NASA/TM-20250005606

# TFExec—A Fortran Bridge for TensorFlow Models

*Ian S. Howell, Joshua Stuckner, and Trenton M. Ricks*
*Glenn Research Center, Cleveland, Ohio*

June 2025

# NASA STI Program Report Series

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION.
  Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM.
  Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT.
  Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION.
  Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.

- SPECIAL PUBLICATION.
  Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION.
  English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

NASA/TM-20250005606

# TFExec—A Fortran Bridge for TensorFlow Models

*Ian S. Howell, Joshua Stuckner, and Trenton M. Ricks*
*Glenn Research Center, Cleveland, Ohio*

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

June 2025

## Acknowledgments

This work was supported by the NASA Transformational Tools and Technologies (TTT) project under the Transformative Aeronautics Concept Program within the Aeronautics Research Mission Directorate.

This work was sponsored by the
Transformative Aeronautics Concepts Program.



*Level of Review*: This material has been technically reviewed by technical management.

This report is available in electronic form at https://www.sti.nasa.gov/ and https://ntrs.nasa.gov/

NASA STI Program/Mail Stop 050
NASA Langley Research Center
Hampton, VA 23681-2199

# TFExec—A Fortran Bridge for TensorFlow Models

Ian S. Howell, Joshua Stuckner, and Trenton M. Ricks
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

## Abstract

Multiscale physics simulations can become computationally expensive, particularly when resolving microscale behavior at every macroscale integration point in a structural scale finite element model. Machine learning surrogates can replace these costly submodels, accelerating simulations while maintaining accuracy. However, integrating TensorFlow models into legacy software tools, such as Abaqus, CalculiX, and NASMAT, has several nontrivial technical challenges. TFExec is a high-performance interface that enables efficient inference of TensorFlow models within Fortran-based simulation codes. It supports batch processing, multiple models in memory, and a low-latency application programming interface (API) via a dynamically linked C++ library. By streamlining machine learning integration, TFExec significantly reduces computational cost in multiscale modeling while preserving the predictive power of physics-based simulations.

## Introduction

Integrating machine learning (ML) surrogate models with physics-based composite modeling codes has the potential to significantly reduce computational costs while maintaining predictive accuracy (Refs. 1 to 4). Finite element analysis (FEA) codes such as Abaqus, CalculiX, and other computational mechanics tools like NASMAT are widely used for modeling complex material behavior, but high-fidelity simulations, particularly in multiscale problems, remain computationally intensive (Refs. 5 and 6). Replacing microscale models with ML-based surrogates can approximate the response of these simulations with substantially more efficient computation through highly parallel tensor operations on graphics processing units (GPUs). However, integrating these surrogates into existing Fortran-based interfaces comes with technical and engineering challenges related to interoperability and computational efficiency.

To be practical for high-fidelity simulations, a ML surrogate must be evaluated efficiently within the FEA solver, often at numerous integration points per timestep. This requires low-latency inference, the ability to handle batch evaluations to minimize overhead and take advantage of parallelization, and support for multiple models in memory to account for spatially varying material behavior. TFExec addresses these requirements by providing a flexible interface between TensorFlow models and Fortran-based tools. It enables batch inference and allows multiple models to be loaded simultaneously.

Beyond composite modeling, many scientific and engineering disciplines rely on Fortran-based software for high-fidelity simulations that is still being actively developed and maintained. In materials science, codes like Thermo-Calc are essential for thermodynamic and phase diagram calculations (Ref. 7). In computational fluid dynamics (CFD), solvers such as FUN3D (Ref. 8) and US3D (Ref. 9) continue to drive advancements in aerospace and thermal analysis. These codes are optimized for large-scale simulations but often face computational bottlenecks when high-resolution physics-based models are required. Integrating ML surrogates into these workflows presents an opportunity to dramatically accelerate simulations, enabling faster design iterations and real-time predictive capabilities.

By providing a robust interface between TensorFlow models and Fortran-based solvers, TFExec extends the benefits of ML acceleration to a wide range of scientific applications. The ability to efficiently replace expensive submodels with trained surrogates can improve the scalability of simulations, making previously intractable problems feasible within practical time constraints. This approach is particularly relevant in fields where high-dimensional parameter spaces or complex multiscale interactions make traditional simulations computationally prohibitive. As scientific computing continues to evolve, bridging modern ML techniques with existing Fortran-based modeling tools ensures that decades of validated physics-based codes remain at the forefront of computational science, benefiting from the rapid advancements in data-driven modeling.

The public open-source code for this software will be hosted at https://github.com/nasa/tfexec. The internal repository can be accessed at https://gitlab.grc.nasa.gov/joshua.stuckner/tfexec.

