yt-dlp GUI Desktop App - Product Requirements Document

Document Version: 1.0
Date: December 20, 2025
Status: Approved for Development
Target Launch: ASAP (MVP within 1 week)

TLDR

A lightweight desktop application that replaces yt-dlp's command-line interface with a simple button-based GUI. Users paste a video URL, select download format (video/audio), choose quality level, and click download. The app handles file organization automatically, provides real-time download progress, and integrates yt-dlp's proven extraction engine without requiring terminal knowledge. Target: macOS/Windows/Linux desktop users downloading YouTube content casually.

---

1. Product Overview and Purpose

Problem Statement

yt-dlp is a powerful command-line tool that enables users to download videos and audio from YouTube and other platforms. However, the command-line interface creates a friction point for non-technical users:

- Requires terminal/command prompt knowledge
- Complex option flags (`-f`, `-S`, `-x`) for basic tasks
- Error messages without clear guidance
- No visual progress feedback
- Manual directory management

Business Objectives

- **Accessibility:** Lower the barrier to entry for casual YouTube downloaders
- **Simplification:** Abstract away yt-dlp's command-line complexity
- **Efficiency:** Enable personal video/audio downloads in under 30 seconds
- **Reliability:** Leverage yt-dlp's battle-tested extraction engine without reinventing

Market Context

Desktop video downloaders exist (4K Video Downloader, MediaHuman, WinX), but most require payment or are outdated. yt-dlp is free, actively maintained, and supports 1000+ sites. A simple GUI wrapper capitalizes on this strength without duplicating extraction logic.

Value Proposition

One-Click Downloads: Paste URL → Select Format → Download. No terminal, no flags, no frustration. Powered by yt-dlp's proven reliability.

---

2. Target Audience and User Personas

Primary User Segment: Casual YouTube Downloader

Demographics:

- Age: 16–60
- Technical skill: Basic (uses browser, file explorer, minimal CLI experience)
- Primary OS: Windows, macOS, Linux desktop
- Primary use case: Download YouTube videos and extract audio for personal use

User Pain Points:

- Don't understand `-f best` or format codes
- Unsure how to merge video + audio files
- No visibility into download progress
- Fear of malware from browser extensions or sketchy websites
- Want organized, automatic file placement

User Motivations:

- Save videos for offline viewing
- Extract audio for music playlists
- Archive content before deletion
- Download for personal projects (non-commercial)

User Journey Context:

1. User finds video on YouTube
2. Copies video URL
3. Opens downloader app
4. Pastes URL
5. Selects format (video/audio) and quality
6. Clicks "Download"
7. File appears in organized folder

---

3. Success Metrics and Release Criteria

Key Performance Indicators

Metric	Target	Rationale
Time to download (start-to-finish)	< 1 min for average 10MB file	Single-user workflow should be friction-free
UI response time	< 500ms from button click to action	Perceived responsiveness
Application startup time	< 2 seconds	Quick launch from desktop
Successful download rate	95%+	Reliability is non-negotiable
Error clarity	100% of errors display user-friendly message	Reduce confusion and support burden

Functionality Criteria (Must-Have for Launch)

- ✅ Paste YouTube URL into input field
- ✅ Download as MP4 video (best quality available)
- ✅ Download as MP3 audio (best quality available)
- ✅ Display real-time download progress (percentage, speed, ETA)
- ✅ Save files to user-selected folder (with sensible default: `~/Downloads/yt-dlp/`)
- ✅ Handle common errors (invalid URL, network timeout, format unavailable) with clear messages
- ✅ Display downloaded file size and duration before/after download
- ✅ One-click "Open Folder" to view downloaded file

Usability Standards

- Simple enough for non-technical users to operate without instructions
- Button labels must be self-explanatory ("Download as Video", "Download as Audio")
- Progress bar provides continuous feedback
- All critical information visible on one screen (no modal dialogs unless necessary)

Reliability Requirements

- Application must not crash on invalid input (malformed URLs, network errors)
- Resumed/partial downloads must be handled gracefully
- Dependency on yt-dlp binary must be verified at startup with clear installation guidance if missing

Performance Benchmarks

- Metadata fetch (title, duration, thumbnail): < 3 seconds
- Format list loading: < 2 seconds
- Concurrent downloads: Support 1 download at a time (no multi-threading for MVP)

Supportability Standards

- Clear error messages with actionable solutions
- Log file saved locally for debugging
- Application state persists (last folder, last URL, download history)

---

4. Features and Functionality

MoSCoW Prioritization

MUST-HAVE (Critical for Launch)

