Derogations, Democratic Backsliding, and International Human Rights During the COVID-19 Pandemic

Audrey L. Comstock
Arizona State University
audrey.comstock@asu.edu

Andrew Heiss
Georgia State University
aheiss@gsu.edu

Suparna Chaudhry
Lewis and Clark College
schaudhry@lclark.edu

Audrey L. Comstock, Andrew Heiss, and Suparna Chaudhry, "Derogations, Democratic Backsliding, and International Human Rights During the COVID-19 Pandemic," *Journal of Human Rights* (forthcoming).

ABSTRACT

Did states misuse international legal emergency provisions during the COVID-19 pandemic to justify human rights abuse or did they follow international human rights law? Many governments restricted citizens' freedom of movement, association, and assembly during the crisis, raising questions about states' commitments to international human rights law. Some states used derogations to communicate temporary suspension of international legal provisions in a proportional and non-discriminatory manner, while others did not. We explore the dynamics of democratic backsliding and derogation use during the pandemic. We find that backsliding states were more likely to issue human rights treaty derogations. These derogations had mitigating effects once issued. Backsliding states that issued derogations were more likely to communicate restrictions and were less likely to issue abusive and discriminatory policy during the pandemic. Derogations helped temper abuse in states not experiencing backsliding. However, derogations did not always protect against abuse and media transparency in backsliding states. These results lend support to

the use of flexibility mechanisms in international law and find that most states did not use emergency derogations to heighten human rights violations. The study contributes to the understanding of how international legal measures may help mitigate elements of democratic backsliding during times of crisis.

REPLICATION

To maximize replicability, we wrote our manuscript using Quarto, which allowed us to mix computational figures, text, and tables with the actual prose of the manuscript. This means that there's no need to rely on comments within code to identify the location of each appropriate result in the manuscript—all results are programmatically included when rendering the document.

We use the {renv} package to create a stable version-specific library of R packages, and we use the {targets} package to manage all the file dependencies and run the analysis. {targets} is especially helpful with long-running objects like the main models, which take ≈40 minutes to run—as long as upstream dependencies don't change, the models only need to run once, and can be loaded from {targets}'s data store thereafter.

Because it can often be difficult to set up and configure version-specific libraries of R packages and install specific versions of Stan, we provide three methods for replicating our analysis:

1. **Running a Docker container built and orchestrated with Docker Compose.** This builds a complete computational environment, including non-R software like Quarto, pandoc, LaTeX, dvisvgm, fonts, and other auxiliary elements.
2. **Restoring a project-specific library of all required R packages on your computer with {renv}.** This is the next recommended approach, since {renv} will install package versions as of December 2024 when we ran this code prior to publication. However, you're responsible for installing all the non-R elements, like Quarto and LaTeX (detailed instructions are included below).
3. **Installing the most current versions of all required R packages on your computer systemwide.** This is the least recommended approach, since the included script will grab the latest version of all R packages from CRAN and install them in your system library. As with method 2, you're still responsible for installing all the non-R elements on your own.

The original pre-cleaned data for the analysis is accessible in `lemon-lucifer/data/raw_data`.

The complete {targets} pipeline generates two output artifacts:

- **Manuscript**: HTML and PDF versions of the manuscript and appendix, located at `lemon-lucifer/manuscript/output/` (or at `http://localhost:8888/analysis/paper.html` if you run the pipeline with Docker Compose).
- **Analysis notebook**: A static website containing more complete details about the data, hypotheses, statistical methods, model diagnostics, and other information, located at `lemon-lucifer/_site` (or at `http://localhost:8888` if you run the pipeline with Docker Compose).

Method 1: Docker Compose

The entire analysis can be run in a Docker container based on R 4.4.1, with all packages locked at specific versions defined in `renv.lock`.

Here's how to do this:

1. Install Docker Desktop on your computer (instructions for macOS or Windows).
2. Make sure Docker is running.
3. In the Docker Desktop settings, make sure you allocate at least 8 CPUs and 16 GB of RAM.

