990M	1,920M	2,733M	3,967M	5,200M	7,367M	9,033M	10,000M
Share of supply	9.9%	19.2%	27.3%	39.7%	52.0%	73.7%	90.3%	100.0%

Unlocked counts everything that *could* circulate, including treasury and incentive balances still held in contracts; actual float is lower, and the two are always published together where float is reported, because publishing only the lower one is the standard way a tokenomics table misleads. On day one 990M PKL is liquid: 250M from the community round, 200M for market-making, 500M of protocol-owned liquidity and 40M released to migration claimants. None of it is insider-held.

Unlocked supply rises by 1,743M PKL in the first year, then by 2,467M, 2,167M, 1,666M and 967M in the four that follow; the second and third years, where investor and contributor vesting overlap, are the steepest. Against that, the modelled combined buyback is $273,200 per year, denominated in dollars rather than PKL. The conclusion that holds at any plausible price is the one stated in §8.4: circulating supply rises during the vesting years, and the buyback offsets a small fraction of that rise rather than reversing it.

9.5 ABX community migration

Direct migration: 400,000,000 PKL, 4% of supply, to verified community holders of ABX, the token of AlphBanX on the Alephium network, 10% liquid at launch and the rest over twelve months. **Loyalty pool: 200,000,000 PKL, 2% of supply**, earned only by claimants who lock their claim for at least twelve months, releasing as those locks mature. Loyalty pays more than exit, by construction.

The per-ABX ratio is an **output** of an eligibility snapshot rather than an input to it: 400,000,000 divided by the eligible ABX at a published historical block, taken before the terms were announced. Team, treasury and protocol-contract balances are excluded by address-level classification; the script, the list and the Merkle root are published, and two independent reproductions precede fixing the root. AlphBanX's own documented allocation caps the eligible community share at roughly 35% of its 100,000,000 supply, so a literal one-for-one against total supply would hand most of the pool to insiders of the previous protocol. Preventing that is precisely what the exclusion is for.

Stated plainly, because a migration is where a project is most tempted to flatter its earliest holders: this is a continuity allocation. It does not restore prior losses, it is not compensation, and it is not intended as either. Snapshots may contain errors, eligibility will be contested, and insider classification depends on information that may be incomplete. The claim terms, root and window, once published, are among the parameters no simple vote can change (§10).

10 Governance

10.1 Voting

The voting asset is **staked PKL only**, where the seven-day unbonding period raises the cost of a flash-loan attack. Treasury, incentive, reserve and unvested balances cannot vote. A proposal requires 0.5% of votable supply or the endorsement of three delegates each above 0.25%. Quorums scale with what is at stake: 4% for parameters, 10% for treasury spend above 1% of the treasury, 15% for constitutional changes, and a proposal that fails quorum fails. Voting runs at least five days. Timelocks are seven days as standard and thirty days for the protected parameters below.

A **security council** of seven members, at most two from the core team, on staggered terms, holds powers limited to pausing and vetoing. It never initiates a transfer of value. Grant recipients recuse from votes on their own grants. Verified applications elect two council seats and hold a consultative veto on registry rate changes.

Treasury operations run through a 4-of-7 multisig with published signers, a 7-day parameter timelock, and quarterly reporting. Any manual intervention in the buyback is published within 24 hours with a reason and is never described as automated.

10.2 What no simple vote can change

Six parameters require a supermajority, a thirty-day timelock, and published legal and technical review: PKL supply and the absence of a mint function; burn semantics; the revenue-base exclusions protecting provider fees, royalties, service payments and user principal; migration terms once

published; the scope of emergency powers; and the rule that registry status never touches the ability to deploy or transact.

The first two are enforced by the absence of code, which is stronger than any governance rule can be. The third is the one most likely to be attacked: a captured governance could otherwise vote to tax provider fees, extracting value once and destroying the liquidity base permanently. Burn semantics are in the list because they are load-bearing for §8, which reduces total supply and not circulating supply during the vesting years.

10.3 Attack resistance, and its limit

Accumulation attacks meet non-voting treasury balances, higher quorums, thirty-day timelocks and the council veto. Flash-loan governance meets staked-only voting and seven-day unbonding. Proposal spam meets thresholds plus a refundable deposit. Low-turnout capture meets quorum floors. Council capture meets staggered terms and pause-and-veto-only powers. **Bribery and vote markets cannot be prevented.** Disclosure and timelocks give the community time to react, and that is the honest extent of the defence.