1. URL Input & Validation

- **Feature Description:** Text input field for YouTube video URL
- **User Story:** As a casual user, I want to paste a YouTube URL so I can download the video without typing complex commands
- **Acceptance Criteria:**
  - Input field accepts full URLs (`https://www.youtube.com/watch?v=...`) and short URLs (`youtube.com/watch?v=...`)
  - Input is validated; invalid URLs show error message immediately
  - Clearing the field resets the form
  - **Priority Justification:** Core entry point; without this, the app has no purpose

2. Format Selection (Video vs. Audio)

- **Feature Description:** Two radio buttons/toggle: "Download as Video (MP4)" or "Download as Audio (MP3)"
- **User Story:** As a user, I want to choose between downloading a full video or just the audio so I can use the content in different ways
- **Acceptance Criteria:**
  - Default is "Video"
  - Selecting "Audio" changes expected output format to MP3
  - Selection persists if user switches URLs
- **Priority Justification:** Covers 80% of use cases (full video or audio extraction)

3. Quality Selection

- **Feature Description:** Dropdown with quality options for video ("Best", "720p", "480p", "360p") and auto-format for audio ("Best Quality")
- **User Story:** As a user, I want to choose download quality so I can balance file size with visual fidelity
- **Acceptance Criteria:**
  - Video options: Best, 720p, 480p, 360p (disabled if not available)
  - Audio: Single "Best Quality" option (no choice needed—always best audio + best encoding)

    - Unavailable options are greyed out after metadata fetch
- **Priority Justification:** Gives power users control without overwhelming casual users

4. Download Progress Display

- **Feature Description:** Real-time progress bar showing download percentage, current speed (MB/s), and estimated time remaining
- **User Story:** As a user, I want to see download progress so I know the app is working and when the download will finish
- **Acceptance Criteria:**
    - Progress bar updates every 100ms
    - Display: "Downloading... 45% (2.3 MB/s, ~12s remaining)"
    - Final line: "Download complete: filename.mp4 (48.5 MB)"
    - Progress information persists in a log area
- **Priority Justification:** Without feedback, users think the app is frozen

5. Save Location Management

- **Feature Description:** Folder selection UI with a default path (`~/Downloads/yt-dlp/`)
- **User Story:** As a user, I want to choose where files are saved so I can organize downloads in my preferred location
- **Acceptance Criteria:**
    - Default folder shown: `/Users/username/Downloads/yt-dlp/` (macOS), `C:\Users\username\Downloads\yt-dlp\` (Windows)
    - "Browse" button opens native file picker
    - Selected path persists across sessions
    - App auto-creates directory if it doesn't exist
- **Priority Justification:** Users need to know where files end up

6. One-Click Download

- **Feature Description:** Primary "Download" button that triggers the download with selected options
- **User Story:** As a user, I want to click one button and start the download so I don't have to memorize complex commands
- **Acceptance Criteria:**
    - Button disabled until valid URL is entered
    - Button text changes to "Downloading..." during operation (with spinner)
    - Button re-enabled when download completes
- **Priority Justification:** Core interaction

7. Error Handling & User Feedback

- **Feature Description:** Clear error messages for common issues (invalid URL, network timeout, format unavailable, yt-dlp not installed)
- **User Story:** As a user, I want to understand why a download failed so I can fix the problem
- **Acceptance Criteria:**
    - Error messages are plain English, not technical jargon
    - Example: "This URL doesn't look like a YouTube link. Please check and try again." (not: "Invalid regex match on extractor pattern")
    - Each error suggests a solution
    - Errors appear in a red alert box, not in console

- **Priority Justification:** Prevents user frustration and support requests

8. File Organization

- **Feature Description:** Downloaded files are automatically named with video title, saved to the selected folder
- **User Story:** As a user, I want files named clearly so I can find them later
- **Acceptance Criteria:**
  - Filename format: `{video_title}.{ext}` (e.g., `My Awesome Video.mp4`)
  - Special characters sanitized automatically
  - Duplicate filenames handled (e.g., `My Awesome Video (1).mp4`)
- **Priority Justification:** Basic file management

SHOULD-HAVE (Important, Post-MVP)

1. Metadata Preview

- Display thumbnail, video title, duration, and channel before downloading
- Helps users confirm they're downloading the correct video

2. Download History

- Sidebar showing last 10 downloaded videos (title, date, file size)
- Quick re-download button for repeat downloads

3. Subtitle Support

- Checkbox: "Download subtitles" → downloads SRT file alongside video
- Covers educational/international use cases

4. Playlist Support

- Option: "Download entire playlist" → downloads all videos to separate files
- Advanced feature for power users

