ChatGPT

# Designing a ComfyUI Node for Downloads with Gallery-dl

## Overview of Gallery-dl's Capabilities

**Gallery-dl** is a versatile open-source downloader for images and videos from numerous sites (over 100 supported sites ranging from art communities to social media) [1]. It can fetch entire galleries, user profiles, or collections, handling both images and multimedia. The tool offers extensive **configuration options** and **powerful file-naming capabilities** out of the box [1]. By integrating gallery-dl into a ComfyUI node, you can harness these features within a workflow. Key benefits include support for site authentication (accounts, cookies, API keys), automated naming and directory structuring, and even filtering of content before download. The node will essentially act as a wrapper around gallery-dl's Python library/CLI, enabling easy batch downloads as part of your ComfyUI pipeline.

## Node Input: Multiple URLs and List Handling

The node should accept **multiple URLs** as input, since gallery-dl is designed to handle a list of target URLs in one go [2]. This can be implemented as a multi-line text field (each line containing a URL) or alternatively a file input (e.g. a `.txt` file with one URL per line). The node will read all provided URLs into a list for processing. Gallery-dl will then attempt to download from each URL in turn – and it conveniently ignores any URLs that it does not recognize (no extractor available) [3], so users can paste mixed content without errors.

For convenience, the node could also support **remote URL lists** using gallery-dl's special syntax. For example, gallery-dl allows prefixing a URL with `r:` to indicate that it's a resource containing links (it will fetch that resource and parse out links) [3]. This means if a user provides a link to a Pastebin or similar containing a batch of URLs, gallery-dl can handle it internally. Additionally, if a URL's domain is non-standard or unrecognized, the node can allow prefixing it with a specific extractor name to force the correct handler (e.g. `"tumblr:<URL>"` to treat a custom domain as Tumblr) [4]. Overall, the node's input system should be flexible: users can paste many links or provide a file of links to queue up a bulk download task.

## Output Path and File Saving Strategy

This node's primary function is to **save files to disk** (not to pass images in-memory through the pipeline). Therefore, an important input is the **output directory path** where downloaded images/videos will be stored. By default, gallery-dl would use a folder like `./gallery-dl/` in the current directory [5], but we want the user to be able to choose a path. The node can accomplish this by configuring gallery-dl's `base-directory` option to the user-specified path [5]. In practice, this could be done via the gallery-dl config or by passing an override option (e.g. `-o base-directory="/my/downloads/path"` in a subprocess call).

Gallery-dl will create subfolders and filenames according to its rules or any templates set in the configuration. By leveraging its **filename templating**, users can automatically organize downloads. For instance, one could define a template to include the site name, gallery title, or author in the folder structure [6] [7]. The ComfyUI node could expose a simplified option to choose a naming preset, but it might suffice to rely on the gallery-dl config for advanced patterns. By default, gallery-dl already has sensible naming for many sites (often separating content by site or user). The **file naming and directory structure** can always be customized in the config (e.g. using keys like `extractor.*.directory` and `extractor.*.filename` to define format strings) [8] [9]. The key point is that all downloaded files will be saved under the chosen base directory, neatly organized, and **no direct image output** will be sent through the node (avoiding heavy memory usage for large batches). Instead, the node could optionally output a list of file paths or a summary (e.g. number of files downloaded) for reference, but the images themselves remain on disk for later use.

## Authentication: Config File, Passwords, and Cookies

One of the strengths of gallery-dl is handling **authenticated downloads** for sites that require login. Many extractors support or even require credentials – for example, sites like nijie require a login, and others like Twitter, Danbooru, e621, etc. allow it for enhanced access [10]. The ComfyUI node should leverage gallery-dl's **configuration file** to supply credentials rather than prompting each time. In the gallery-dl config (a JSON file), users can specify usernames and passwords for supported sites under the `"extractor"` section. For instance, to enable Twitter login, the config would include:

```
{
    "extractor": {
        "twitter": {
            "username": "<your username>",
            "password": "<your password>"
        }
    }
}
```

This way, the node can download from these sites as an authenticated user (where needed) without manual login steps [11]. The node should be designed to **load the user's gallery-dl config** (which by default is looked for in standard locations like `~/.config/gallery-dl/config.json` on Linux, etc. [12] [13]). You could either instruct users to place their credentials in the default config or allow an input to specify a custom config file path. Using the config file is advantageous for security (no plaintext passwords in the UI) and convenience.

