This manuscript describes a Python wrapper for the atmospheric component of the Unified Forecast System (UFS), NOAA's next-generation modeling system that will replace a large variety of legacy systems. The manuscript is very well written, easy to understand and follow. The motivation and need for such a tool are laid out clearly, and a high-level description of the software is accompanied but full access to the open source code and examples to exercise.

A few minor comments for the authors to consider:

- Readers may find the terminology "FV3GFS" confusing, in particular when referred to as the "prototype of the operational GFS". The term "FV3GFS" presumably refers to the initial releases v0/v1 of the atmospheric component (called fv3atm by NOAA) of NOAA's modeling system, embedded in the NEMS infrastructure at that time (e.g., https://vlab.ncep.noaa.gov/documents/2381028/3659001/FV3GFS+Public+Release+V1/37d9d3e8-bd99-b728-d1d7-49c2b007be38). The entire model was accordingly called NEMSfv3gfs, which later was changed into UFS (Unified Forecast System). This is mentioned briefly in the acknowledgements. In particular there, but possibly also in the introduction, it would be great if that distinction could be made more clearly.

- Line 39: Should this read "... it allows copying variables into (set_state) or out of (get_state) the Fortran ..." (it is the opposite in the text)?

- Line 87: see lines 253-256 below.

- Figure 3 and line 207: The manuscript somewhat assumes that readers are familiar with the underlying grid of the FV3 model, i.e. that a cubed sphere has six tiles and that the code is written such that at least one MPI task per tile is required. Maybe this could be mentioned in the text.

- Lines 2018-2019: The data copies return the data on the native grid. Would it be possible, if so desired by the user or the specific type of application, to perform regridding operations on the fly during this step (note: this may not make sense when applying machine learning models, but for other applications such as plotting etc it would).

- Lines 225-232: The variable "database" encoded in json files resembles in its contents and requirements the Common Community Physics Package (CCPP) metadata. Both
contains attributes such as standard names, units, dimensions, etc. Since CCPP was selected as the operational physics interface in all UFS applications in the following years (post-GFSv16), and since the authors explicitly mention taking advantage of the CCPP infrastructure in the future, would it make sense to consider interoperability here, i.e. either adopt the CCPP metadata table format or provide tools to convert between the formats?

- Lines 234-243. Since the prototype releases FV3GFS v0/v1, the portability of the UFS has improved considerably. This also includes the prerequisite libraries mentioned in the text. To date, the UFS community released the Medium-Range Weather Application v1.0 and v1.1, as well as the Short-Range Weather Application v1.0. All of these run on a wide range of platforms, including generic Linux and macOS, with detailed instructions and software packages (NCEPLIBS-external, NCEPLIBS). Also, as part of the releases, ready-to-use Amazon Machine Images (AMIs) for use on Amazon’s AWS are provided, for the SRW App v1.0 a Docker container is also provided.

- Line 245. How independent is the wrapper from the model grid (horizontal and vertical)? Are the interfaces generic and the developers need to add implementations for their model?

- Lines 253-256: One alternative option that this reviewer has played with briefly is to use shared memory for the in-situ model variables and to provide an external process with information such as "variable x can be found at this location in memory". With the appropriate metadata (type/kind/size/dimensions), an external process could read this data while the model is running. This would also be useful wrt. NVRAM. Have the authors considered this idea, and if yes, what were the stumbling blocks?

- Section 4.5: One important aspect for model development is debugging and profiling, for which dedicated tools exist (gdb, valgrind, Intel Parallel Studio, ARM Forge, tau, Score-P/Scalasca, ...). Can these tools still be used if the top-level routine is Python like in the current implementation?

- Lines 281-283: The authors have shown that there is no performance degradation in their tests. These use a C48 grid with 6 MPI tasks, presumable on a single node. Have tests been conducted for larger runs across multiple nodes, and do the authors expect that this statements holds true even when going to very high / extreme scaling applications across entire HPCs?

- Lines 296 and 333: This reference is incorrect.

- Acknowledgements: see above for a clearer distinction of UFS, FV3GFS, fv3atm.

- References: Please check all references thoroughly, several of them have "wrong" author names etc (e.g. lines 345 and 350, also 333 as mentioned above)

- General comment on language: The document uses American English as far as the reviewer can see, but in at least one place (line 24, "modelling workflows") British English is used. Please check.