p76 ATA Factory Whitepaper

Version 1.3

Status: V1 shipped, mainnet verified, npm SDK live. V1.3 adds a Swap All to SOL mode that sweeps every priced SPL balance in the connected wallet into native SOL via Jupiter routing, using DexScreener for richer price/metadata coverage (pump.fun, Raydium, Meteora, Orca, etc.). V1.2 added the full Multisender suite (one-to-many, many-to-one, many-to-many, SOL relay) with native SOL support, a sidebar buy widget, and an expanded test suite.

Date: May 25, 2026

Application: `https://p76atafactory.fun/`

SDK: `https://www.npmjs.com/package/@p76/batch-sdk`

Repository: `https://github.com/itsbenjiidunn/p76-ata-factory`

Contracts (Solana mainnet):

- v1 (fuel): `9KDvApw8gAUSYEzrQLi23qSggMZKRwV99nxu877zYPdV`
- v2 (access, `$76`): `7T1F8ERfwBgrArR8QMBQw1naPv9YfBvDdrTdx3MBibb3`

Both mints are immutable. Mint authority renounced. Freeze authority renounced. Verify on Solscan or Solana Explorer before interacting.

Abstract

p76 ATA Factory is a Solana infrastructure tool for batch Associated Token Account workflows.

The app helps users create, inspect, transfer through, and close token accounts with fewer manual steps and clearer transaction visibility. It is built around the p-token `batch(0xFF)` primitive introduced with Solana's optimized Token program implementation.

`batch(0xFF)` is a p-token instruction. p-token is the Pinocchio-based Token program rewrite shipped by Anza (`github.com/anza-xyz`) and activated on Solana mainnet on 2026-05-13 via SIMD-0266. This project packages the wire format and adds an application layer; the runtime that dispatches every batch call on-chain is Anza's implementation.

The current production release ships as a client-side dApp with:

- one-to-many (uniform or per-row amounts, SPL or native SOL) - many-to-one sweep from N source wallets to one destination - many-to-many CSV distribution (`from_pk, to, mint, amount`) - SOL relay transfer through N intermediate addresses

into native SOL through Jupiter, with DexScreener-sourced price discovery

- batch ATA creation for a selected SPL mint
- batch token transfer through `batch(0xFF)`
- batch mint to many destinations through `batch(0xFF)` (mint authority required)
- batch burn across token accounts through `batch(0xFF)`

- batch close for empty token accounts through `batch(0xFF)`
- batch revoke for active token delegations through `batch(0xFF)`
- batch freeze and thaw for mints with a freeze authority
- SOL wrap / unwrap utility for wSOL workflows
- a Multisender suite with four sub-modes:
- a Swap All to SOL mode that sweeps every priced SPL balance in the wallet
- a read-only Mint Inspector for token due diligence
- a Balance Dashboard listing every non-empty token account in a wallet
- an Approval Checker that audits outstanding token delegations
- close-mode filters for ALL / FUNGIBLE / NFT / WSOL accounts
- rent reclaim from unused Associated Token Accounts
- wallet-level access tiers
- an atomic burn fee model
- measured benchmarks
- Batch Lab for wire-format inspection
- a sidebar buy widget integrating Jupiter Plugin for direct SOL -> v1/v2 swaps
- a public npm package, `@p76/batch-sdk`

p76 ATA Factory is not a generic token landing page. It is a working tool, documentation set, benchmark suite, and reusable SDK for p-token-native batch workflows on Solana.

1. Problem

Associated Token Accounts are part of normal Solana usage. Every SPL token interaction can leave account state behind. Over time, active wallets collect many ATAs from swaps, airdrops, mints, tests, markets, and one-off token interactions.

That creates three practical problems.

First, users often need to create many ATAs before distributing or moving tokens. Doing that one by one is slow and hard to verify.

Second, empty token accounts still hold rent reserve. A normal ATA rent reserve is roughly 0.00204 SOL. If the token balance is zero, the account can be closed and the rent can be reclaimed.

Third, developers need a clear way to understand p-token batch transactions. The `batch(0xFF)` instruction is powerful, but its wire format is low-level and easy to misuse if treated as a generic transaction wrapper.

p76 ATA Factory addresses these problems directly.

2. Product Scope

p76 ATA Factory has fourteen user-facing modes split across three groups:

Thaw, the Multisender suite (4 sub-modes), and Swap All to SOL.

