The Journal of Open Source Software

# WRF-CMake: integrating CMake support into the Advanced Research WRF (ARW) modelling system

**M. Riechert**[1] **and D. Meyer**[1]

**1** Independent scholar

**DOI:** 10.21105/joss.01468

**Software**

- Review
- Repository
- Archive

**Submitted:** 14 May 2019
**Published:** 09 September 2019



## Summary

The Weather Research and Forecasting model (WRF[1]) model (Skamarock et al., 2019) is an atmospheric modelling system widely used in operational forecasting and atmospheric research (Powers et al., 2017). WRF is released as a free and open-source software and officially supported to run on Unix and Unix-like operating systems and on several hardware architectures from single-core computers to multi-core supercomputers. Its current build system relies on several bespoke hand-written Makefiles, and Perl and Shell scripts that have been supported and extended during the many years of development.

The use of build script generation tools, that is, tools that generate files for native build systems from specifications in a high-level language, rather than manually maintaining build scripts for different environments and platforms, can be useful to reduce code duplication and to minimize issues with code not building correctly (Hoffman, Cole, & Vines, 2009), to make software more accessible to a broader audience, and the support less expensive (Heroux & Willenbring, 2009). As such, a common build script generation tool is CMake. Today, CMake is employed in several projects such as HDF5, EnergyPlus, and ParaView to build modern software written in C, C++, and Fortran in high performance computing (HPC) environments and, by CERN, to allow users to easily set-up and build several million lines of C++ and Python code used in the offline software of the ATLAS experiment at the Large Hadron Collider (LHC) (Elmsheuser, Krasznahorkay, Obreshkov, & Undrus, 2017).

WRF-CMake aims at helping model developers and end-users by adding CMake support to the latest version of WRF and the WRF Processing System (WPS), while coexisting with the existing build set-up. The main goals of WRF-CMake are to simplify the build process involved in developing and building WRF and WPS, add support for automated testing using continuous integration (CI), and the generation of pre-built binary releases for Linux, macOS, and Windows thus allowing non-expert users to get started with their simulations in a few minutes, or integrating WRF and WPS into other software (see, for example, the GIS4WRF project (Meyer & Riechert, 2019)). The WRF-CMake project provides model developers, code maintainers, and end-users wishing to build WRF and WPS on their system several advantages such as robust incremental rebuilds, dependency analysis of Fortran code, flexible library dependency discovery, automatic construction of compiler command-lines based on the detected compiler, and integrated support for MPI and OpenMP. Furthermore, by using a single language to control the build, CMake removes the need to write and support several hand-written Makefiles, and Perl and Shell scripts. The current WRF-CMake set-up on GitHub offers model developers and code maintainers an automated testing infrastructure (see Testing) for Linux, macOS, and Windows, and allows end-users to directly download pre-built binaries

[1] By WRF, we specifically mean the Advanced Research WRF (ARW). The Non-hydrostatic Mesoscale Model (NMM) dynamical core, WRF-DA, WRFPLUS, WRF-Chem, and WRF-Hydro are not currently supported in WRF-CMake.

for common configurations and architectures from the project's website (experimental). WRF-CMake is available as a free and open-source project on GitHub at https://github.com/WRF-CMake and currently includes CMake support for the main Advanced Research WRF (ARW) core and WPS.

## Testing

A fundamental aspect of software development is testing. Ideally, model components should be tested individually and under several testing methodologies (Feathers, 2004). Here, however, as the WRF framework does not offer a way to unit test its components, we instead run separate build and regression tests to evaluate the effects of our changes. While build tests are used to check the absence of compilation errors, regression tests are used to estimate the size of simulation errors resulting from our change.

Build tests are performed for all supported build variants (Table 1) using CI services at every commit. As noted by Hodyss and Majumdar (2007), and Geer (2016), the high sensitivity to initial conditions of dynamical systems, such as the ones used in weather models, can lead to large differences in skill between any two forecasts. It is this high sensitivity to initial conditions that can obscure the source of model error, whether this originates from a change in compiler or architecture, an actual coding error, or indeed, the intrinsic nature of the dynamical system employed.