COULD-HAVE (Nice-to-Have)

- Dark mode / light mode toggle
- Settings panel (yt-dlp binary path, preferred default folder, language)
- Drag-and-drop URL support
- System tray minimization
- Batch URL import (paste multiple URLs, download in queue)

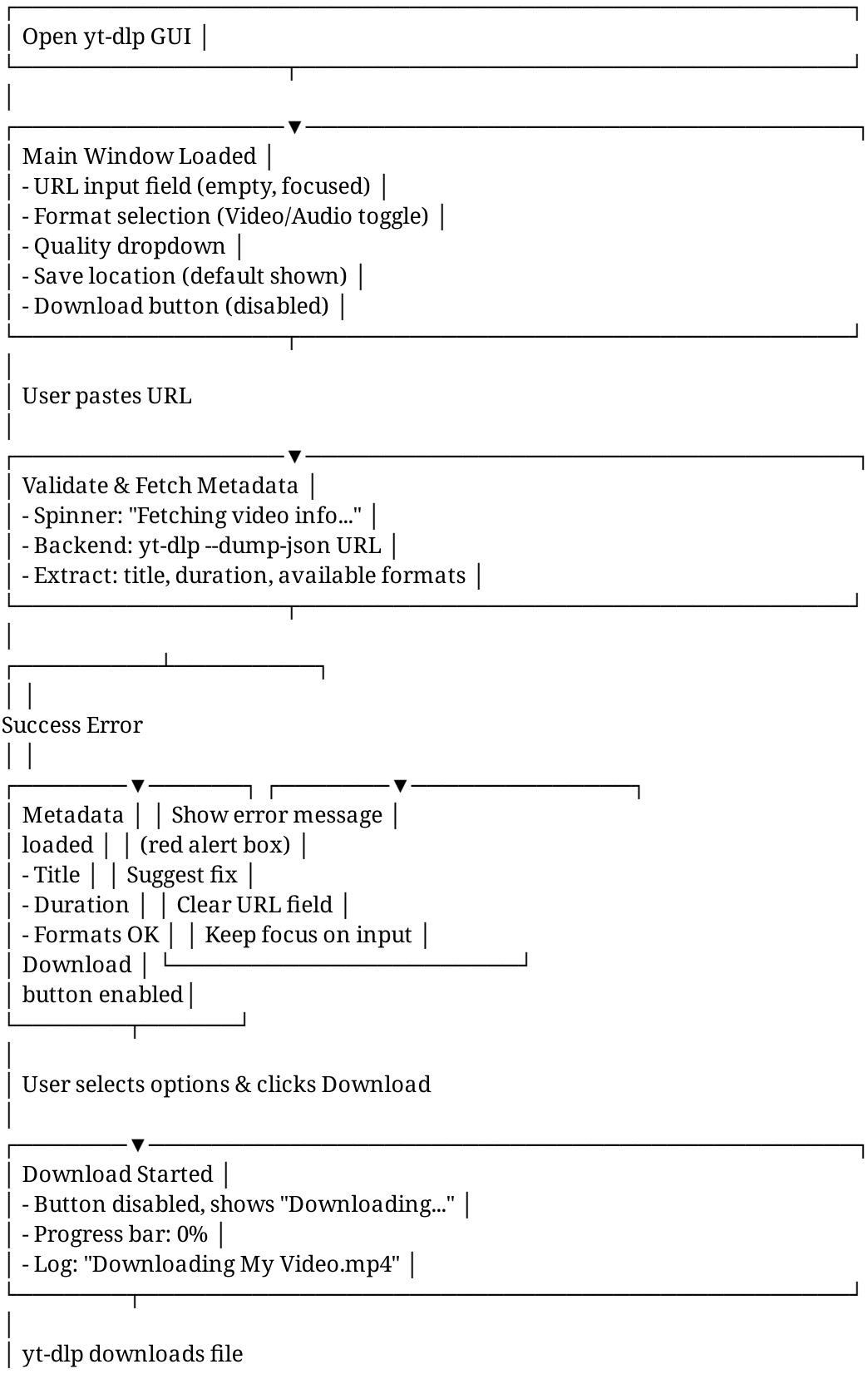
WON'T-HAVE (Explicitly Out of Scope)