- OPERATIONS: Batch Create, Mint, Burn, Close, Revoke, Wrap / Unwrap, Freeze /
- INSPECT: Balance Dashboard, Approval Checker, Mint Inspector.
- EXPERIMENTAL: Batch Lab.

Every active mode shares the same shape: connect wallet, configure the action, preview what will land on-chain, sign one or more atomic transactions. Reads are wallet-optional.

2.1 Batch Create

Batch Create lets a user create missing ATAs for a target SPL mint.

The flow is:

1. Connect wallet.
2. Enter a token mint.
3. Paste recipient addresses.
4. Validate and deduplicate recipients.
5. Derive each recipient ATA.
6. Query existing accounts.
7. Preview create vs skip counts.
8. Simulate and send one atomic transaction.

The V1 production path uses standard ATA program instructions because `CreateAssociatedTokenAccount` belongs to the ATA program, not the Token program.

2.2 Batch Transfer

Batch Transfer uses p-token `batch(0xFF)` for token-program Transfer instructions.

The app can build a transaction that creates missing recipient ATAs where needed, burns the p76 fee, and then sends multiple token transfers through one outer Token batch instruction.

This is the first app-level path where the `0xFF` primitive appears directly inside the production transaction after the original $76 genesis mint.

2.3 Batch Mint

Batch Mint lets a token's mint authority mint new tokens to many destinations in one atomic transaction. The same primitive that minted $76 in May 2026, productized as a general-purpose distribution tool.

The flow is:

1. Connect the wallet that holds mint authority for the target mint.
2. Enter a token mint address.
3. Enter the amount to mint per recipient.
4. Paste recipient wallet addresses.
5. App reads the mint on-chain and verifies the signer is the mint authority.

If not, the action is blocked before signing.

6. Missing recipient ATAs are added to the same transaction (idempotent

create).

7. MintTo operations are wrapped inside `batch(0xFF)` as inner instructions.
8. Wallet signs and sends one atomic transaction.

This is the first general user-facing tool to expose the `batch(0xFF){MintTo}` pattern as a self-serve product on Solana.

2.4 Batch Close

Batch Close scans a connected wallet for empty token accounts and lets the user close them in one transaction.

The app currently caps batch close at 25 empty ATAs per transaction. Each inner CloseAccount instruction is encoded inside `batch(0xFF)` with one byte of instruction data:

```
0x09
```

That single byte is the Token CloseAccount discriminator. Each inner close consumes three accounts from the outer account list:

```
[account_to_close, rent_destination, owner]
```

When the transaction lands, the rent reserve flows back to the destination, which defaults to the user's wallet.

Batch Close also surfaces filter chips so the user can target a specific kind of empty account:

- ALL
- FUNGIBLE (decimals > 0)
- NFT (decimals = 0)
- WSOL (mint = Wrapped SOL)

Wrapped SOL accounts are tagged specifically because closing them also unwraps any remaining wrapped SOL inside back to native lamports.

2.5 Batch Revoke Delegate

Batch Revoke clears active token-account delegations across a wallet in a single transaction. It is a security hygiene primitive: useful as a periodic cleanup or immediately after interacting with a dApp the user no longer trusts.

The flow is:

1. Connect wallet.
2. App scans the wallet for token accounts with a non-zero `delegatedAmount`.
3. Each candidate row shows mint address, delegate address, and approved

amount.

4. User selects which delegations to clear.
5. Token Revoke instructions are wrapped inside `batch(0xFF)` (discriminator `0x05`, one byte of inner data, two accounts per inner).
6. Wallet signs and sends one atomic transaction.

Each Revoke operation clears the delegate and the approved amount on the selected token account back to zero. Owners can re-approve a delegate later if they choose to.

2.6 Mint Inspector

Mint Inspector is a read-only mode for token due diligence. It takes any SPL mint address and returns on-chain information without signing anything.

It surfaces:

account address, raw amount, and percentage of total supply

- decimals
- current supply (raw and human-formatted)
- mint authority (or `null` if renounced, indicating immutable supply)
- freeze authority (or `null` if the mint is unfreezable)
- top 20 holders by balance (via `getTokenLargestAccounts`), each row showing
- status pills: IMMUTABLE, MUTABLE, FREEZABLE, UNFREEZABLE, NFT-LIKELY

Mint Inspector is intended for quick due diligence before buying or interacting with an unfamiliar token. It does not modify on-chain state and does not require a wallet connection to read.

2.7 Multisender Suite

Multisender bundles the four distribution patterns that show up repeatedly in Solana operational work. Each sub-mode supports either an SPL token or native SOL via an asset-type toggle (except many-to-many, which detects per row, and relay, which is SOL-only by design).