**Table 1:** Build variants used in build and regression tests. `Make`: original WRF build system files, `CMake`: this paper; `Debug`: compiler optimizations disabled, `Release`: enabled; `serial`: single processor, `dmpar`: multiple with distributed memory (MPI), `smpar`: multiple with shared memory (OpenMP), `dm_sm`: multiple with MPI and OpenMP.

| | Variant |
|---|---|
| OS | `Linux`, `macOS`, `Windows` |
| Build tool | `Make`, `CMake` |
| Build type | `Debug`, `Release` |
| Mode | `serial`, `dmpar`, `smpar`, `dm_sm` |

As a result, the impact of our changes are evaluated using the range-normalized relative percentage error ($\boldsymbol{\delta}_x$) and range-normalized root-mean-square percentage error (NRMSPE; Appendix A). These are computed per domain for all grid points, and for all vertical levels. The errors are assessed by (a) comparing the outputs of prognostic-variable outputs (Table 2) from WRF (Make) against those from WRF-CMake and (b) comparing the outputs for all build variants (for both Make and CMake) against a reference build variant defined as `Linux/Make/Debug/serial`.

These tests are then run for all supported build variants (Table 1) using the WRF-CMake Automated Testing Suite (WATS), and a subset of namelists[2] from the official WRF Testing Framework, using CI services at major code changes (e.g. before merging pull requests), and for 1 hour of simulation time, to constrain computing resources.

Here, we report summary results for the domain showing the greatest error (i.e. innermost; domain 2) after simulating 60 minutes. Values of $\boldsymbol{\delta}_x$ are aggregated for all quantities reported in Table 2 and referred to as $\boldsymbol{\delta}$.

[2] See https://github.com/WRF-CMake/wats/tree/master/cases/wrf

**Table 2:** WRF prognostic variables evaluated during regression tests.

| Symbol | Name | Unit |
|---|---|---|
| $p$ | Air pressure | Pa |
| $\phi$ | Surface geopotential | $m^2\ s^{-2}$ |
| $\theta$ | Air potential temperature | K |
| $\boldsymbol{u}$ | Zonal component of wind velocity | $m\ s^{-1}$ |
| $\boldsymbol{v}$ | Meridional component of wind velocity | $m\ s^{-1}$ |
| $\boldsymbol{w}$ | Vertical component of wind velocity | $m\ s^{-1}$ |

At the start of the simulation, the NRMSPE between WRF (Make) and WRF-CMake is zero (Appendix B, Figure 5), but small, when comparing WRF build variants (both Make and CMake) against a reference variant (`Linux/Make/Debug/serial`; Appendix B, Figure 6), thus suggesting an expected variability of outputs when running WRF across different platforms.

After 60 minutes (simulation time), WRF-CMake produces, on average, small values of $\boldsymbol{\delta}$, with mean close to zero, and most of the error (99.8 %) between -0.05 and 0.05 % (Figure 1). On Linux, the only build variants showing no error are for `Debug/serial` and `Debug/dmpar` (Figure 1 and 2). For NRMSPE (Figure 2), values of $\boldsymbol{w}$ show to be the most sensitive, however, the largest errors are shown for all components of wind velocity, on both, Linux and macOS.

Differences, in particular for `Release` build variants, most likely arise from an inconsistent use of compiler optimization options in WRF (Make) across its C and Fortran files, whereas in WRF-CMake, such options are centrally and consistently applied. Given that `Debug/serial` and `Debug/dmpar` show no error, we would expect the same to be true for the OpenMP variants `Debug/smpar` and `Debug/dm_sm`. Further investigation is required to establish the source of these differences.

When comparing both Make and CMake versions against the reference build variant (i.e. `Linux/Make/Debug/serial`; Figure 3 and 4), the errors appear to be of equal, or greater, magnitude than those shown when comparing WRF (Make) against WRF-CMake for both $\boldsymbol{\delta}$ (Figure 3) and NRMSPE (Figure 4), thus indicating that the variability across build variants may be more important and may also be an inherent feature of WRF.

The choice of operating system has the greatest impact on both $\boldsymbol{\delta}$ and NRMSPE (Figure 3 and 4) over compiler optimization strategies and build tool used. A change in build tool to CMake appears to produce values of $\boldsymbol{\delta}$ and NRMSPE consistent with those obtained from versions of WRF built with the original build scripts[3]. The largest errors are shown for wind velocity and, specifically for $\boldsymbol{u}$ and $\boldsymbol{w}$. Larger values of $\boldsymbol{\delta}$ and NRMSPE between operating systems appear to be a general property of WRF (i.e. with/without CMake support) and should be investigated further.

[3] Comparison on Windows is not made as Windows support is only available in WRF-CMake.

