The Journal of Open Source Software

Sapsan: Framework for Supernovae Turbulence Modeling with Machine Learning

Platon I. Karpov[1, 2], **Iskandar Sitdikov**[3], **Chengkun Huang**[2], **and Chris L. Fryer**[2]

1 Department of Astronomy & Astrophysics, University of California, Santa Cruz, CA **2** Los Alamos National Laboratory, Los Alamos, NM **3** Provectus IT Inc., Palo Alto, CA

DOI: 10.21105/joss.0XXXX

Software

- Review
- Repository
- Archive

Editor: Editor Name

Submitted: 01 January XXXX
Published: 01 January XXXX

Summary

Sapsan is a framework to make Machine Learning (ML) more accessible in the study of turbulence, with a focus on astrophysical applications. Sapsan includes modules to load, filter, sample down, batch, and split your data for training and validation. Furthermore, the evoluation modules, among other features, include construction of power spectra and comparison to established analytical turbulence closure models, such as a gradient model. Thus, Sapsan takes out all the hard work from data preparation and analysis, leaving one focused on ML model design, layer by layer.

Statement of Need

It has been challenging for domain sciences to adapt Machine Learning (ML) for their respective projects. From the authors' background, it has been particularly difficult to employ ML for physical simulations, modeling turbulence. It is rather challenging to prove that an ML model has arbitrarily learned the laws of physics embedded into the problem with the ability to extrapolate within the parameter-space of the simulation. Inability to directly infer the prediction capabilities of ML is one of the major causes behind the slow adaption rates, however, the effectiveness of ML cannot be ignored by the community.

Turbulence is ubiquitous in astrophysical environments; however, it involves physics at a vast range of temporal and spatial scales, making accurate fully-resolved modeling difficult. Various analytical turbulence models have been developed to be used in simulations using time or spatial averaged governing equations, such as in RANS (Reynolds-averaged Navier-Stokes) and LES (Large Eddy Simulation), but accuracy is lacking. In search of better methods to model turbulence in core-collapse supernovae, it became apparent that ML has great potential to produce more accurate turbulence models on an unaveraged subgrid-scale than the current methods. Scientists from both industry and academia (King et al., 2016), (Zhang et al., 2018) have already begun using ML for applied turbulent problems, but none reach out to a theoretical medium of physics and astronomy community on a practical level. For example, physics-based model evaluation and interpretability tools are not standardized nor they are widely available. As a result, it is a common struggle to verify published results, with the setup not fully disclosed, the code being poorly structured and/or commented or even worse - not publicly available; the problem ML community can relate to as well (Hutson, 2018). Thus, it is not surprising that there is considerable skepticism against ML in physical sciences, with astrophysics being no exception (Carleo et al., 2019).

In pursuit of our supernova (SNe) study, the issues outlined above became painfully apparent. Thus, we attempted to simplify the barrier to entry for new researchers in domain science

The Journal of Open Source Software

fields studying turbulence to employ ML, with the main focus on astrophysical applications. As a result, an ML python-based pipeline called `Sapsan` has been developed. The goals have been to make it accessible and catered to the community through Jupyter Notebooks, command-line-interface (CLI) and graphical-user-interface (GUI)[1] available for the end-user. `Sapsan` includes built-in optimized ML models for turbulence treatment, both conventional and physics-based. More importantly, at its core, the framework is meant to be flexible and modular, hence there is an intuitive interface for users to work on their own ML algorithms. In addition, `Sapsan` brings best practices from the industry when it comes to ML development frameworks. For example, `Sapsan` includes docker containers for reproducible release, as well as MLflow for experiment tracking.

`Sapsan` is distributed through Github and pip. For further reference, wiki is mainted on Github as well.

Framework

`Sapsan` organizes workflow via three repesctive stages: data preparation, machine learning, and analysis. The whole process can be further wrapped in Docker to publish your results for reproducibility, as reflected by Figure 1. Let's brake down each stage in the context of turbulence subgrid modeling, e.g. a model to predict turbulent behavior at the under-resolved simulation scales.