2.7.1 One-to-Many

One sender (the connected wallet), N recipients, one signed transaction.

Two amount modes:

textarea.

- UNIFORM: a single amount input is applied to every row in the recipient
- PER-ROW: each line in the textarea is `recipient,amount` (raw atomic units).

For an SPL token, all transfers are wrapped in a single `batch(0xFF)` call to amortise the per-inner CPI cost. For native SOL, transfers are stacked as individual `SystemProgram::Transfer` instructions (System Program is outside the `batch(0xFF)` boundary, so they cannot be wrapped).

2.7.2 Many-to-One Sweep

Collect tokens or SOL from N source wallets into one destination address.

Each source signs its own transaction locally; private keys are pasted into the textarea and never leave the browser. Three collection modes:

(~0.00001 SOL) is left behind so the source can pay its own tx fee.

balance are skipped with a reason.

- SEND ALL: drain the full balance per source. For SOL, a small fee reserve
- RETAIN FIXED: keep an exact amount per source and send the rest.
- FIXED PER SOURCE: send a fixed amount per source; sources with insufficient

The flow requires the user to re-paste the destination address as a typo defence before the START COLLECTION button enables, plus an explicit acknowledgement that the operation is irreversible.

2.7.3 Many-to-Many CSV

Free-form distribution. Each row is `from_pk, to, mint, amount`:

(`So11111111111111111111111111111111111111112`) to mean native SOL.

- `from_pk` is base58 OR a JSON byte array.
- `mint` is an SPL mint address, or the canonical wSOL mint
- `amount` is in raw atomic units (lamports for SOL).

Rows are grouped by source. Within a source group, multiple rows that share the same SPL mint collapse into one `batch(0xFF)` call to save CU; SOL rows route to `SystemProgram::Transfer`. Each source signs one transaction.

2.7.4 SOL Relay Transfer

Route SOL through N user-generated intermediate addresses to obscure the transfer path in casual on-chain analysis.

The flow has two phases:

1. GENERATE CHAIN: the app generates `N` random keypairs (1 to 10), computes

the per-hop carry amount (each hop subtracts the protocol fee), and offers a JSON download of the relay keypairs. The download is the only way to recover funds if a hop fails mid-chain.

2. TRANSFER NOW: the user signs hop 0 with their wallet; each subsequent hop

is signed locally with the corresponding generated relay keypair and sent via `connection.sendRawTransaction`.

The UI is explicit about what relay does and does not provide:

> **NOT ANONYMITY - PATH OBFUSCATION. Every hop is on-chain and publicly**

> **traceable. A determined observer can still link source to destination via**

> **timing and amounts. This is not zero-knowledge mixing.**

The acknowledgement checkbox forces the user to confirm they downloaded the recovery JSON before the TRANSFER NOW button enables.

2.8 Buy Widget

A persistent sidebar block shows both p76 contract addresses (V1 and V2) with three actions each: COPY the address, open it on Solscan, or click BUY.

BUY opens an embedded modal containing Jupiter Plugin (`plugin.jup.ag/plugin-v1.js`), preconfigured with `SOL` as the input mint and the selected p76 token as the output mint. Quotes, routing, and execution all happen inside the embed using the user's connected RPC endpoint. A footer "open in jup.ag" link is provided as a fallback if the embed fails to initialise.

The widget exists for one reason: the project is the tool and the token. The contract addresses are the primary thing a visitor needs to be able to see and act on without leaving the app.

2.9 Swap All to SOL

Swap All to SOL is a one-click wallet sweep: scan every SPL token balance the connected wallet holds, price each one, let the user select which to liquidate, then route the selected balances to native SOL through Jupiter.

The flow is:

1. Connect wallet.
2. App calls `discoverNonEmptyAtas` and lists every token account with a

non-zero balance.

3. For each mint, the app queries DexScreener for symbol, name, and USD price

from the highest-liquidity Solana pair where that mint is the base token. USDC, USDT, and PYUSD are hard-coded to `$1` so stables always price even if the DEX call momentarily fails. Mints with no DEX pool show a blank placeholder instead of a price.

4. Rows are sorted by USD value (descending) and pre-selected by an

auto-threshold (default `$0.10`, user-adjustable).

5. The user adjusts selection (per-row toggle, select-all, invert) and sets

slippage in basis points (default `300` = 3%).