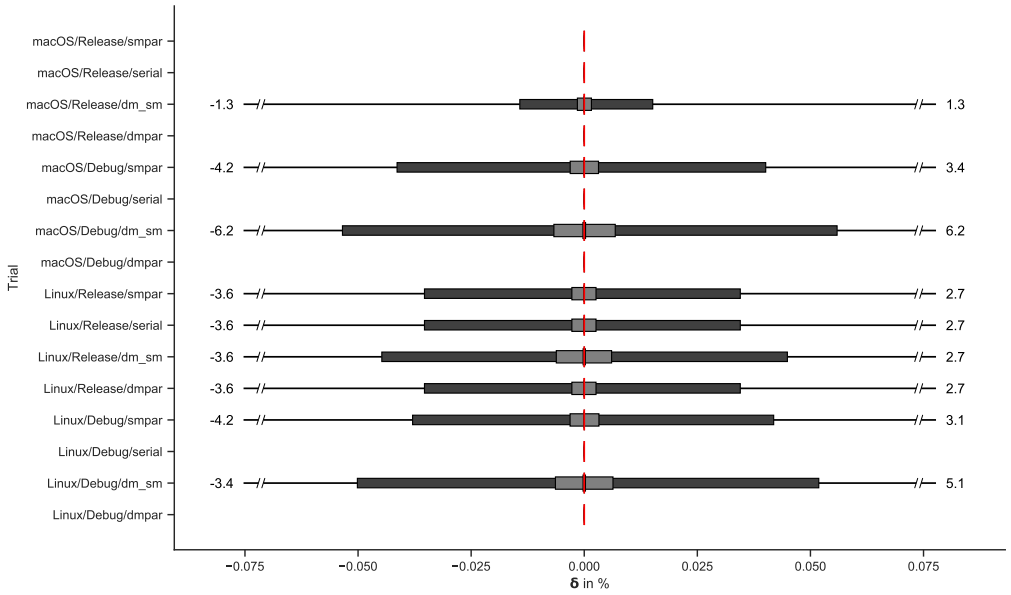


**Figure 1:** WRF (Make) vs WRF-CMake: extended box plots of range-normalized relative percentage errors ($\delta$) for the domain with highest errors only (domain 2) after 60 minutes (simulation time). Extended boxplots show minimum, maximum, median, and percentiles at [99.9, 99, 75, 25, 5, 1, 0.1].

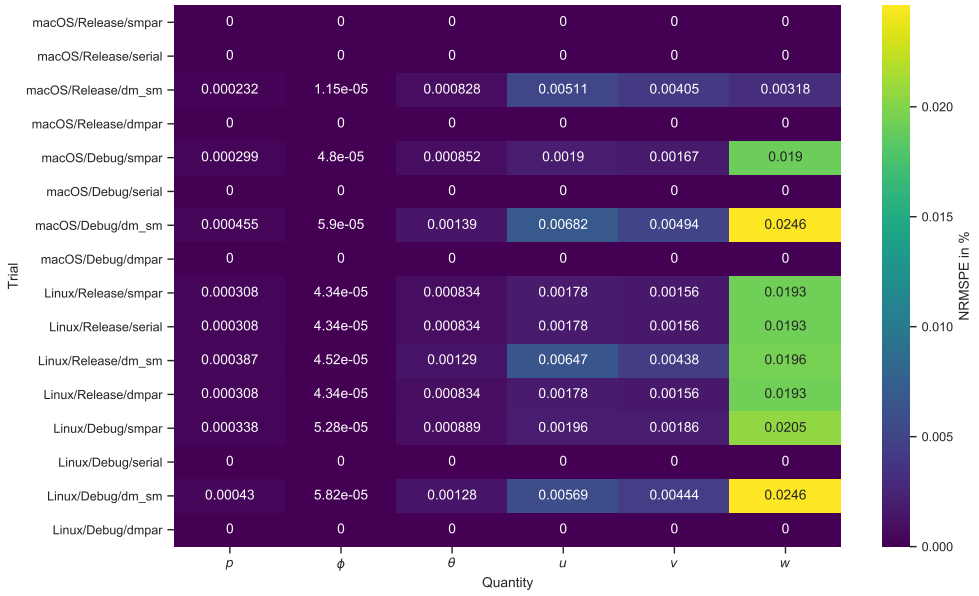


**Figure 2:** WRF (Make) vs WRF-CMake: range-normalized root mean-square percentage error (NRMSPE) for the domain with highest errors only (domain 2) after 60 minutes (simulation time).

