Catalyst Report

Linkage Finance Team

August 23, 2025

1 Introduction

Cardano decentralized applications (DApps) often require off-chain services called batchers to facilitate complex transactions. A batcher is an off-chain node service that aggregates multiple user actions into one transaction for efficiency [**?**]. In the current ecosystem, each DApp team typically develops its own batcher or off-chain code from scratch, duplicating effort and ultimately achieving inconsistent quality. This redundancy slows development and introduces variability in reliability. To address this problem, our project's first milestone focuses on designing a reference batcher architecture that can be reused across projects. By providing a unified, open-source batcher framework, we aim to streamline Cardano DApp development, ensuring efficiency and consistency across projects. This report details the comprehensive architecture design for the reference batcher and the selection of tools and libraries for its implementation.

2 Background

On Cardano, batchers play a crucial role in enabling complex DApp functionalities such as decentralized exchanges (DEXs), crowdfunding protocols, and fund management. Unlike Ethereum's account-based model, Cardano's extended UTXO model does not maintain a global shared state for smart contracts. This means that certain multi-party or multi-step operations (like matching trades or bundling fund disbursements) cannot be executed entirely on-chain in a single transaction per user. Batchers fill this gap by running off-chain logic: they collect pending user requests or orders, combine them, and submit an aggregated transaction that triggers on-chain scripts for all at once. This approach improves throughput and reduces blockchain load, since one batch transaction can replace many individual ones. In addition, we are solving the problem of UTXO contention. It also significantly lowers fees for users by amortizing the cost of a single transaction over many operations. For example, batching 30 token transfers into one transaction was shown to use 0.29 ADA in fees versus 5.1 ADA if done as 30 separate transactions [**?**] – an order-of-magnitude fee reduction.

Despite these benefits, developing a batcher is non-trivial. Each project has so far implemented its own custom off-chain infrastructure, which duplicates effort and may miss best practices. Common challenges include ensuring atomicity (all operations in the batch succeed or none do), preventing race conditions [**?**] (ordering the actions to avoid conflict and maintaining security of the batcher funds. Our reference batcher initiative is motivated by the need to standardize solutions to these challenges. By offering a well-designed, open-source batcher architecture, we enable Cardano developers to avoid reinventing the wheel and instead focus on their DApp's unique logic. An open-source reference also promotes decentralization: it can be run by multiple community operators, not just the original developers, reducing single points of failure in the ecosystem. The following sections describe our first architecture report and the tools we are planning to use to implement the reference batcher.

3 Reference Batcher Architecture Design

3.1 Design Objectives

When designing the architecture for our reference batcher we defined the following key-objectives

1. **Reusability** - The batcher should be general enough to adapt to various DApp use-cases (DEXs, fund management, marketplaces, etc) with minimal changes. The main changes should be implementing the DApp specific business logic.

2. **Modularity** - We are following one of the basic software design principles of Modularity. The idea is that the system/code setup is divided into clear components and functions so that developers can easily replace or modify parts (for example, the matching logic for orders) without affecting the entire logic of the DApp.

3. **Reliability** - the batcher must handle failures gracefully, for example if a specific smart contract transaction fails. T

4. **Security** - While security of user funds should be ensured through smart contract logic it is also important to ensure safeguarding of batcher private keys

5. **Transparancy** - As an open-source tool, the design should be clear and well-documented to support the community and contribute in a meaningful way.

In addition, we are also planning to incorporate known best practices for existing batchers such as sorting or prioritizing inputs to avoid UTXO conflicts and failed transactions.

3.2 System Overview

The batcher operates as a continuously running off-chain service connected to the Cardano network. Figure 1 provides a high-level overview of its operation. At its core, the batcher monitors a set of input sources for new user requests or data that need processing. These sources can be on-chain or off-chain: for instance, UTXOs locked in a specific script address (representing user orders or fund contributions on-chain), or an external input (such as a file or API feed, depending on the DApp). The batcher's **Processing Engine** periodically gathers all new inputs and determines how to bundle them. This may involve matching complementary actions (e.g. matching buyers and sellers in a DEX) or simply grouping similar operations (e.g. aggregating multiple fund disbursements or token transfers). The output of this stage is a proposed batch transaction that encapsulates all the intended operations. The batcher then uses a **Transaction Construction Module** to build a Cardano transaction containing the necessary inputs, outputs, and script calls for the batch. Finally, the transaction is *signed and submitted* to the network through the batcher's node connection. Throughout this process, the batcher performs validations (ensuring that all inputs are still unspent and conditions are met) and handles any exceptions (if an input became invalid, it can be dropped or the batch rescheduled).

Components and Workflow: The reference batcher is built from several connected parts. Each part has a clear role in the overall process of collecting, grouping, and sending transactions on Cardano.