6. On execute, the app fetches a Jupiter quote and swap transaction for each

selected mint in parallel. Each swap is a separate versioned transaction (Jupiter swaps depend on address-lookup tables and are too large to coalesce into one tx).

7. If the wallet exposes `signAllTransactions`, the entire batch is signed in

a single wallet prompt. Otherwise the app falls back to sequential signing, one prompt per swap.

8. Signed transactions are submitted sequentially, each confirmed before the

next is sent. A failing swap reports its error in the result list and does not abort the remaining swaps.

Design choices worth calling out:

data for tokens in its verified registry, which excludes most fresh pump.fun mints, low-cap memecoins, and unverified Raydium / Meteora listings. DexScreener indexes every Solana pool regardless of curation, so the user's actual wallet contents are visible.

remains the strongest aggregator for *swap routing* across Raydium, Orca, Meteora, Phoenix, Lifinity, and pump.fun. The two systems are decoupled: data layer vs execution layer.

silently truncates the pairs response when several mints are queried at once, dropping some tokens entirely. Per-mint parallel calls return complete data and stay well under the 300-RPM free-tier limit.

Solflare, Backpack all support it); the sequential fallback keeps support for wallets without batch signing.

- DexScreener over Jupiter Price API: Jupiter's price endpoint only returns
- Jupiter for execution despite using DexScreener for pricing: Jupiter
- One DexScreener call per mint: the batched `/tokens/<a,b,c>` endpoint
- Hybrid signing: `signAllTransactions` minimises wallet prompts (Phantom,

Swap All to SOL is the natural cleanup primitive that pairs with Batch Close: Close reclaims rent from empty token accounts, Swap All converts the remaining live balances into SOL so the wallet ends up consolidated.

---

3. Core Technical Primitive

p76 ATA Factory is built around Solana's p-token batch instruction:

```
batch(0xFF)
```

The wire format is:

```
[0xFF] [num_accounts: u8] [ix_data_len: u8] [ix_data: bytes] ...
```

There is no inner `program_id` field.

That matters. `batch(0xFF)` is not a generic cross-program transaction wrapper. It dispatches inner instructions inside the Token program processor.

Valid batch targets include token-program instructions such as:

- Transfer
- MintTo
- Burn
- CloseAccount
- TransferChecked
- InitializeAccount variants

Invalid batch targets include:

- System Program instructions
- Associated Token Account Program instructions
- Compute Budget instructions
- arbitrary custom program instructions

This boundary is central to p76 ATA Factory's design. The app does not claim that ATA creation itself is wrapped in `batch(0xFF)`. ATA creation stays as a top-level ATA program instruction. Token-native operations use the batch primitive where it is valid.

Native SOL transfers sit firmly outside the boundary. SOL is owned by the System Program (`11111111111111111111111111111111`), not the Token program, so a `SystemProgram::Transfer` cannot appear inside a `batch(0xFF)` inner slot. This is why the Multisender suite has two distinct code paths: SPL flows use `batch(0xFF){Transfer x N}` for CU savings; SOL flows stack plain `SystemProgram::Transfer` instructions inside the same outer transaction. The wSOL mint (`So111...112`) refers to the wrapped-SOL SPL token, which is a different asset; the Multisender treats the wSOL mint string as a user-friendly alias meaning "native SOL" because that matches operator intent in practice.

---

4. Transaction Architecture

The app uses atomic transactions. If any required step fails, the whole transaction fails.

4.1 Batch Create transaction

```
ComputeBudget::SetComputeUnitLimit
ComputeBudget::SetComputeUnitPrice
Token::Burn                 (p76 fee)
ATA::CreateIdempotent x N   (missing ATAs)
```

This path is size-bound. Current measurements put the max safe production batch at 11 ATA creates per transaction.

4.2 Batch Transfer transaction

```
ComputeBudget::SetComputeUnitLimit
ComputeBudget::SetComputeUnitPrice
Token::Burn                 (p76 fee)
ATA::CreateIdempotent x N   (only where needed)
Token::Batch(0xFF) {
  Transfer x N
}
```

Transfer is a native Token program instruction, so it can be wrapped by `batch(0xFF)`.

4.3 Batch Mint transaction

```
ComputeBudget::SetComputeUnitLimit
ComputeBudget::SetComputeUnitPrice
Token::Burn                 (p76 fee)
ATA::CreateIdempotent x N   (only where dest ATAs do not exist)
Token::Batch(0xFF) {
  MintTo x N
}
```

Same outer wrapper shape as Batch Transfer, with MintTo (discriminator `0x07`) as the inner. The signer must hold the mint authority of the target mint or the transaction is rejected on-chain. The app pre-checks authority before exposing the sign button to avoid wasted tx fees.