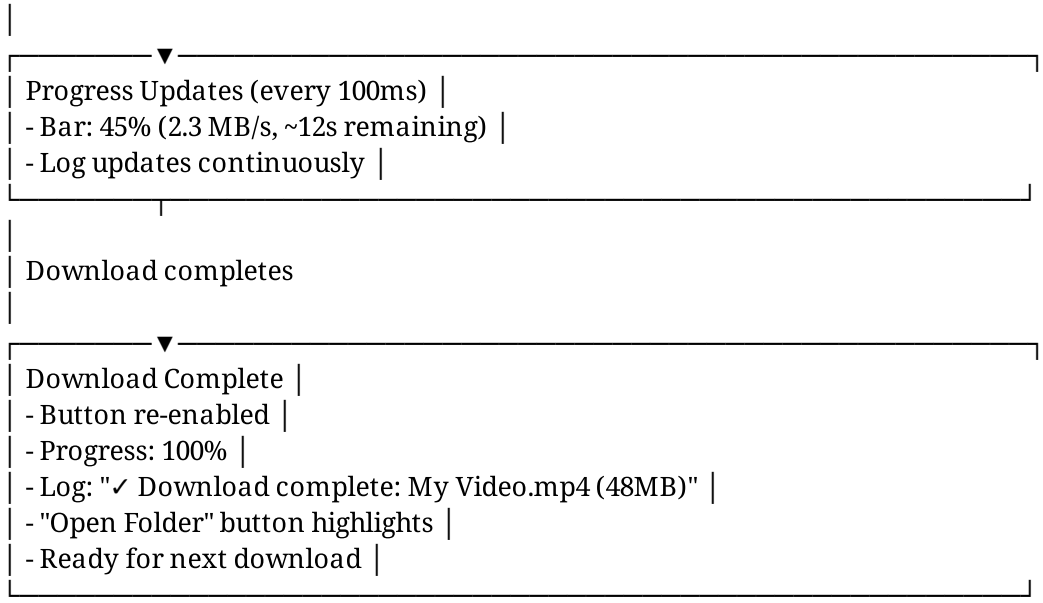
- Multi-threaded concurrent downloads (too complex for MVP)
- Advanced format customization (e.g., custom codec selection)
- Video trimming/editing (use FFmpeg separately)
- Support for non-YouTube sites (focus on YouTube first; yt-dlp supports 1000+ but we simplify UX for YouTube)
- Monetization (ads, paid tiers)