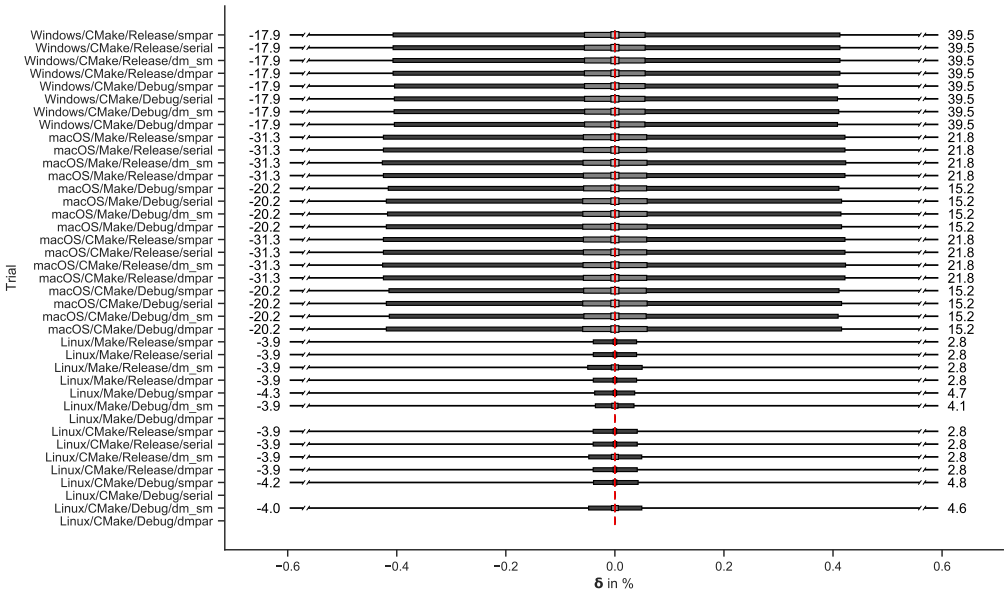


**Figure 3:** WRF (Make and CMake) vs reference build variant: extended box plots of range-normalized relative percentage errors ($\delta$) against the reference build variant (`Linux/Make/Debug/serial`) for the domain with highest errors only (domain 2) after 60 minutes (simulation time). Extended boxplots show minimum, maximum, median, and percentiles at [99.9, 99, 75, 25, 5, 1, 0.1].

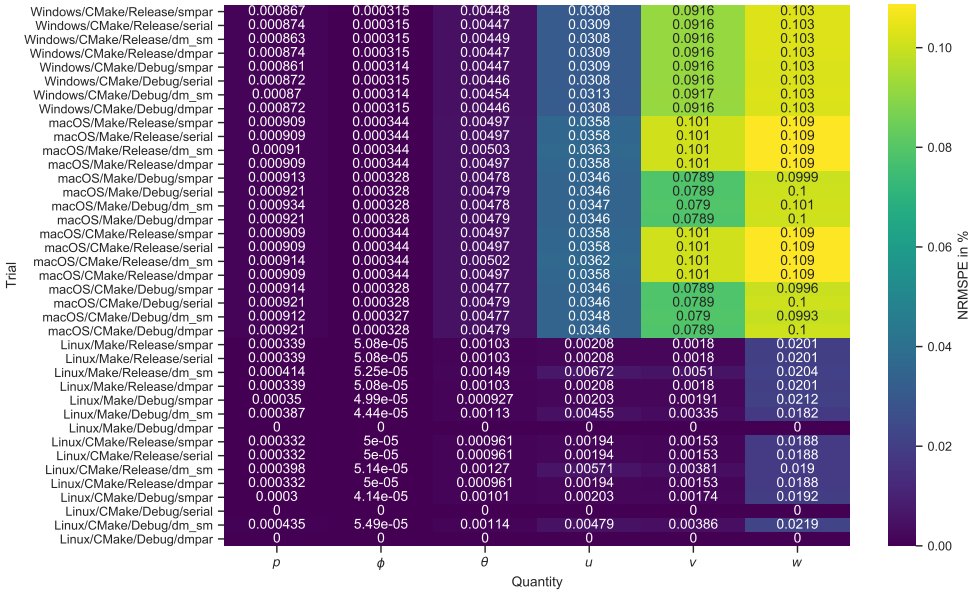


**Figure 4:** WRF (Make and CMake) vs reference build variant: range-normalized root mean-square percentage errors (NRMSPE) against the reference build variant (`Linux/Make/Debug/serial`) for the domain with highest errors only (domain 2) after 60 minutes (simulation time).