4.4 Batch Close transaction

```
ComputeBudget::SetComputeUnitLimit
ComputeBudget::SetComputeUnitPrice
Token::Burn                 (tier-scaled p76 fee)
Token::Batch(0xFF) {
  CloseAccount x N
}
```

CloseAccount is also a native Token program instruction. This is the cleanest rent-reclaim path in the app because every selected empty account can be closed through one outer batch instruction.

4.5 Batch Revoke transaction

```
ComputeBudget::SetComputeUnitLimit
ComputeBudget::SetComputeUnitPrice
Token::Burn                 (p76 fee)
Token::Batch(0xFF) {
  Revoke x N
}
```

Each inner Revoke is one byte of instruction data (discriminator `0x05`) and two accounts per inner (`[token_account, owner]`). No ATA creation step is needed because Revoke operates only on existing token accounts.

4.6 Multisender one-to-many (SPL)

```
ComputeBudget::SetComputeUnitLimit
ComputeBudget::SetComputeUnitPrice
Token::Burn                 (p76 fee)
ATA::CreateIdempotent x N   (only where needed)
Token::Batch(0xFF) {
  Transfer x N (per-row or uniform amount)
}
```

Same shape as 4.2 Batch Transfer but allows per-recipient amounts.

4.7 Multisender one-to-many (native SOL)

```
ComputeBudget::SetComputeUnitLimit
ComputeBudget::SetComputeUnitPrice
Token::Burn                 (p76 fee)
SystemProgram::Transfer x N
```

No `batch(0xFF)` because System Program is outside the wrapper. Each SystemProgram::Transfer is about 80 bytes with accounts, so this fits roughly 12 recipients in a single transaction before hitting the 1232-byte limit.

4.8 Multisender many-to-one sweep (one tx per source)

```
[per source keypair]
ComputeBudget::SetComputeUnitLimit
ComputeBudget::SetComputeUnitPrice
Token::Transfer  (SPL)    OR  SystemProgram::Transfer  (SOL)
```

Each source signs its own minimal tx and pays its own fee. The connected wallet does not need to sign and is not part of the source set. There is no p76 fee burn because the burning wallet would have to be the source, which may not hold v1.

4.9 Multisender many-to-many (one tx per source, mixed payloads)

```
[per source keypair]
ComputeBudget::SetComputeUnitLimit
ComputeBudget::SetComputeUnitPrice
[for each mint group owned by this source]
  if mint = wSOL alias:
    SystemProgram::Transfer x M
  else if M >= 2:
    ATA::CreateIdempotent x M
    Token::Batch(0xFF) { Transfer x M }
  else:
    ATA::CreateIdempotent
    Token::Transfer
```

The per-source tx is built lazily from the parsed CSV. Same-source same-mint rows collapse into a batch call; single rows or SOL rows fall through to plain instructions.

4.10 SOL relay (N+1 sequential txs)

```
[hop 0, signed by user wallet]
ComputeBudget x2
SystemProgram::Transfer  user -> relay_0

[hop i = 1..N-1, signed by relay_{i-1}]
ComputeBudget x2
SystemProgram::Transfer  relay_{i-1} -> relay_i

[hop N, signed by relay_{N-1}]
ComputeBudget x2
SystemProgram::Transfer  relay_{N-1} -> destination
```

Each hop is a distinct transaction with its own signer. Each subtracts a small lamport amount from the carry to cover the next signer's fee. If any hop fails, the lamports stop at the last successful relay; the user's downloaded keypair JSON is the recovery path.

4.11 Swap All to SOL (one Jupiter tx per selected mint)

```
[per selected mint, parallel-fetched from Jupiter then signed in a batch]
VersionedTransaction (v0) {
  ComputeBudget x2
  [route-specific Token / DEX program instructions assembled by Jupiter]
  ATA::CreateIdempotent      (destination wSOL ATA, if needed)
  Token::SyncNative          (after wSOL receive, before unwrap)
  Token::CloseAccount        (auto-unwrap wSOL back to native SOL)
  address lookup tables x N  (to fit large route account sets)
}
```

The exact instruction list inside each tx is built by Jupiter's `/swap/v1/swap` endpoint based on the chosen route plan; the app does not assemble the inner instructions itself. The user wallet signs all selected txs (one wallet prompt if the wallet supports `signAllTransactions`, otherwise one prompt per tx). The app then sends them sequentially, confirming each before moving on.