- **Data Preparation**
  - **Loading Data:** `Sapsan` is ready to process common 2D & 3D hydrodynamic (HD) and magnetohydrodynamic (MHD) turbulence data, in simulation-code-specific data formats, such as HDF5 (with more to come per community need).
  - **Transformations:** a variety of tools are ready for you to ready the data for training
    - **Filter:** in order to build a subgrid model, one will have to filter the data, e.g. remove small-scale perturbations. A few examples would be either a box, spectral, or a gaussian filter. The data can be filtered on the fly within the framework.
    - **Sample:** to run quick tests of your model, you might want to test on a sampled version of the data, wyhile retaining the full spacial domain. Thus, equidistant sampling is ready to go in `Sapsan`
    - **Batch:** the pipeline will spatially batch the data
    - **Split:** data is divided into testing and validation datasets
- **Machine Learning**
  - **Model Set Up:** different ML models would be appropriate for different physical regimes. `Sapsan` provides templates for a selection of both conventional and physics-based models with more to come. Only important options are left up to the user to edit, with most of the overhead kept in the backend.
    - **Layers:** define and order the ML layers
    - **Tracking:** add extra parameters to be tracked by MLflow
    - **Loos Function:** decide on a conventional or physics-based/custom loss function to model the data
    - **Optimizer:** choose how to optimize the training
    - **Scheduler:** select how to adjust the learning rate
- **Analysis**

[1] demo available at [sapsan.app](https://sapsan.app/])

- **Trained Model:** a turbulence subgrid model telling us how small-scale structure affects the large scale quantities, i.e. it completes or ''closes" the governing large-scale equations of motion with small-scale terms. The prediction from a trained ML model is used to provide the needed quantities.
- **Analytical Tools:** compare the trained model with conventional analytic turbulence models, such as Dynamic Smagorisnky (Lilly, 1966) or Gradient model [@]. Furthermore, conduct other physics-based tests, for example, compute the power spectrum of your prediction.

For further information on each stage please refer to Sapsan's Wiki on Gihub.

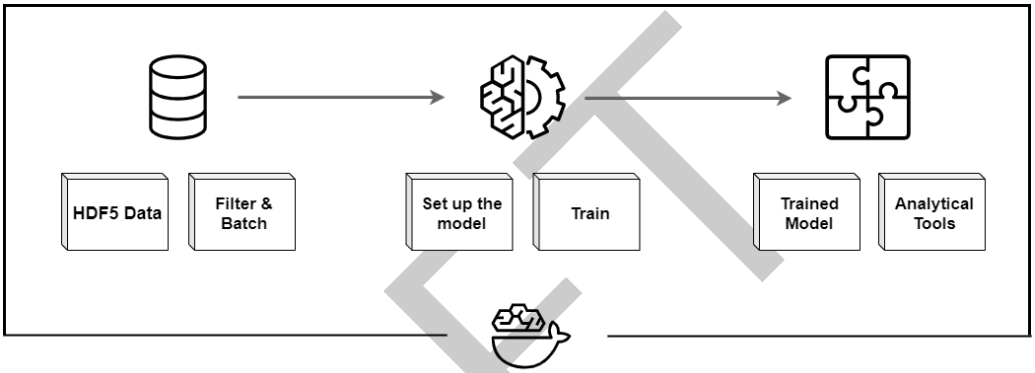

Figure 1: High-level overview of `Sapsan`'s workflow.

Dependencies

Here only the core dependencies will be covered. Please refer to GitHub for the full list.

- **Training**
  - **PyTorch:** `Sapsan`, at large, relies on PyTorch to configure and train ML models. Thus, the parameters in the aforementioned **Model Set Up** stage should be configured with PyTorch functions. Convolutional Neural Network (CNN) and Physics-Informed Convolutional Auto Encoder (PICAE) examples included with `Sapsan` are based on PyTorch. (Paszke et al., 2019)
    - **Catalyst:** used as part of the backend to configure early-stopping of the model and logging (Kolesnikov, 2018)
  - **Scikit-learn:** the framework supports it, as shown in the Kernel Ridge Regression (KRR) example in `Sapsan`. However, due to lack of scalability and features, it is advised to use PyTorch based setup. (Pedregosa et al., 2011)
- **Tracking**
  - **MLflow:** allows for tracking across large quantities of training and evaluation runs through an intuitive web interface. Beyond the few default parameters, a user can include custom parameters to be tracked. (Databricks, 2020)
- **Interface**
  - **Jupyter Notebook:** the most direct and versatile way to use `Sapsan`
  - **Streamlit (GUI):** a graphical user interface (GUI) for `Sapsan`. While lacking in its flexibility, it is perfect to construct a demo to present your project. To try it out, please visit sapsan.app (Treuille, 2019)
  - **Click (CLI):** a command line interface (CLI) for `Sapsan`. It is used to get the user up and running with templates for a custom project. (Ronacher, 2021)

Applications

While Sapsan is built to be highly customizable to be used in a wide variety of projects in physical sciences, it is optimized for the study of turbulence.