10.4 What decentralisation would require

Pickle does not use the word decentralised of itself. The specific properties that would have to hold before it could are these: sequencer redundancy and rotation, so that no single operator's liveness is the chain's liveness; a forced-inclusion path from Ethereum, so that no single operator can censor; and a challenge or proof path, so that a divergence a replica detects is arbitrated rather than merely reported. Until all three hold, a user relies on the operator's honesty plus the public data needed to check it, as §3.5 states.

11 Canonical constants

```
Product brand:       Pickle
Chain name:          Pickle Chain (chain ID 78271)
Gas token:           ETH (permanently)
Ecosystem token:     Pickle (PKL), 18 decimals, 10,000,000,000 fixed supply
Burn:                holder-only burn(), no burnFrom, no mint
Mini-block:          10 ms scheduling target (preconfirmation)
EVM block:           250 ms scheduling target (confirmation)
Settlement:          full data per block to the data-availability layer;
                     one commitment to Ethereum per 60 s scheduling target
Execution:           immediate; revm pinned to Cancun
Native oracle:       33 feeds, 19 of them real-world; 8 decimals;
                     one system transaction per block; the publisher
                     is the sequencer
Gas split:           on a net basis: 40% buyback / 30% stakers in ETH
                     / 20% treasury / 10% security
App revenue split:   marginal 0% / 6% / 12% / 18% ecosystem share
                     (brackets at $100k / $500k / $1M);
                     the builder keeps at least 82%, permanently
Ecosystem share use: 60% buyback / 20% treasury / 10% security
                     / 10% incentives
App fee share:       verified applications only; verification is the
                     application's choice, requested on the dApp directory
App tokens:          permitted at every tier; the router settles first
Name service:        .pkl display; 0.05 / 0.025 / 0.005 ETH per year
                     by name length
Staking:             fee share in ETH, 7-day unbond; never PKL emissions
Allocation:          17 community / 15 contributors / 13 builders
                     / 12 treasury / 10 private investors / 8 liquidity
                     / 5 public sale / 5 security / 5 protocol-owned
                     liquidity / 4 migration direct / 4 market-making
                     / 2 ABX loyalty
Insiders:            25% (15% contributors + 10% investors)
Launch float:        990,000,000 PKL liquid on day one (9.9%)
Migration:           4% direct (10% liquid at launch, 12 months)
                     + 2% earned loyalty (12-month lock)
```

Every buyback figure above reduces **total** supply. During the vesting years it does not reduce circulating supply, and §8.4 gives the arithmetic.

12 Model assumptions, and what they are worth

Every quantitative result in this paper rests on the inputs below. They are assumptions, not findings, and a reader who disagrees with one should expect the result that depends on it to move.

Input	Value
ETH price	$3,000
L1 settlement	one aggregated commitment per window, instrument chosen from measured prices
Activity: Low / Mid / High	250k / 2M / 10M transactions per day
Average fee: Low / Mid / High	$0.002 / $0.003 / $0.004
Total chain cost: Low / Mid / High	$1.42M / $1,840,000 / $2.86M per year
Aggregate application take: Low / Mid / High	$0.4M / $3,000,000 / $12M per year

Settlement cost. Posting one Ethereum transaction per sealed block would cost, at modelled parameters, $34M to $344M per year across a 3 to 30 gwei gas-price range. Aggregating to one commitment per window, on the 60 s scheduling target of §3.3, with full data on the data-availability layer, brings that to $1.0M to $2.9M per year across a 1 to 10 gwei range, roughly a hundredfold reduction at a 10 gwei gas price. A 12 s window would cost more than gross gas revenue at every modelled activity level.

The application-size distribution, which is the assumption most likely to be overlooked. The Mid scenario's aggregate take is modelled as eight applications: one at $1,500,000, two at $500,000 and five at $100,000. The ecosystem share is computed per application and summed, never by applying the brackets to the aggregate. A different distribution of the same aggregate produces a different ecosystem share, and the schedule collects more from a concentrated ecosystem than from a fragmented one.

What this paper does not model. There is no PKL price assumption anywhere in it. That is a deliberate omission with a consequence worth naming: it means the buyback of §8 cannot be expressed as a percentage of each year's float increase, because the two are denominated differently, the buyback in dollars and the float in PKL. Both sides are published in §9.4 for a reader who wants to do that arithmetic at a price of their choosing. For the same reason the paper states the size and lock terms of the private and community rounds but not their prices: a round price is a negotiating position, and a valuation derived from one would be a price claim this paper declines to make.