This is the only mode in the app that signs versioned (v0) transactions; all batch(0xFF) flows use legacy transactions because they do not need ALTs at their current size.

---

5. Access and Fee Model

p76 ATA Factory uses the p76 token pair only as app infrastructure.

The contracts on Solana mainnet:

Mint	Address	Decimals	Role
v1	`9KDvApw8gAUSYEzrQLi23qSggMZKRwV99nxu877zYPdV`	6	fuel / per-action burn fee
v2 (`$76`)	`7T1F8ERfwBgrArR8QMBQw1naPv9YfBvDdrTdx3MBibb3`	6	access / tier

Both mints are immutable. Mint authority renounced. Freeze authority renounced. Supply cannot grow, accounts cannot be frozen, no admin can pause trading. Verify on Solscan before holding or interacting.

The roles are:

- v2 is access. Holding more v2 raises the user's app tier.
- v1 is fuel. The app burns v1 as the per-action fee.

The app does not send fees to a treasury. The fee is burned atomically inside the same transaction as the user action. The CLI does not charge any fee; it calls the same pure libraries with feeRaw set to zero.

5.1 Batch create tiers

Batch create limits are based on measured transaction size.

Tier	v2 balance	Max ATA creates
Observer	0	3
Builder	>= 0.1	6
Operator	>= 0.5	9
Factory	>= 1	11

The Factory tier is capped at 11 because the Solana transaction size limit is the binding constraint for createATA today. The same tier cap applies to Batch Mint when it co-creates destination ATAs in the same transaction.

5.2 Batch close fee tiers

Batch close uses a tier-scaled burn fee. The larger the user's v2 holding, the lower the v1 close fee.

Tier	Close fee
Observer	0.001 v1
Builder	0.0001 v1
Operator	0.00001 v1
Factory	0.000001 v1

This matches the product goal: rent recovery should become progressively cheaper for stronger p76 holders.

5.3 Other operations fee

Batch Create, Batch Transfer, Batch Mint, Batch Burn, Batch Revoke, Batch Freeze/Thaw, Wrap/Unwrap, and Multisender one-to-many use a flat fee of `0.001 v1` per batch transaction regardless of tier. These are active service operations rather than rent-recovery utilities, so the same flat burn applies to all.

Multisender many-to-one, many-to-many, and SOL relay do NOT burn the p76 fee on each sub-transaction because the signing wallet in those modes is the pasted source keypair (which may not hold v1), not the connected user wallet.

Swap All to SOL does NOT burn the p76 fee. The economic surface there is already non-trivial (Jupiter routing fees, DEX LP fees, per-tx network fee across N swaps); adding a per-batch burn on top would complicate the user math without much upside. The mode exists primarily as a convenience layer over Jupiter, not as a paid service.

Mint Inspector, Balance Dashboard, and Approval Checker are read-only and have no fee.

6. Benchmarks

p76 ATA Factory publishes measured numbers instead of guessed performance claims.

The first benchmark run was completed on May 22, 2026. The repo includes:

- `scripts/benchmark.ts`
- `scripts/verify.ts`
- `docs/BENCHMARKS.md`
- `benchmarks/raw-2026-05-22.csv`

Key measured results:

Metric	Value
Per-ATA transaction size growth	74 bytes
Production size at 10 ATA creates	1,133 bytes
Production size at 11 ATA creates	about 1,207 bytes
Production size at 12 ATA creates	about 1,281 bytes, over limit
Max safe production create batch	11 ATAs
Devnet measured createATA CU	about 15,514 CU per ATA
Mainnet observed createATA CU	8-12k CU per ATA
Mainnet batch CloseAccount CU	about 2.5k CU per inner close

The main conclusion is:

> **For batch ATA creation in V1, transaction size is the constraint. Compute is**

> **not.**

At the measured create limit, compute usage is still far below the 1.4M transaction compute budget. The serialized transaction reaches the 1232-byte limit first.

Batch CloseAccount behaves differently. Each inner close has only one byte of instruction data, so the close flow is far more compact than createATA.

7. Mainnet Proof

p76 ATA Factory has landed real production transactions on Solana mainnet.