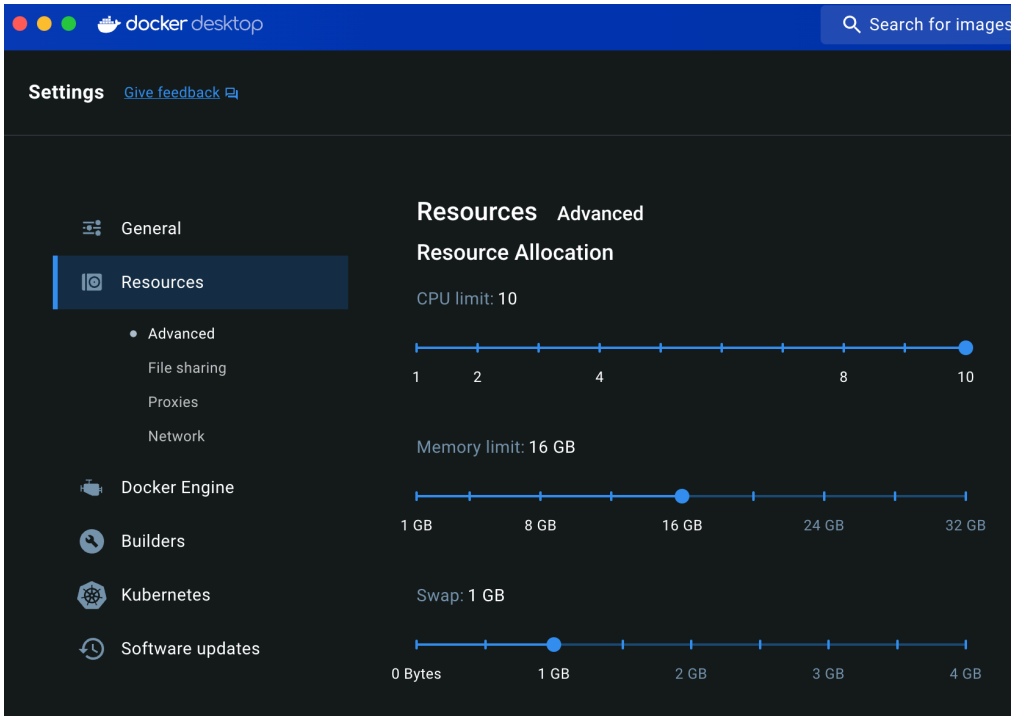

Figure 1: Docker Desktop resource settings

4. Build the analysis with Docker Compose. There are two general approaches:
   - **Using Visual Studio Code or Positron *(recommended)***: If you download Visual Studio Code or Positron and its Docker extension, you can right click on the `docker-compose-prebuilt.yml` file in the File Explorer sidebar and select "Compose Up".

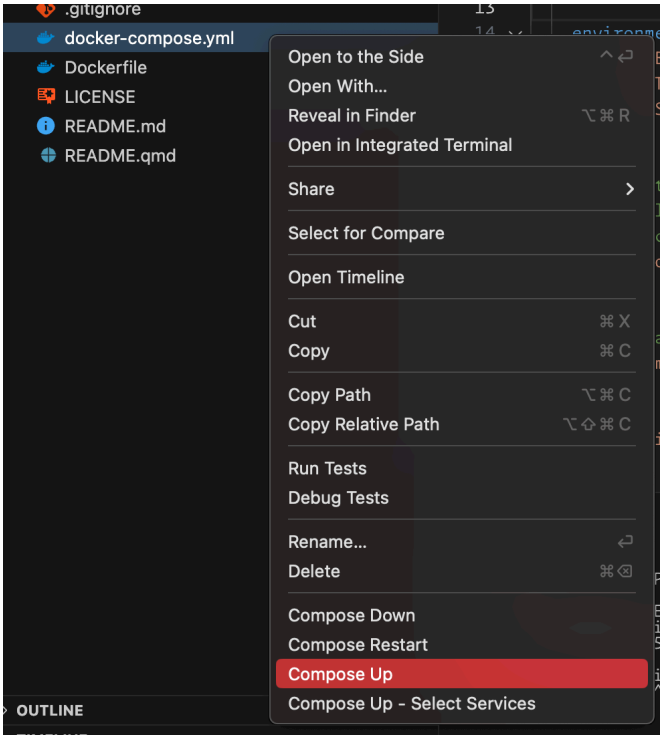

Figure 2: Docker Compose contextual menu in the Visual Studio Code sidebar

- **Using the terminal**: Using a terminal, navigate to this replication code directory and run this:

```
docker compose -f docker-compose.yml up -d
```

5. Wait for the container to build. It takes ≈20 minutes to build the {renv} library (but only the first time you run this; subsequent runs of `docker compose` should be instant), and it takes about ≈20 minutes to run the analysis (but only the first time; subsequent runs of `targets::tar_make()` should be instant).

6. Visit `http://localhost:8787` and open an RStudio session inside the newly-built container in your browser. Any edits you make here will also be reflected on your local computer.

7. Run the {targets} pipeline by running `targets::tar_make()` in the R console. Wait again; it takes ≈20 minutes to run the models, build the statistical notebook website, and render the manuscript in multiple formats. Subsequent runs of the pipeline should be fairly instant, though.

8. When the pipeline is all the way done, visit `http://localhost:8888` to see the analysis notebook and finished manuscript (at `http://localhost:8888/analysis/paper.html`).

   You can also see these outputs on your computer: the analysis notebook is at `_site` and the manuscript and appendix files are at `manuscript/output/`.

Method 2: Run locally with {renv} and project-specific packages

It's also possible to not use Docker and instead run everything locally in a special R package library that is separate from your system library.