---

5. User Experience and Design

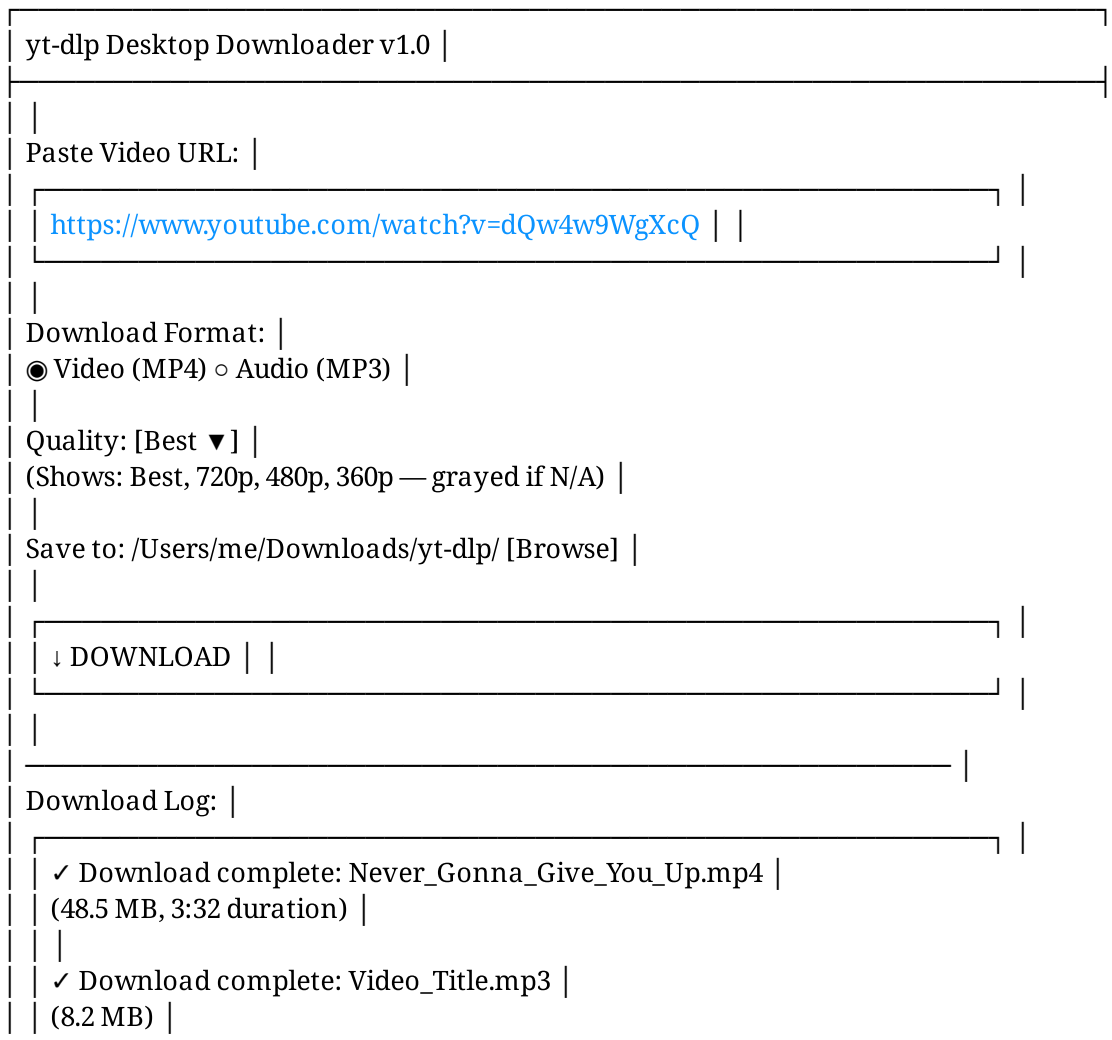
User Flow Diagram

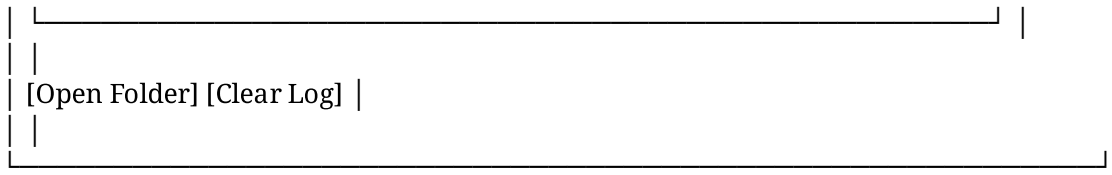




Interface Requirements

Layout (Single Window):





Design Principles

- **Minimalism:** One screen, no tabbed interface. All critical controls visible at once.
- **Clarity:** Button labels and field names explain purpose without jargon.
- **Feedback:** Every action (URL validation, metadata fetch, download progress) provides visual feedback.
- **Accessibility:** Keyboard navigation (Tab, Enter), readable contrast, large touch targets (buttons ≥44px).
- **Progressive Disclosure:** Advanced options hidden; defaults work for 80% of use cases.