Tx	Mode	N	CU consumed	Result
`M3q3...UHnSok`	createATA	3	42,473	burned fee
`Cw277...39nzh`	createATA	5	55,817	burned fee
`4PNq...ouKm`	createATA	5	46,817	burned fee
`3ZGf...Nn2G`	createATA + `batch(0xFF){Transfer x 3}`	3	52,877	batch Transfer verified
`2QWh...3ABG`	`batch(0xFF){CloseAccount x 5}`	5	12,634	reclaimed 0.010121 SOL net of fees

The app has now production-verified three important `batch(0xFF)` patterns:

Inner op	Proof
MintTo	original $76 genesis transaction
Transfer	p76 ATA Factory transfer transaction
CloseAccount	p76 ATA Factory batch close transaction

This is the core claim of the project: p76 ATA Factory is not only a UI. It is a mainnet-tested batch tooling project.

---

8. Batch Lab

Batch Lab is the developer-facing inspection mode.

It gives users and developers a way to study the transaction structure instead of trusting marketing copy.

Batch Lab includes:

- transaction size inspection
- compute reference data
- account count visibility
- raw `batch(0xFF)` decoding
- a wire-format viewer
- a worked genesis batch example
- comparison between standard ATA creation and future p-ATA expectations

Batch Lab turns the app into living documentation. It is useful for users who want to understand what they are signing and for developers who want to inspect the primitive.

---

9. SDK

The reusable developer layer is published as:

```
https://www.npmjs.com/package/@p76/batch-sdk
```

Install:

```
npm install @p76/batch-sdk
```

Current published version:

```
0.2.0
```

The package provides:

- `encodeBatchData`
- `decodeBatchData`
- `transferInner`
- `mintToInner`
- `burnInner`
- `closeAccountInner`
- `transferCheckedInner`
- `TOKEN_IX` constants
- TypeScript declarations
- ESM and CommonJS builds
- zero runtime dependencies

Example:

```
import {
  closeAccountInner,
  encodeBatchData,
  transferInner,
} from '@p76/batch-sdk';

const data = encodeBatchData([
  transferInner(1_000_000n),
  transferInner(2_500_000n),
  closeAccountInner(),
]);
```

The SDK does one job: encode and decode the p-token `batch(0xFF)` format correctly. It does not try to become a wallet adapter, RPC client, or transaction framework.

The npm package matters because it makes the p76 ATA Factory work portable. The app proves the primitive. The docs explain the primitive. The package lets other developers use the primitive.

10. p-ATA Readiness

p76 ATA Factory is p-ATA ready by design.

The current ATA creation path uses the standard Associated Token Account program. The app keeps the ATA program ID centralized so that a future p-ATA program path can be adopted with minimal surface change when it is live and safe to use.

The project does not claim p-ATA performance before p-ATA is live. Current performance claims are based on measured V1 behavior.

Expected future improvements include:

- lower compute for ATA creation

- larger practical batch flows when combined with address lookup tables
- better UX for high-volume token distribution and cleanup

The project separates readiness from hype. V1 works today. Future paths are documented as future paths.

---

11. Security and Failure Model

p76 ATA Factory is a client-side app. There is no backend custody and no database.

Important safety properties:

parsed and signed entirely in the browser. They are never sent to any server, never logged, and never persisted across reloads

address before the execute button enables (typo defence)

starting, and labels itself "path obfuscation, not anonymity"

many-to-one or many-to-many does not abort the run, it is reported as FAILED in the result list and the remaining sources continue

- wallet signs every transaction
- fee burn is atomic with the user action
- failed transactions do not burn the fee
- CloseAccount only works on zero-balance token accounts
- createATA uses idempotent creation
- recipient input is validated and deduplicated
- oversize transactions are blocked before signing
- benchmark claims are reproducible from repo scripts
- Multisender private-key paths: secret keys pasted into the textarea are
- Multisender many-to-one requires the user to re-type the destination
- SOL relay forces the user to download the relay keypair JSON before
- Multisender sources sign their own transactions; a failed sub-tx in

Important limitations:

they are not safe on a compromised machine or a malicious browser extension

privacy; a determined observer can still link source to destination via hop timing and amounts

price/symbol data; Jupiter routes and builds the swap transactions. Both are trusted to return honest data and instructions, and the user's wallet is the final signature gate

the run; the remaining swaps continue and the failure is surfaced in the result list

- users must still verify wallet prompts
- closing an account is irreversible once confirmed
- rent reclaim only applies to closeable empty token accounts
- transaction limits can change with Solana runtime changes

- RPC behavior may affect simulation and discovery UX
- Multisender private-key paths only protect what the browser can protect:
- SOL relay is on-chain path obfuscation, not mixing or zero-knowledge
- Swap All to SOL depends on external services. DexScreener provides
- Swap quotes are time-sensitive. A failed sub-tx in Swap All does not abort

The app is designed to surface constraints instead of hiding them.