Hydro simulations

Here are a few examples of a turbulence closure model, trained on the high-resolution Johns Hopkins Turbulence Database (JHTDB) (Li et al., 2008). The dataset used in this comparison is a direct numerical simulation (DNS) of statistically-stationary isotropic 3D MHD turbulence dataset, 10243 in spatial resolution and covering roughly one large eddy turnover time over 1024 checkpoints, e.i. dynamical time of the system (Eyink et al., 2013). We are comparing it with a commonly used Dynamic Smagorinsky (DS) turbulence closure model (Lilly, 1966). On Sapsan side, a Kernel Ridge Regression model (Murphy, 2012) is used to demonstrate the effectiveness of conventional ML approaches in tackling turbulence problems. In this test we used the following setup:

- **Train features:** velocity (u), vector potential (A), magnetic field (B), and their respective derivatives at timestep $= 1$. All quantities have been filtered to remove small-scale perturbations, mimicking the lower fidelity of a non-DNS simulation.
- **Model Input:** low fidelity velocity (u), vector potential (A), magnetic field (B), and their respective derivatives at a set timestep in the future.
- **Model Output:** velocity stress tensor (τ) at the matching timestep in the future, which effectively represents the difference between large and small scale structures of the system.

In Figure 2, it can be seen that the ML-based approach significantly outperforms the DS subgrid model in reproducing the probability density function, i.e. statistical distribution of the stress tensor. The results are consistent with (King et al., 2016).

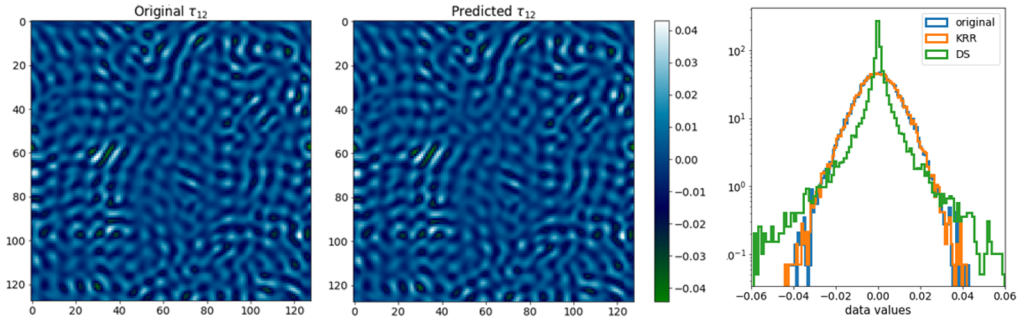

Figure 2: Predicting turbulent stress-tensor in statistically-stationary isotropic MHD turbulence setup. The outmost left plot compares the original spatial map of the stress-tensor component to the plot in the middle depicting the predicted spatial map. The plot on the right presents probability density functions (PDF), i.e. distributions, of the original stress-tensor component values, the ML predicted values, and the conventional DS subgrid model prediction.

Supernovae

Even though the conventional regression-based ML approach worked well in the previous section, the complexity of our physical problem forced us to seek a more rigorous ML method. Supernovae host a different physical regime that is far from the idealistic MHD turbulence case

from before. Here we are dealing with dynamically changing statistics and evolution of the turbulence that is not necessarily isotropic. Depending on the evolutionary stage, turbulence can behave drastically different, hence a more sophisticated model is required. With Sapsan, we have tested a 3D CNN (Convolutional Neural Network) model in attempts to predict a turbulent velocity stress tensor in a realistic Core-Collapse Supernova (CCSN) case. Figure 3 presents results of the following:

- **Train features:** velocity (u), magnetic field (B), and their respective derivatives at timestep = 500 (halfway of the total simulation). All quantities have been filtered to remove small-scale perturbations, mimicking the lower fidelity of a non-DNS simulation.
- **Model Input:** low fidelity velocity (u), magnetic field (B), and their respective derivatives at a set timestep in the future.
- **Model Output:** velocity stress tensor (τ) at the matching timestep in the future, which effectively represents the difference between large and small scale structures of the system.