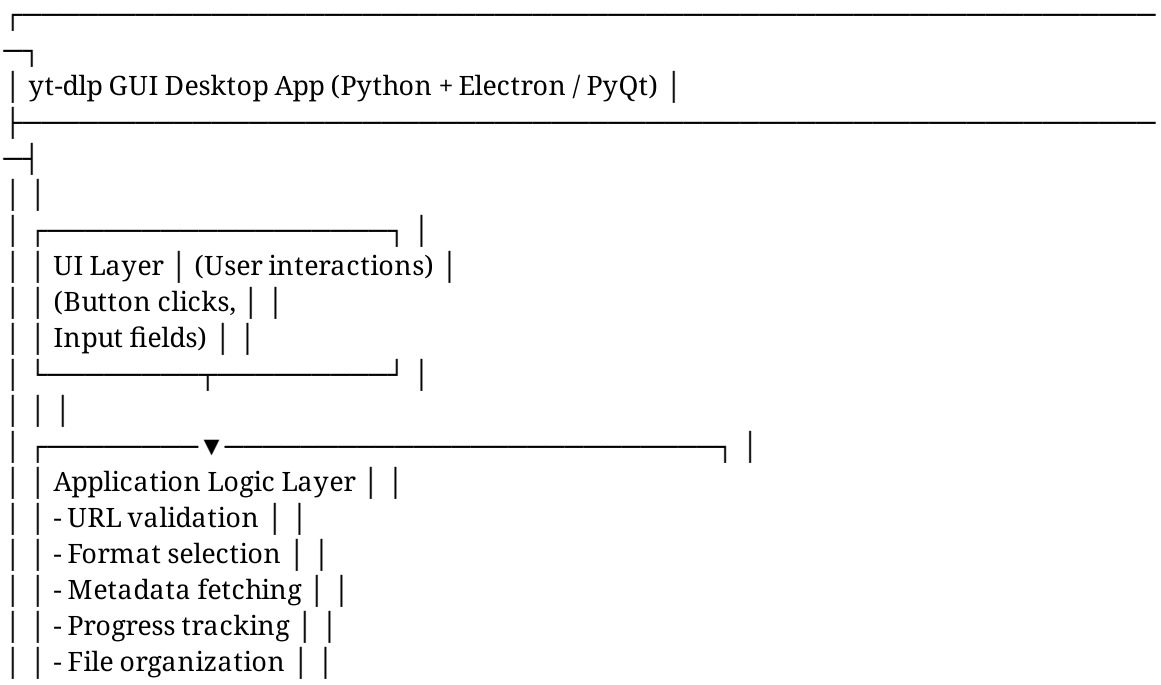
Accessibility Requirements

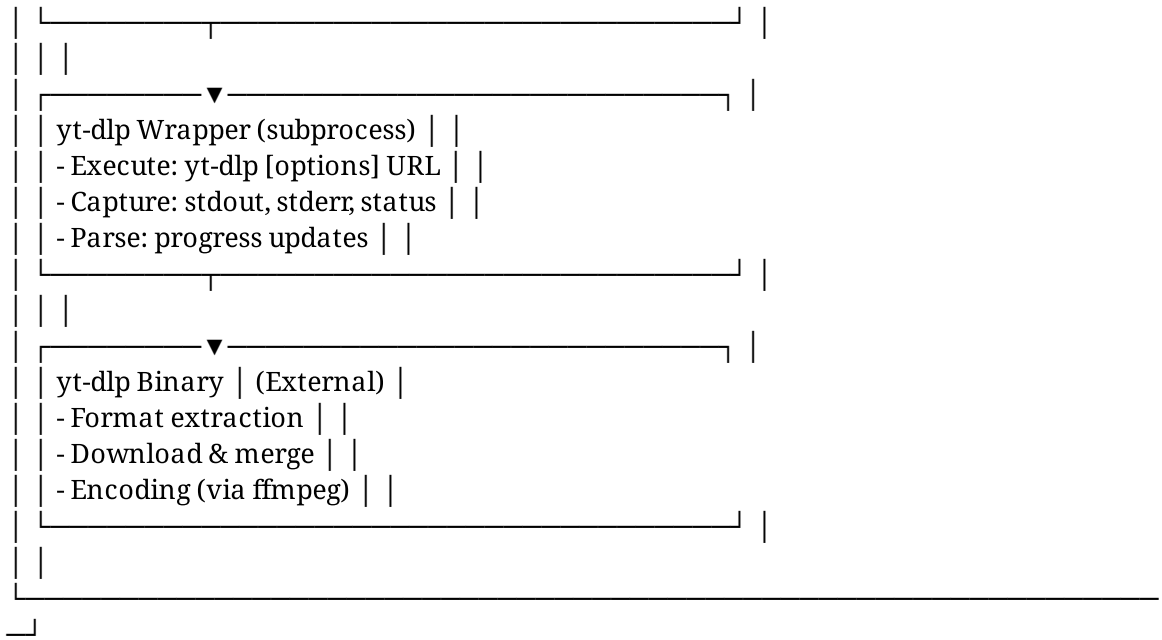
- **WCAG 2.1 Level AA compliance:**
  - Color contrast: 4.5:1 for text, 3:1 for graphics
  - Keyboard navigation: All buttons/inputs reachable via Tab
  - Focus indicators: Visible outline on focused elements
  - Text alternatives: Buttons labeled (no icon-only buttons)
- **Screen reader support:** ARIA labels on inputs ("URL Input", "Format Selection")
- **Error messaging:** Errors announced to screen readers, visible as text (not color alone)

---

6. Technical Requirements

System Architecture





Technology Stack (Recommendation)

Framework: PyQt6 (Python + Qt6)

- **Why:** Native look-and-feel on macOS/Windows/Linux, single codebase, quick iteration
- **Alternative:** Electron (JavaScript/Node) if you prefer web technologies

Language: Python 3.9+

- **Why:** yt-dlp is Python-based; easy integration; cross-platform

Bundling: PyInstaller (Python) or Electron Forge (Electron)

- Produces standalone executable (no Python installation required)

Backend Process: `subprocess` (Python) to invoke yt-dlp binary

API Requirements

External Dependencies:

- **yt-dlp binary** (must be present on system or bundled)
- **ffmpeg** (optional but strongly recommended for audio encoding; yt-dlp can work without it, but with limitations)