- **Node Integration Layer:** This part connects directly to a Cardano node (or to a node API). It checks the blockchain for new information, such as fresh UTXOs at certain addresses, and it also sends constructed and signed transactions to the network. For security and reliability, the batcher should run with a fully synchronized Cardano node, ideally set up as a relay. Using a full node ensures the batcher always has correct blockchain data and can submit transactions without depending on third parties. During development, lighter API services like Blockfrost can be used, but in production a full node is recommended. However, it might make sense to connect a full-node with other tools like Blockfrost for fast and reliable analytics data.

- **Request Pool & State Manager:** This module keeps track of all pending requests that need to be batched. In a DEX, this means open buy and sell orders. In a fund batcher, this means withdrawal or distribution requests. The pool makes sure that each request is handled only once and that already-used UTXOs are not reused. If a user cancels a request or a UTXO is spent elsewhere, the pool updates itself to stay correct.

- **Batching Logic / Matching Engine:** This is the decision-making part. It chooses which requests can be grouped into a single transaction. In a trading app, it might match buy and sell orders. In a fund batcher, it

might collect all withdrawals waiting at a certain time. The logic also checks that the rules of Cardano scripts are respected, such as inputs and outputs balancing, having all required signatures, and following validator rules. To avoid conflicts, it can order requests by time or priority. If the batcher charges a service fee, this is also calculated here.

- **Signing and Submission Component:** Once the batch transaction is built, this part signs it with the batcher's own key. Private keys are stored securely as part of the code base and ideally never shared or revealed. After signing, the transaction is submitted through the Cardano node. This part also checks whether the transaction is included in a block. If it fails (for example, due to a UTXO conflict), the request can be put back into the pool for another attempt.
- **Logging and Monitoring:** The batcher records what it does at each step. It logs successful batches, errors, and any skipped requests. Monitoring makes sure developers or operators are alerted if the batcher stops or if too many requests are waiting. While this does not affect on-chain logic, it is essential for running a reliable service.

With these components, the batcher follows a simple cycle of steps. First, it listens for new requests on the blockchain or through an API. Next, it collects these requests and keeps them in a pool. When ready, it decides which requests can be grouped together. It then builds a transaction for this group, signs it with the needed keys, and sends it to the Cardano network. Once the transaction is confirmed, the requests are marked as finished, and the cycle starts again.

The architecture is designed to be flexible. It can be used as a general template for many types of DApps. As a first example, we will apply it to the Linkage Finance fund batcher, which will use the same structure but with logic specific to Linkage's protocol. This will show how the reference batcher can be adapted to real projects and reused by other developers.

4 Tools and Libraries Selection

An important task in this milestone was to decide on the tools and technologies we will use to build the batcher. These include the programming language, libraries, and the basic infrastructure. The choices were made to keep the system reliable, easy to understand, friendly for open-source collaboration and newcomers to the Cardano ecosystem.

4.1 Programming Language

We decided to use **Python 3** for building the batcher. Python is simple, widely used, and has a large community. It is easy to read and write, which lowers

the barrier for other developers who may want to contribute to the project. Our team also has strong experience with Python, which helps us work faster. While Cardano smart contracts are mostly written in Haskell or Aiken, we chose Python to make the batcher more accessible. However, we are working on intergrating tools that help for making an easy connection with smart contracts written in Aiken.

4.2 Cardano Node and Network Access

The batcher connects to the Cardano blockchain through a **Cardano node**. Running our own node ensures that the batcher always has correct blockchain data and can submit transactions without relying on third parties. For testing and development, we also use the **Cardano testnet** and the **Blockfrost API**. Blockfrost allows us to quickly check UTXOs and test transactions without running a local node, which makes early development easier. However, for production we recommend using a self-hosted node to stay decentralized and independent. To make communication with the Cardano node easier and more efficient, we also plan to use **Ogmios**. Ogmios is a lightweight service that sits on top of a Cardano node and provides a WebSocket and JSON interface. This makes it much simpler for applications like our batcher to query blockchain data (such as UTXOs or block information) and to submit transactions. By using Ogmios, we avoid having to directly parse complex node outputs, and instead get a fast and developer-friendly API. This improves performance and reduces the amount of code we need to maintain in the batcher itself.

4.3 Cardano Python SDK

For building and signing transactions we use the **PyCardano** library. This open-source Python library makes it easier to create transactions, manage keys, and handle multi-asset UTXOs. It removes much of the low-level complexity of Cardano, so developers can focus on writing logic instead of dealing with binary formats. In special cases, we can also use the official **cardano-cli** tool as a backup, but PyCardano covers most of our needs. All of these libraries are open-source and actively maintained making them a good building ground for our open-source batcher template.

4.4 Development and Deployment

The project is hosted on **GitHub**, with Git used for version control. To make the batcher easy to deploy, we use **Docker**. This allows anyone to run the batcher on their machine or on a server without worrying about the setup and for us an easy way to pin our requirements. Docker ensures that the environment is the same everywhere, which avoids common errors and makes collaboration easier.