0. Install these preliminary things:
   - **R 4.4.1** (or later) and **RStudio**.
   - **Quarto 1.6.1** (or later). As of this writing, the current stable version of Quarto is 1.6.
   - **A C++ compiler and GNU Make**. Complete instructions for macOS, Windows, and Linux are available at CmdStan's documentation. In short, do this:
     - **macOS**: Run this terminal command and follow the dialog that pops up after to install macOS's Command Line Tools:

       ```
       xcode-select --install
       ```

     - **Windows**: Download and install Rtools from CRAN
     - **Linux**: Run this terminal command (depending on your distribution; this assumes Ubuntu/Debian):

       ```
       sudo apt install g++ make
       ```

   - (*macOS only*): Download and install XQuartz
   - **Fonts**: Download and install these fonts (or install them from `misc/fonts` in this repository). On Windows, install these as an administrator so that R and Quarto have access to them.
     - Noto Sans
     - Linux Libertine
     - Libertinus Math
1. Open `lemon-lucifer.Rproj` to open a new RStudio project.
2. Run this to install {cmdstanr}. This is supposed to happen automatically as part of `renv::restore()` below, since {cmdstanr} is in the lockfile, but due to an issue with {renv} (fixed in the development version as of 2024-08-06), it doesn't install correctly because it is hosted at https://stan-dev.r-universe.dev instead of CRAN. So for now, until the next stable release of {renv}, it's easiest to install {cmdstanr} in a separate step.

   ```
   install.packages("cmdstanr", repos = c("https://stan-dev.r-universe.dev",
   "https://packagemanager.posit.co/cran/latest"))
   ```

3. Run `renv::restore()` to install all the packages.
4. Run `cmdstanr::install_cmdstan()` to install CmdStan.

5. Run `tinytex::install_tinytex()` to install a mimimal LaTeX installation if you don't have one installed already.

   TinyTex should install any additional LaTeX packages that it needs when it comes across anything that's missing. You can help it along by installing those non-core packages first with this R command:

   ```
   tinytex::tlmgr_install(
     # tikz things
     c("dvisvgm", "adjustbox", "collectbox", "currfile", "filemod", "gincltex",
       "standalone", "fp", "pgf", "grfext", "libertine", "libertinust1math",
     # template things
       "nowidow", "tocloft", "orcidlink", "abstract", "titling", "tabularray",
      "ninecolors", "enumitem", "textcase", "titlesec", "footmisc", "caption",
       "pdflscape", "ulem", "multirow", "wrapfig", "colortbl", "tabu",
      "threeparttable", "threeparttablex", "environ", "makecell", "sidenotes",
       "marginnote", "changepage", "siunitx", "mathtools", "setspace",
       "ragged2e", "fancyhdr", "pdftex", "preprint")
   )
   ```

6. *(Finally!)* Run `targets::tar_make()` to run the full analysis pipeline. This will take ≈20 minutes the first time.

7. When the pipeline is all the way done, find the analysis notebook at `_site` and the manuscript and appendix files at `manuscript/output/`.

Method 3: Run locally with packages installed systemwide

Finally, it's also possible to not use Docker *and* not use {renv} and instead run everything using R packages that you install systemwide.

0. Install these preliminary things:

   - **R 4.4.1** (or later) and **RStudio**.
   - **Quarto 1.6.1** (or later). As of this writing, the current stable version of Quarto is 1.6.
   - **A C++ compiler and GNU Make**. Complete instructions for macOS, Windows, and Linux are available at CmdStan's documentation. In short, do this:
     - **macOS**: Run this terminal command and follow the dialog that pops up after to install macOS's Command Line Tools:

       ```
       xcode-select —-install
       ```

     - **Windows**: Download and install Rtools from CRAN
     - **Linux**: Run this terminal command (depending on your distribution; this assumes Ubuntu/Debian):

```
sudo apt install g++ make
```

- (*macOS only*): Download and install XQuartz
- **Fonts**: Download and install these fonts (or install them from `misc/fonts` in this repository). On Windows, install these as an administrator so that R and Quarto have access to them.
  - Noto Sans
  - Linux Libertine
  - Libertinus Math

1. ***Before*** opening `lemon-lucifer.Rproj`, open `.Rprofile` and remove or comment out the line that says `source("renv/activate.R")`:

   ```
   # source("renv/activate.R")
   ```

   This will disable {renv} and will make it so R doesn't try to automatically install all the packages specified in `renv.lock`.

2. **Also before** opening `lemon-lucifer.Rproj`, run `docker/misc/install_packages.R`. This uses {pacman} to install all required R packages. It downloads the latest versions of each package from CRAN and installs them systemwide. It will also install {cmdstanr} and LaTeX through {tinytex}.

3. Open `lemon-lucifer.Rproj` to open a new RStudio project.

4. Run `targets::tar_make()` to run the full analysis pipeline. This will take ≈20 minutes the first time.

5. When the pipeline is all the way done, find the analysis notebook at `_site` and the manuscript and appendix files at `manuscript/output/`.

🍋😈: NOTE ON "LEMON LUCIFER" PROJECT NAME

Because project titles change all the time with revisions, rewriting, and peer review, we used {codename} to generate an internal-to-us project name that won't change.

```
library(codename)
codename_message()
#> code name generated by {codename} v.0.5.0. R version 4.4.0 (2024-04-24).

codename(type = "gods", seed = "derogate backslides")
#> [1] "lemon lucifer"
```

LICENSES

CONTRIBUTIONS AND CODE OF CONDUCT

We welcome contributions from everyone. Before you get started, please see our contributor guidelines. Please note that this project is released with a Contributor Code of Conduct. By contributing to this project, you agree to abide by its terms.