Internal API (Application Logic):

class DownloadManager:
def validate_url(url: str) -> bool
def fetch_metadata(url: str) -> dict # Returns: title, duration, available_formats
def start_download(url: str, format: str, quality: str, output_path: str) -> AsyncDownloadTask
def get_progress() -> dict # Returns: percentage, speed_mbps, eta_seconds
def cancel_download() -> None

Database Requirements

None for MVP. Persist state using simple JSON file:

```
{
"last_output_path": "/Users/me/Downloads/yt-dlp/",
"last_format": "video",
"last_quality": "best",
"download_history": [
{
"url": "https://youtube.com/watch?v=xyz",
"title": "My Video",
"date": "2025-12-20T14:23:00Z",
"file_size_mb": 48.5
}
]
}
```

Security Requirements

- **No network exposure:** App is local-only (no API backend, no cloud sync)
- **No credential storage:** Don't store YouTube cookies unless explicitly requested (post-MVP)
- **Input validation:** Sanitize file paths to prevent directory traversal
- **Process safety:** Run yt-dlp in subprocess with restricted environment (no shell injection)
- **File permissions:** Set reasonable defaults (user-readable only)

Scalability Needs

MVP: Single download at a time. No concurrent operations.
Future (Post-MVP): Queue system for multiple URLs.

Performance Requirements

Operation	Target	Notes
Startup	< 2s	App launches, ready to download
Metadata fetch	< 3s	Title, duration, formats loaded
Progress update	< 500ms latency	UI remains responsive
Download start	< 1s after clicking Download	Immediate feedback

Compliance Obligations

- **Copyright:** Respect YouTube's Terms of Service (educational/personal use disclaimer in app)
- **Licensing:** Use yt-dlp (licensed under UNLICENSE); include attribution
- **GDPR:** No personal data collection (app is local-only)

---

7. Dependencies and Constraints

Internal Dependencies

- **Python 3.9+ runtime:** Required to execute the app
- **Qt6 libraries:** UI framework (bundled in PyInstaller executable)
- **Operating system:** macOS 10.14+, Windows 8+, Ubuntu 18.04+ (or equivalent Linux)

External Dependencies

Critical:

- **yt-dlp binary:** Must be installed or bundled
  - Check for presence at startup
  - If missing, display clear installation instructions: "Download yt-dlp from [link]"

Strongly Recommended:

- **ffmpeg & ffprobe:** For audio encoding and video/audio merging
  - If missing, app still works but with limitations (e.g., audio-only downloads may fail)
  - Check and warn user if missing, but don't block functionality

Technical Constraints

- **Single-threaded UI:** Download runs in background thread to prevent UI freezing
- **No external servers:** App must work offline after yt-dlp is installed
- **Platform differences:** Handle Windows path separators (backslash) vs. macOS/Linux (forward slash)

Business Constraints

- **Timeline:** MVP in 1 week → simplified scope (no playlist, no subtitle support, no advanced options)
- **Resources:** Single developer → focus on core workflow
- **Maintenance:** Lightweight codebase to reduce ongoing support burden

Regulatory Constraints

- **Fair use disclaimer:** Include note that app is for personal/educational use (respect copyright)
- **Terms of Service:** Users responsible for complying with YouTube ToS
- **Data privacy:** No telemetry, analytics, or user tracking

---

8. Assumptions and Risks

Key Assumptions

1. **yt-dlp binary is available:** User has installed yt-dlp or we bundle it
2. **User has network access:** Downloads require internet connectivity
3. **YouTube URLs are stable:** Video titles/availability don't change during download
4. **ffmpeg is optional:** App can work without ffmpeg for basic video downloads
5. **Single downloads at a time:** No need for queue management in MVP
6. **Users trust yt-dlp:** They don't need education on what yt-dlp is (it's in the app name)

Identified Risks

Risk	Likelihood	Impact	Mitigation
YouTube changes API/blocks extraction	Medium	High	Monitor yt-dlp releases; display note that app depends on yt-dlp (users should check for updates)
User downloads copyrighted content	High	Medium	Display disclaimer: "For personal use only. Respect copyright." Don't block downloads (fair use is ambiguous)
yt-dlp not installed	High	High	Check at startup; provide clear download/install link
Network timeout during download	Medium	Medium	Implement retry logic (3 retries, exponential backoff); show clear error message
Invalid URL or video unavailable	High	Low	Validate URL format; fetch metadata to confirm availability before download
File system permission error	Low	High	Test write permissions to output path; show clear error if can't write
ffmpeg missing	Medium	Medium	Warn user but allow video-only downloads (may not merge separate video+audio streams)
Long filenames cause issues	Low	Low	Truncate/sanitize filenames to <200 chars, remove special characters
Disk space full	Low	High	Check available space before download; show warning if < 100MB free