12. Why This Matters

Solana users do not need another dashboard that only displays balances. They need tools that reduce operational friction.

p76 ATA Factory focuses on concrete workflows:

pump.fun / Raydium / Meteora coverage via DexScreener pricing

- create many ATAs safely
- transfer through a real Token batch primitive
- mint to many destinations in one tx (genesis-of-$76 pattern)
- burn token balances across multiple accounts in one batch
- reclaim SOL from dead token accounts, with NFT and WSOL filters
- revoke active token delegations as security hygiene
- freeze and thaw token accounts (with freeze authority)
- wrap and unwrap native SOL into wSOL
- distribute SPL or SOL one-to-many, many-to-one, or many-to-many
- route SOL through user-generated relay hops for path obfuscation
- sweep every priced SPL balance into native SOL via Jupiter, with
- inspect any SPL mint for due diligence (read-only)
- audit outstanding token delegations across a wallet (read-only)
- list every non-empty token account a wallet holds (read-only)
- inspect the underlying batch wire format
- buy v1 or v2 directly from the sidebar via Jupiter embed
- publish reusable helpers for other developers

The project contribution is practical infrastructure:

- a working dApp
- mainnet transactions
- source-verified documentation
- measured benchmarks
- public SDK
- honest limitation reporting

That is the standard for this project: if a claim is made, someone should be able to open the repo and verify it.

---

13. Verification References

Application

```
https://p76atafactory.fun/
```

SDK

```
https://www.npmjs.com/package/@p76/batch-sdk
```

Repository

```
https://github.com/itsbenjiidunn/p76-ata-factory
```

Contracts (Solana mainnet)

```
v1 (fuel):          9KDvApw8gAUSYEzrQLi23qSggMZKRwV99nxu877zYPdV
v2 ($76, access):   7T1F8ERfwBgrArR8QMBQw1naPv9YfBvDdrTdx3MBibb3
```

Both mints are immutable: mint authority renounced, freeze authority renounced. Verify on Solscan (`solscan.io/token/<mint>`) or Solana Explorer before interacting.

Mainnet proofs

```
M3q3NHrpZGg5sDFuF5LpkMde3BpxiUVuYQdE1Ufkb6cjwMNVYroXeSPEEhQegqJoqpXeD8ns5w5PBx3APUHnSok
Cw277g36TDj21QPPgt5FbmgAFnAFAtvSdZfnBoCUMyMCLmuahVxsPYdtRXCm8mNhCKgtWU9EQEWqqojkwb39nzh
4PNqPTzViLf6pfSVMy6wR6VEcJ8rcwUVMBdYW2jtNVxSnyBkdNaEX4G4e5C11FnLWuYRTDrD2E28x9wnGTGFouKm
3ZGfejsPrCCKraukWmDGGi2tWiSszLB1AmsfYV6bfpghBgNjZ4AokcobLx1geDfqSVf6dMkDW6mFSuqzFzP3Nn2G
2QWhmHMWoFSh5PX2DS92GadcwUq58xndDhWPo2DVkzmXyDkwphdj5G2K7kx7kehyK8GCsAD2Dewk1qwSV4qf3ABG
```

Documentation

```
docs/BATCH_FORMAT.md
docs/BENCHMARKS.md
README.md
packages/batch-sdk/README.md
```

14. Disclosures

p76 ATA Factory is experimental Solana software. Users should inspect wallet prompts, verify addresses, and understand the effects of every transaction before signing.

The app's fee model uses p76 tokens, but using the app does not represent equity, governance, profit share, legal ownership, or a claim on any third party.

Benchmark numbers are point-in-time measurements. Solana runtime behavior, validator versions, RPC behavior, p-ATA availability, and transaction message constraints may change. The project publishes scripts and raw data so the numbers can be rerun.

15. Closing

p76 ATA Factory is a batch tooling project for Solana.

Its purpose is narrow and concrete: make ATA creation, token-account cleanup, token batch flows, and p-token wire-format inspection easier to use and easier to verify.

The dApp gives users practical workflows. Batch Lab gives developers a window into the transaction format. `@p76/batch-sdk` turns the same primitive into an installable package.

That is the whitepaper claim: p76 ATA Factory is a working, measured, mainnet-verified tooling layer for p-token-native account workflows.