In this case, the matching level of the distributions is the most important factor. It can be seen that the probability density functions match quite closely, with the outlier at the most negative values being the exception, even though the prediction is done far into the future (timestep=1000, end of the simulation time). For further discussion on the comparison of the models and the results, please refer to [the ApJ paper].

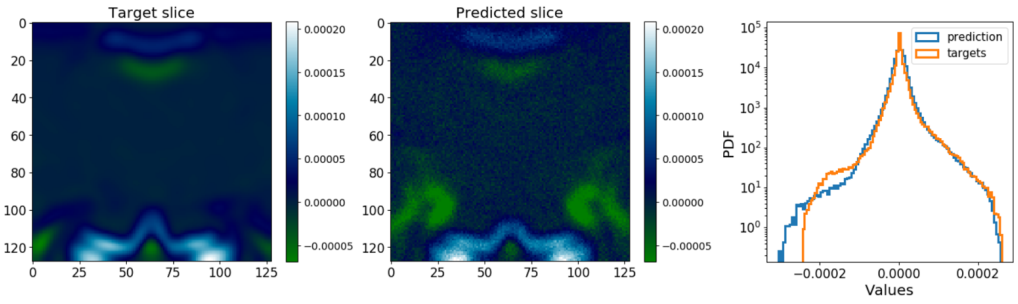

Figure 3: Predicting Turbulent Stress-Tensor in a Core-Collapse Supernovae (CCSN). The model has been trained on a 3D MHD direct numerical simulation (DNS) of the first 10ms after the shockwave has bounced off of the core in a CCSN scenario. On the left, the two figures are the 2D slices of a 3D data-cube prediction, with the right plot presenting a comparison of PDFs of the original 3D data, 3D ML prediction, and a conventional Gradient subgrid model.

Acknowledgements

The development of Sapsan was supported by the Laboratory Directed Research and Development program and the Center for Space and Earth Science at Los Alamos National Laboratory through the student fellow grant. In addition, We would like to thank DOE SciDAC for additional funding support.

References

Carleo, G., Cirac, I., Cranmer, K., Daudet, L., Schuld, M., Tishby, N., Vogt-Maranto, L., & Zdeborová, L. (2019). Machine learning and the physical sciences*. *Reviews of Modern Physics*, *91*(4), 045002. https://doi.org/10.1103/RevModPhys.91.045002

Databricks, I. (2020). MLflow. In *GitHub repository*. https://github.com/mlflow/mlflow; GitHub.

Eyink, G., Vishniac, E., Lalescu, C., Aluie, H., Kanov, K., Bürger, K., Burns, R., Meneveau, C., & Szalay, A. (2013). Flux-freezing breakdown in high-conductivity magnetohydrodynamic turbulence. *Nature*, *497*(7450), 466–469. https://doi.org/10.1038/nature12128

Hutson, M. (2018). Artificial intelligence faces reproducibility crisis. *Science*, *359*(6377), 725–726. https://doi.org/10.1126/science.359.6377.725

King, R. N., Hamlington, P. E., & Dahm, W. J. A. (2016). Autonomic closure for turbulence simulations. *Phys. Rev. E*, *93*, 031301. https://doi.org/10.1103/PhysRevE.93.031301

Kolesnikov, S. (2018). Accelerated DL r&d. In *GitHub repository*. https://github.com/catalyst-team/catalyst; GitHub.

Li, Y., Perlman, E., Wan, M., Yang, Y., Meneveau, C., Burns, R., Chen, S., Szalay, A., & Eyink, G. (2008). A public turbulence database cluster and applications to study Lagrangian evolution of velocity increments in turbulence. *Journal of Turbulence*, *9*, N31. https://doi.org/10.1080/14685240802376389

Lilly, D. K. (1966). On the application of the eddy viscosity concept in the Inertial sub-range of turbulence. *NCAR Manuscript 123*.

Murphy, K. P. (2012). *Machine learning: A probabilistic perspective* (pp. 492–493). The MIT Press.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Ronacher, A. (2021). *Click*. https://click.palletsprojects.com/

Treuille, A. (2019). Turn python scripts into beautiful ML tools. *Towards Data Science*, *8*.

Zhang, W., Zhu, L., Liu, Y., & Kou, J. (2018). Machine learning methods for turbulence modeling in subsonic flows over airfoils. *arXiv e-Prints*, arXiv:1806.05904. http://arxiv.org/abs/1806.05904