Beyond basic username/password, many modern sites require session cookies or OAuth tokens. Gallery-dl addresses this by letting you reuse your **browser session cookies**. The node can expose this feature so users don't have to manually copy cookies. In the config, the `"cookies"` option for an extractor can point to either a cookies.txt file or even directly to a browser name, and gallery-dl will pull the cookies from that browser's logged-in session [14]. For example, to use your Firefox cookies for Twitter, you can set in config:

```
"extractor": {
    "twitter": {
        "cookies": ["firefox"]
    }
}
```

This tells gallery-dl to automatically extract the necessary cookies from Firefox's profile for Twitter [15] [16]. Supported browsers include Chromium-based browsers, Firefox, and Safari [14]. The ComfyUI node might provide a dropdown to select a browser for cookies (e.g. "Use cookies from: None/Firefox/Chrome/..."), which internally would invoke the equivalent of `--cookies-from-browser <browser>` when running gallery-dl. (Under the hood, gallery-dl can do this on its own if the config is set, or via a command-line flag [17].) Keep in mind that to use browser cookie extraction, certain optional dependencies may be needed – for example, on Linux, installing the **SecretStorage** Python package enables gallery-dl to access browser keyrings for decryption of cookies [18].

For some sites, **OAuth** is the required auth method (e.g. Pixiv, and optionally DeviantArt, Flickr, Reddit, etc.) [19]. In these cases, the user must first run an OAuth flow (gallery-dl provides a command like `gallery-dl oauth:<site>` which guides the user through authorizing and produces tokens) [20]. The resulting tokens would be placed in the config file. The node doesn't need to handle the OAuth handshake itself (that's a one-time setup externally), but it should be aware and use the tokens from the config. In summary, the node should **fully support gallery-dl's auth mechanisms**: whether via direct credentials, cookies, or tokens, all configured in the standard JSON config. This allows downloading content from private profiles or subscription sites (like Patreon or Pixiv Fanbox) as long as the user's credentials are provided to gallery-dl.

## Leveraging Advanced Gallery-dl Features

To make the node truly comprehensive, we should incorporate some of gallery-dl's advanced options:

- **Selective Download and Filtering:** Gallery-dl allows filtering which items to download based on metadata. The node could expose parameters for common filters. For example, you might only want certain chapters of a comic or certain resolution images. In gallery-dl CLI you can do things like `--chapter-filter "10 <= chapter < 20"` or supply options like `-o "lang=fr"` to download only a specific language version [21]. A ComfyUI node might have an optional field for a gallery-dl filter expression or specific site option (this would be advanced usage, but the backend can simply pass it to gallery-dl). By using `-o` flags, any configuration option can be overridden per run – giving power users flexibility without editing the config file for one-off tasks.

- **Video and Media Support:** Since the goal is to download images *and* videos, ensure that the node supports video downloads as well. Gallery-dl can grab videos (for example, Twitter videos, Tumblr videos, etc.) and it often delegates to **yt-dlp** (a youtube-dl fork) for complex video retrieval [22]. To fully utilize this, the environment running ComfyUI should have *yt-dlp* available (gallery-dl will use it if installed) [23]. Similarly, for certain animated image formats (like Pixiv's Ugoira), having **FFmpeg** installed allows gallery-dl to automatically convert those into videos [24]. The node should note in documentation or requirements that installing these optional tools will expand its capabilities. You might also include a node option to **skip videos** if the user only wants images – this could be done by setting appropriate extractor options (e.g. `"videos": false` for certain extractors)[25]. Otherwise, by default, the node will download all media content (images **and** videos) that gallery-dl is able to pull from the URLs.