## Concluding remarks

We introduce WRF-CMake as a modern replacement for the existing WRF build system. Its main goals are to simplify the build process involved in developing and building WRF and WPS, add support for automated testing using CI, and automate the generation of pre-built binary releases for Linux, macOS, and Windows. Results from regression tests indicate that, when evaluating outputs of prognostic variables, errors between WRF and WRF-CMake are generally small, or smaller, then errors originating from a change in optimization strategy (e.g. `Debug`, `Release`) or a change in platform (e.g. Linux to macOS). These larger errors appear to be a general property of WRF (i.e. with/without CMake support) and should be investigated further. Depending on feedback and general uptake by the community, future work may involve adding support for WRF-DA, WRFPLUS, WRF-Chem, and WRF-Hydro.

## Acknowledgements

We thank A. J. Geer at the European Centre for Medium-Range Weather Forecasts (ECMWF) for the useful discussion and feedback concerning the topic of error growth in dynamical systems. We also thank the reviewers I. Beekman and A. Hilboll for their time and useful contributions to both paper and software.

## Appendix A Statistics

The vector of range-normalized relative percentage error ($\boldsymbol{\delta}_x$) between two vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ of paired quantities $x_1$ and $x_2$ is defined as:

$$\boldsymbol{\delta}_x := \frac{\boldsymbol{x}_1 - \boldsymbol{x}_2}{R_{\boldsymbol{x}_1}} \times 100\ \%, \tag{1}$$

where $R_{\boldsymbol{x}_1}$ is the range of $\boldsymbol{x}_1$.

Similarly, the range-normalized root-mean-square percentage error (NRMSPE) is defined as:

$$\mathrm{NRMSPE} := \frac{\mathrm{RMSE}}{R_{\boldsymbol{x}_1}} \times 100\ \%, \tag{2}$$

with the root-mean-square-error (RMSE) defined as:

$$\mathrm{RMSE} := \sqrt{\frac{\sum_{i=1}^{N} (\boldsymbol{x}_{1,i} - \boldsymbol{x}_{2,i})^2}{N}}, \tag{3}$$

and $N$ is the size of the vector.

# Appendix B Supplementary figures

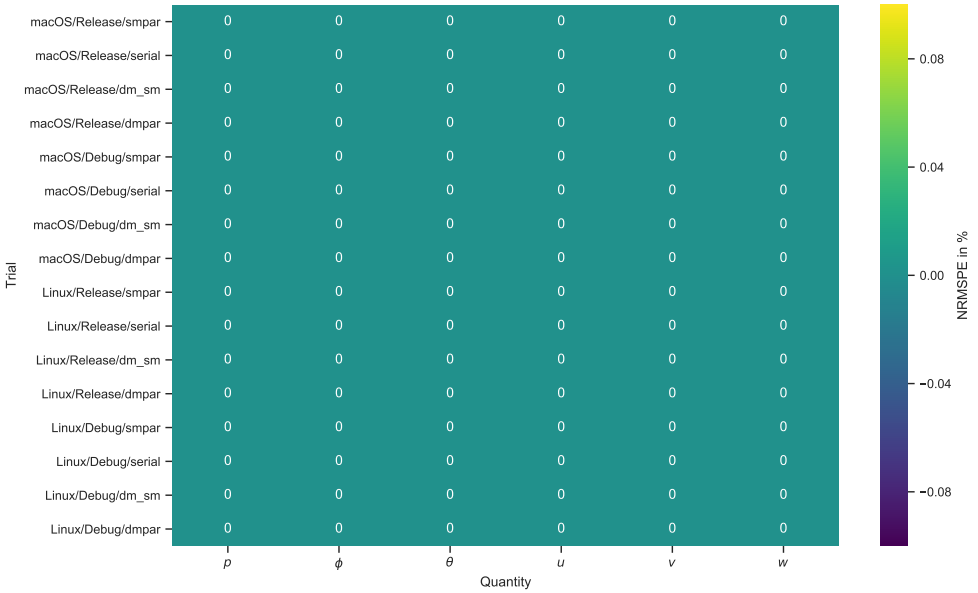


**Figure 5:** WRF (Make) vs WRF-CMake: range-normalized root mean-square percentage error (NRMSPE) at 0 minutes (simulation time).

