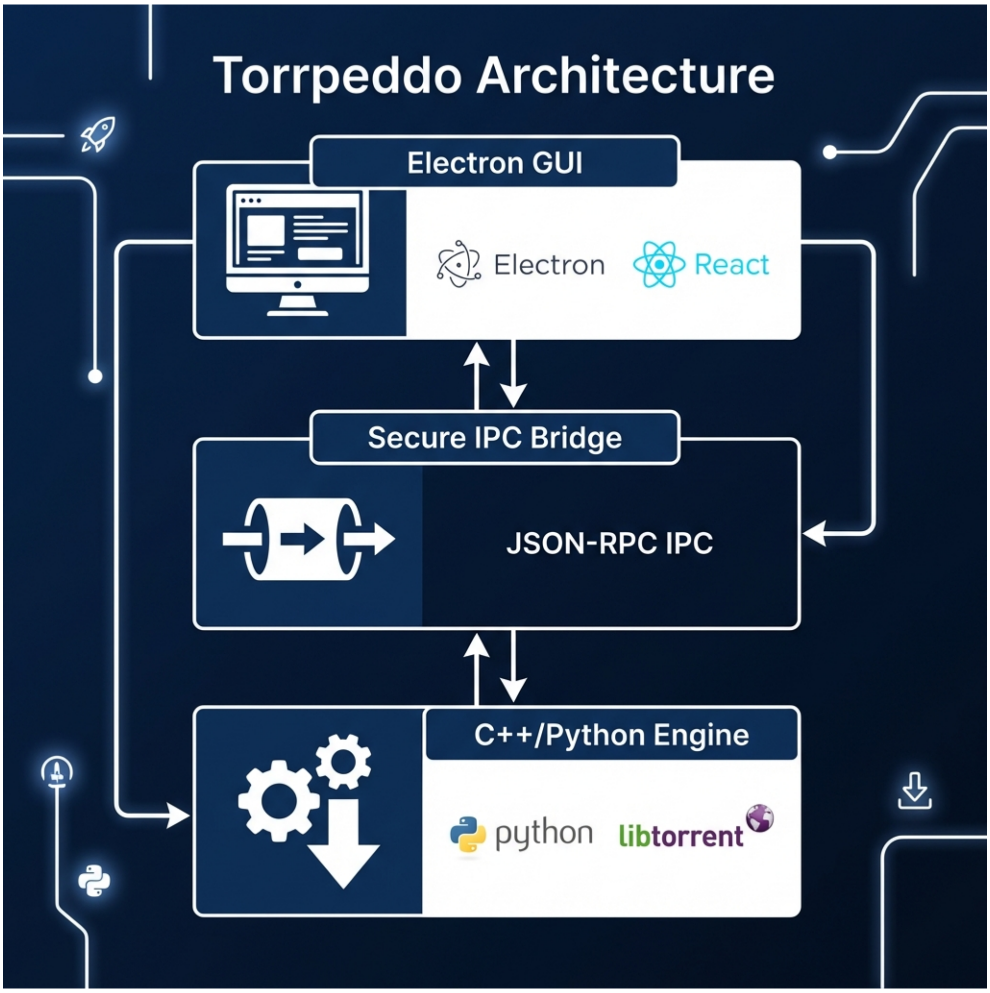
TORRPEDDO PROJECT BOOK



Executive Summary

Torrpeddo is an industrial-grade, premium torrent client designed for the modern desktop. Built primarily with Python and the Electron framework, Torrpeddo leverages the powerful libtorrent suite to offer a seamless, high-performance experience that bridges the gap between complex network protocols and professional user interfaces.

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

Overview: What is Electron?

Electron is an open-source framework developed by GitHub that allows developers to build cross-platform desktop applications using web technologies: HTML, CSS, and JavaScript. It combines the Chromium rendering engine (for the UI) and the Node.js runtime (for system-level access).

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)

Concept: What is IPC?

IPC, or Inter-Process Communication, is a mechanism that allows different processes to share data and coordinate actions. In Torrpeddo, we use a custom IPC bridge to connect the Electron frontend with the Python backend.

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.

Advantages of the IPC Bridge

- Decoupling: The engine can be updated, debugged, or even replaced without touching the UI.

- Security: The backen

isolation.

3. Backend Engine: Python & libtorrent

The Core: libtorrent with Python Bindings

At the heart of Torrpeddo is libtorrent, a feature-complete BitTorrent implementation. While the underlying engine is implemented in high-performance C++, Torrpeddo utilizes the official Python bindings for rapid development and seamless integration with the bridge logic.

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

handle non-blocking to

stalls while metadata is

Management & Operations

Torrpeddo provides robust tools for managing the lifecycle of downloads and maintaining disk health.

1. Torrent Cancellation

Users can stop an active download at any time by clicking the "Cancel" button. This operation: - Safely removes the torrent from the active `libtorrent` session. - Stops all network and disk I/O associated with the task. - Transitions the torrent to a "Cancelled" state in the UI for further action.

2. Disk Cleanup & File Deletion

For cancelled torrents, Torrpeddo offers a dedicated "Delete Files" operation. - Target Removal: Deletes the specific files or directories associated with the cancelled torrent. - State Sync: Once files are deleted, the torrent entry is removed from the session tracking, keeping the UI clean.

Development Process & Methodology

The Torrpeddo project followed a "Platform-First" methodology:

Phase 1: Language Choice

Python was selected for its extensive library support and ease of integration with libtorrent and IPC protocols.

Phase 2: Engine Validation

Rigorous testing of libtorrent benchmarks to ensure maximum throughput on varied hardware.

Phase 3: Bridge Optimization

Implementation of non-blocking I/O in the IPC bridge to prevent UI "micro-stutters".

Phase 4: Packaging & Distribution

Integration of electron-builder and PyInstaller to create unified, single-binary distributors for end-users.