- **Download Archive for Duplicate Skipping:** A highly useful feature for a persistent workflow is gallery-dl's **download archive** option. This feature keeps track of which files (usually by ID or URL) have already been downloaded, so they can be skipped on subsequent runs[26]. The node can integrate this by providing an "**Skip Previously Downloaded**" toggle. Under the hood, enabling this would point gallery-dl to an archive database (SQLite format) file. Gallery-dl will then record each downloaded item's ID, and if the same gallery or post is processed again later, it will **skip files already in the archive** to avoid duplicates[26]. You can use a single archive file for all sites or separate per site (the default archive path template can include the `{category}` (site name) to separate entries[27]). Implementing this might involve either setting the `"extractor.*.archive"` setting in config or passing a parameter like `-o archive="~/gallery-dl-archive.sqlite3"`. This feature is great for nodes that might be run repeatedly (for example, periodically downloading new posts from a creator without redownloading old ones). The user in ComfyUI could specify the archive file location or just accept a default. We should also handle the case where no archive is used (then every run downloads everything anew).

- **Post-Processing Hooks:** Gallery-dl supports post-processing steps, such as setting file timestamps to match upload time (the "mtime" postprocessor) or zipping downloads, etc., via config[28][29]. In a ComfyUI context, some of these might be less necessary, but it's worth noting. If, for example, the user wants all images zipped after download, one could enable the "zip" postprocessor in the config for that extractor. The node doesn't need a dedicated UI for this; users can configure it in gallery-dl's config if needed. However, the node should not interfere with such postprocessors – i.e., let them run if they are configured. This way, advanced users can extend the functionality (for instance, automatically create a CBZ archive for a comic chapter) without extra coding.

- **Error Handling and Logging:** The node should capture gallery-dl's output logs and errors. Gallery-dl prints info, warnings, and errors to the console. During node execution, these can be redirected to ComfyUI's log or a text output field. That way, if a download fails (e.g., due to network issues or a missing login), the user can see it. We could run gallery-dl in verbose mode (or even debug mode) if needed by passing `-v` or `-J` (for debug JSON output) for troubleshooting, but normally just capturing errors is sufficient. Since gallery-dl will **skip unsupported URLs** and continue[3], the node might want to inform the user if some URLs were not processed (perhaps by counting how many were skipped vs downloaded). This could be part of the node's output summary.

## Implementation Approach in ComfyUI

Implementing this node will involve calling the gallery-dl functionality from Python. There are two possible approaches:

1. **Using the gallery-dl Python API** – Since gallery-dl is available as a PyPI library, you can import and use it directly. For example, `from gallery_dl.job import DownloadJob` gives access to a job runner class. You can then create a job for each URL (or one job for a list of URLs) and run it in code.

In a simple usage, it might look like: `DownloadJob(url).run()` for each URL [30]. This approach will automatically load the default config and apply all settings. It keeps everything in-process, which is convenient for capturing outputs and handling errors in Python. You would want to initialize any global config or logging once (the library likely reads config on first use). Using the API also means you could potentially get Python objects for results (though gallery-dl mostly just writes files and logs to console, rather than returning data).

2. **Calling the CLI via subprocess** – This approach treats gallery-dl as an external command-line tool. You'd construct the command string with all required options (e.g. `gallery-dl --quiet -o base-directory="..." [maybe --cookies-from-browser] URL1 URL2 ...`) and run it with Python's subprocess module. This might be slightly simpler to implement, as it leverages exactly the CLI behavior and ensures the config file is read exactly as when run in a terminal. The subprocess stdout/stderr can be captured for logging. If taking this route, ensure that the environment running ComfyUI has `gallery-dl` installed and in PATH (or call it via Python module: `python -m gallery_dl ...`). One advantage here is that you can easily include command-line flags (like `--cookies-from-browser`) based on node inputs without delving into the internals of the library.

Both approaches are valid; using the Python API is more integrated, whereas subprocess might be more straightforward to mirror gallery-dl's behavior. In either case, the node execution will be potentially time-consuming (if downloading many files), so consider making the node run asynchronously so as not to block the UI. (ComfyUI may handle this by default for custom nodes, but it's something to be mindful of.)

