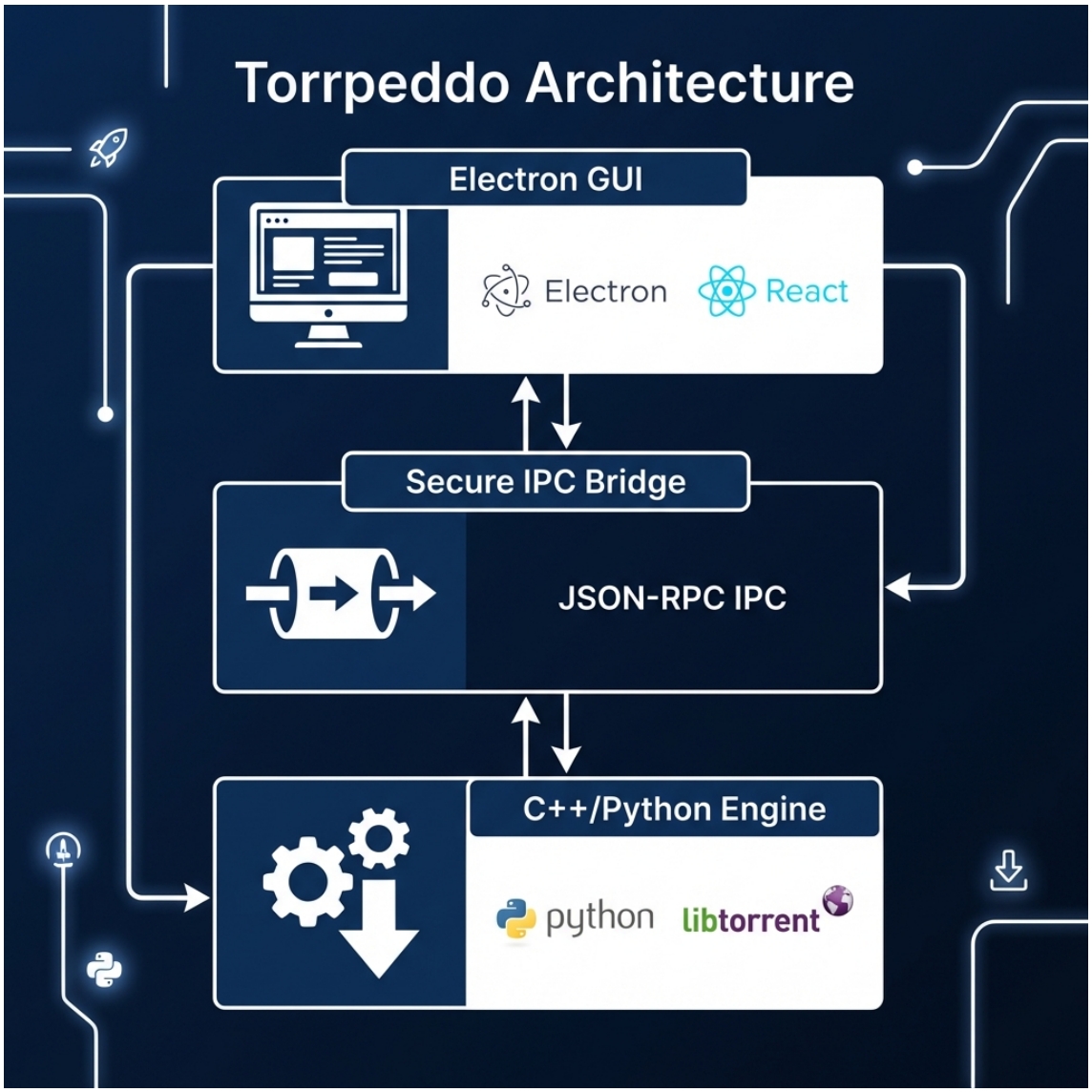
TORRPEDDO PROJECT BOOK



Executive Summary

Architectural Deep Dive

Torrpeddo follows a decoupled architectural pattern, separating the presentation layer from the core logic and network engine. This is achieved through three primary layers:

1. Frontend: Electron Framework

Benefits for Torrpeddo: - Visual Excellence: Leveraging the full power of modern CSS and web components to create a "WOW" factor UI that feels premium. - Cross-Platform Compatibility: A single codebase provides a consistent experience across Linux, Windows, and macOS. - Native Experience: Provides access to native OS features like file dialogs, tray notifications, and filesystem integration.

2. The Bridge: IPC (Inter-Process Communication)

Implementation: Secure JSON-RPC Communication is handled via a JSON-RPC protocol over stdin/stdout channels. - The Electron process spawns a dedicated Python child process. - Commands (e.g., `add_magnet`, `get_status`) are serialized into JSON strings and sent to the Python process. - The Python process executes the logic and returns a structured JSON response.

Why this approach? - Decoupling: The engine can be updated, debugged, or even replaced without touching the UI. - Security: The backend runs as a separate process, providing a layer of isolation. - Performance: High-speed communication with minimal overhead compared to HTTP-based local servers.

3. Backend Engine: Python & libtorrent

Multi-threaded Performance: - Engine Level: The `libtorrent` 2.0+ engine utilizes an internal thread pool for disk I/O, network polling, and piece validation. This allows for parallel processing of multiple torrent fragments simultaneously. - Manager Level: The Python `TorrentManager` utilizes daemon threads to handle non-blocking torrent additions, ensuring that the IPC bridge never stalls while metadata is being fetched from the DHT or trackers.

Development Process & Methodology