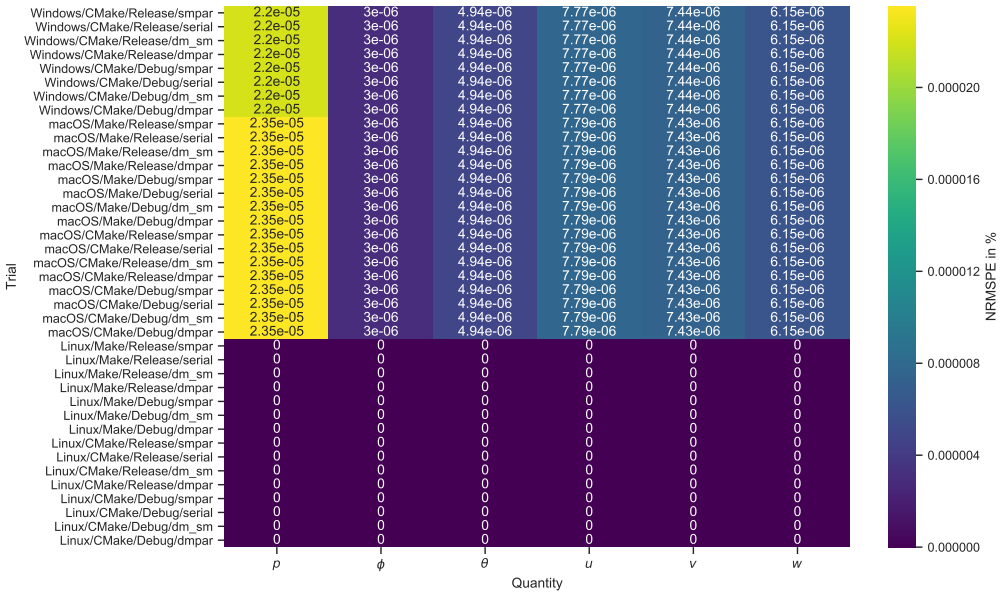


**Figure 6:** WRF (Make and CMake) vs reference build variant: range-normalized root mean-square percentage errors (NRMSPE) against the reference build variant (`Linux/Make/Debug/serial`) for the domain with highest errors only (domain 2) at 0 minutes (simulation time). Extended boxplots show minimum, maximum, median, and percentiles at [99.9, 99, 75, 25, 5, 1, 0.1].

## References


Elmsheuser, J., Krasznahorkay, A., Obreshkov, E., & Undrus, A. (2017). Large scale software building with CMake in ATLAS. *Journal of Physics: Conference Series*, *898*, 072010. doi:10.1088/1742-6596/898/7/072010

Feathers, M. (2004). *Working Effectively with Legacy Code* (p. 456). New Jersey: Prentice Hall.

Geer, A. J. (2016). Significance of changes in medium-range forecast scores. *Tellus A: Dynamic Meteorology and Oceanography*, *68*(1), 30229. doi:10.3402/tellusa.v68.30229

Heroux, M. A., & Willenbring, J. M. (2009). Barely sufficient software engineering: 10 practices to improve your CSE software. In *2009 ICSE workshop on software engineering for computational science and engineering*. IEEE. doi:10.1109/secse.2009.5069157

Hodyss, D., & Majumdar, S. J. (2007). The contamination of 'data impact' in global models by rapidly growing mesoscale instabilities. *Quarterly Journal of the Royal Meteorological Society*, *133*(628), 1865–1875. doi:10.1002/qj.157

Hoffman, B., Cole, D., & Vines, J. (2009). Software process for rapid development of HPC software using CMake. In *2009 DoD high performance computing modernization program users group conference*. IEEE. doi:10.1109/hpcmp-ugc.2009.62

Meyer, D., & Riechert, M. (2019). Open source QGIS toolkit for the advanced research WRF modelling system. *Environmental Modelling & Software*, *112*, 166–178. doi:10.1016/j.envsoft.2018.10.018

Powers, J. G., Klemp, J. B., Skamarock, W. C., Davis, C. A., Dudhia, J., Gill, D. O., Coen, J. L., et al. (2017). The Weather Research and Forecasting Model: Overview, System Efforts, and Future Directions. *Bulletin of the American Meteorological Society*, *98*(8), 1717–1737. doi:10.1175/BAMS-D-15-00308.1

Skamarock, W. C., Klemp, J. B., Dudhia, J., Gill, D. O., Liu, Z., Berner, J., Wang, W., et al. (2019). *A description of the advanced research wrf model version 4* (p. 145). NCAR Technical Note NCAR/TN-556+STR. doi:10.5065/1dfh-6p97