**Node Input/Output Design:** You will create inputs in the node for everything outlined: a multi-line URL input (and/or a file input), an output directory chooser, perhaps booleans or dropdowns for options like "use browser cookies" (with a choice of which browser) and "skip duplicates (archive mode)", plus any other relevant toggles (like skip videos). However, you don't want to overwhelm the user with too many toggles – a good approach is to implement sensible defaults (e.g. use config for auth, download everything available, etc.) and only expose options that the user is likely to need to change frequently. The **archive option** for duplicate skipping is probably worth exposing since it changes behavior significantly (the user should know if they want incremental updates vs full re-download). The **cookies option** could be automatic (if the config has a browser specified, it will just work; otherwise, if the user checks "Use browser cookies" and selects a browser, you pass the flag accordingly). Many other fine-tuning options can remain in the gallery-dl config file (for example, API keys, specific filename templates, site-specific switches) so the node UI stays clean. Essentially, the node will bridge the user inputs to gallery-dl's options.

**Execution and Post-Execution:** When the node runs, it will invoke gallery-dl on the provided URLs. All files will download to the chosen directory (subfolders as configured). Once done, the node can output a brief summary. This might include a list of file paths downloaded (if another downstream node needs to load them, you could output this list), or simply a success message with count. If integrating with other ComfyUI nodes, you could output the directory path or list of filenames which a subsequent "Image Load" node could take to bring images into the pipeline. If the node is just for fetching data to disk (for later manual use or training), the outputs might not be used further in the graph, and that's fine too.

Finally, thorough testing with a variety of sites should be done. Try the node with a simple case (e.g. a direct image link), a gallery page, a profile page that requires login, etc., to ensure that the config is being respected and files land in the right place. With this design, the **ComfyUI Gallery-dl node** will be a powerful addition, enabling you to download large batches of images/videos from supported sites with minimal hassle, all configurable and integrated into your workflows.

## Summary of Recommended Features

- **Multi-URL Support:** Accept multiple URLs (multi-line input or file list) and process them in one run [2].
- **Custom Output Directory:** Let users set an output folder, overriding the default `./gallery-dl/` base directory [5].
- **Organized Saving:** Utilize gallery-dl's naming templates for automatic subfolders and filenames (configurable via config) [6] [9].
- **Authentication via Config:** Use gallery-dl's config file for site logins (username/password) and API keys/tokens as needed [10] [11].
- **Browser Cookies Integration:** Option to use cookies from the user's web browser session for sites that need it (e.g. Instagram, Twitter), by specifying a browser name in config or via `--cookies-from-browser` [14] [16].
- **Video Download Support:** Download videos in addition to images when available (ensure optional tools like yt-dlp and ffmpeg are installed for full support) [23] [25]. Possibly provide an option to skip videos if not desired.
- **Content Filtering Options:** (Advanced) Allow passing filter expressions or site-specific options (e.g. download only certain items) using gallery-dl's filtering capabilities [21].
- **Skip Already Downloaded:** Implement an archive database to record downloaded files and skip duplicates on subsequent runs [26] (enable via config or option).
- **Robust Error Logging:** Capture gallery-dl output/errors and surface them to the user (e.g. if a URL was unsupported or a login failed), so they know what happened.

By incorporating all the above, the ComfyUI node will cover "everything the gallery-dl tool can do," providing a comprehensive solution for bulk media downloading within the ComfyUI environment.

---

[1] [18] [23] [24] gallery-dl·PyPI
https://pypi.org/project/gallery-dl/

[2] [3] [4] [10] [11] [12] [13] [14] [15] [16] [17] [19] [20] [21] GitHub - mikf/gallery-dl: Command-line program to download image galleries and collections from several image hosting sites
https://github.com/mikf/gallery-dl

[5] [6] [7] [8] [9] [22] [25] [26] [27] [28] [29] Configuration
https://gdl-org.github.io/docs/configuration.html

[30] [QUESTION] Use gallery-dl v1.23.1 directly in python · Issue #2956 · mikf/gallery-dl · GitHub
https://github.com/mikf/gallery-dl/issues/2956