Risk Mitigation Strategies

1. **Monitor yt-dlp project:** Set up notifications for major releases; test compatibility
2. **Fair use notice:** Display in settings: "This app is designed for downloading personal/educational content. You are responsible for ensuring your use complies with copyright law."
3. **Dependency checking:** On startup, verify yt-dlp binary exists and is executable; provide install guide if missing
4. **Error handling:** Catch and display all exceptions as user-friendly messages
5. **Network resilience:** Implement retry logic with exponential backoff for network requests
6. **Version pinning:** Bundle specific yt-dlp version; alert users when updates are available

Open Questions

1. Should we bundle yt-dlp in the executable, or ask users to install separately?
    - **Decision pending:** Bundling increases file size (~50MB); separate install adds setup step
    - **Recommendation:** Separate install for MVP (lighter executable), check at startup
2. Should we support sites other than YouTube initially?
    - **Decision pending:** yt-dlp supports 1000+ sites, but UX should stay YouTube-focused
    - **Recommendation:** WON'T-HAVE for MVP; hardcode YouTube-specific defaults
3. How do we handle user authentication (for age-restricted/private videos)?
    - **Decision pending:** Cookies from browser? Manual cookie input?
    - **Recommendation:** WON'T-HAVE for MVP; users must download public videos

---

9. Timeline and Milestones

High-Level Roadmap

Phase	Duration	Key Deliverables
MVP Development	1 week	UI shell, URL input, format selection, basic download, error handling
Testing & Bug Fixes	2-3 days	Cross-platform testing (macOS, Windows, Linux), edge case handling
Release	1 day	Bundle executable, publish to GitHub releases, documentation
Post-MVP (v1.1)	2 weeks	Download history, metadata preview, subtitle support
v1.2	4 weeks	Playlist support, queue system, settings panel

Key Milestones

Week 1 (MVP):

- Day 1-2: UI layout (PyQt6), basic styling
- Day 2-3: yt-dlp integration (subprocess, metadata fetching)
- Day 3-4: Download logic, progress tracking
- Day 4-5: Error handling, file organization
- Day 5: Testing, bug fixes, first release candidate

Post-MVP (Optional, Timeline TBD):

- Download history sidebar
- Subtitle checkbox
- Playlist support

Release Strategy

MVP (v0.1.0):

- Single executable for Windows, macOS, Linux
- Published to GitHub Releases
- Announcement: "yt-dlp GUI—Download YouTube videos with one click"

v1.0:

- Stabilized after user feedback
- Additional features added based on demand

Ongoing:

- Monitor yt-dlp releases; test compatibility

- Bug fixes and performance improvements

Feature Flags

Post-MVP consideration:

- `FEATURE_HISTORY` - Enable/disable download history sidebar
- `FEATURE_SUBTITLES` - Enable/disable subtitle download option
- `FEATURE_PLAYLISTS` - Enable/disable playlist download

---

10. Stakeholder Review and Approval

Stakeholder List

Stakeholder	Role	Approval Required
Developer (You)	Builder	Yes (decision-maker)
Intended Users	Casual YouTube downloaders	Feedback post-MVP
yt-dlp Maintainers	Upstream dependency	No formal approval (open-source)

Review Process

1. **Development phase:** Self-review during coding
2. **Testing phase:** Manual testing on macOS, Windows, Linux (if possible; can focus on primary OS)
3. **Release:** Publish to GitHub, gather feedback via Issues
4. **Iteration:** Address user feedback in v1.1, v1.2

Approval Requirements

Before Release:

- ✅ MVP features implemented and tested
- ✅ No critical bugs blocking basic download workflow
- ✅ Error messages are user-friendly
- ✅ Executable bundles for primary OS (Windows or macOS)

Internal (Self-Approval):
This PRD is approved for development. You are authorized to proceed with implementation.

Appendix: Glossary

Term	Definition
yt-dlp	A feature-rich command-line audio/video downloader supporting 1000+ sites
ffmpeg	A multimedia processing library used to merge video/audio streams and encode
Format	Video container (MP4, WebM) or audio codec (MP3, AAC)
Quality	Resolution/bitrate selection (Best, 720p, 480p, etc.)
Metadata	Information about a video: title, duration, available formats
Subprocess	A spawned child process (used to invoke yt-dlp from the GUI)
MVP	Minimum Viable Product—core features only, no polish

Document Approval: ✅ Ready for Development
Next Step: Begin UI design and yt-dlp integration testing