## Interface Architecture

The architecture of TFExec allows users to load models, evaluate inputs on models, and remove models from memory. The majority of this work leverages the CPPFlow library (Ref. 10) for both maintainability and speed. CPPFlow is a modern C++ library built on top of the official TensorFlow language bindings for C and C++. There are two interfaces defined in this library, the first is a C/C++ interface that may be statically compiled into other C/C++ projects. The other is an external Fortran interface that allows other projects to use the same functionality.

### C++ Interface and Implementation

The C++ interface consists of three functions. The generic interface, specifically used by Fortran in this project, formats inputs before feeding them into these functions to use the library. This interface was built with modern C++, following the C++17 standard.

1. The *load_model* function takes in a string as well as a pointer to an unsigned integer. The string provides the path to the TensorFlow saved model while the unsigned integer pointer indicates the new model's ID. The models are stored as unique pointers in a list. A separate map points unsigned integer IDs to the list iterator for the corresponding model. If the program loads the model successfully with no errors, load_model returns 0, while if any errors occurred, they are printed to the standard error pipe and then a 1 is returned.
2. The *delete_model* function removes a model from memory. This function takes as input an unsigned integer representing the ID of the model. If the model denoted by the ID is successfully deleted and removed from memory, the function returns a 0, while if an error occurs, the error is printed to the standard error pipe and a 1 is returned.
3. The *predict* function executes the model on some input and returns the output. The inputs to this function include the ID of the model that is being executed, the data being input into the model, and the names of the outputs to retrieve from the model. The output of the function is the retrieved output tensors. The input data is a vector of pairs of strings and tensors where the string is the name of the input that the tensor corresponds to. After execution of the model on the input data, each output name provided is iterated over and the corresponding output is moved into the output vector of tensors. This function handles errors in the same way as the previously stated functions.

## Fortran Interface and Implementation

The Fortran interface should be compiled along with a user's Fortran source code. Alternatively, this file can be compiled into a dynamic library to allow for multiple other programs or libraries to use this tool without compiling it into their binaries. To keep the interface generic, a few more functions are required than the C++ interface described above. Multiple models can be utilized simultaneously if desired. Each of these functions returns 0 if no errors occurred during runtime and 1 if a runtime error occurred along with printing the error message to the standard output pipe. An example is provided in the repository that demonstrates a typical implementation. The *load_model* function works the same as in the C++ interface.

1. The *register_output* function specifies which outputs should be available for retrieval after the model runs. Only outputs registered with this function can be accessed post-execution.
2. The *provide_input* function allows users to provide input to a specific input operation before executing the model. If multiple inputs are necessary to model execution, then this function should be called multiple times before calling to *predict*. This function takes as input the model ID with which to provide input, the operation name that is being provided input, the number of shape dimensions in the input, the shape dimensions, and finally the data itself. Each of these data, the number of dimensions, dimensions themselves, and data are provided as flat arrays.
3. The *predict* function takes all inputs provided from *provide_input* calls and all registered output names and runs the *predict* function in the C++ interface. All output is then stored for retrieval.
4. The *retrieve_output* function retrieves a user-specified output for a model that has been evaluated with the *predict* call. This function should be called multiple times if more than one output is requested.
5. The *retrieve_output_no_check* function operates similarly to the above *retrieve_output* function, however it does not check the shape and instead simply copies the data directly to the given pointer. This function only exists to speed up output retrieval if the original *retrieve_output* function is too slow to use.
6. The *delete_model* function works the same as in the C++ interface.

Figure 1 shows how to use these functions to load the model, get predictions, and finally delete the model from memory.

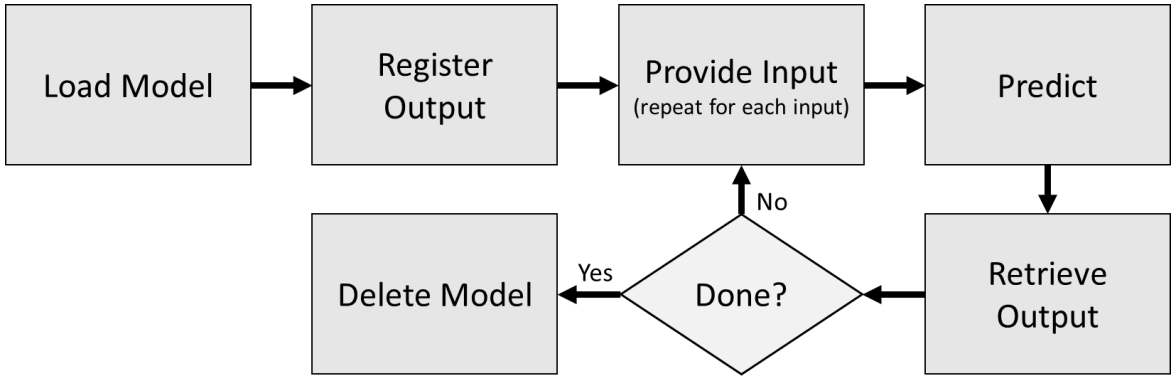


Figure 1.—Flow diagram of model use.

### Identifying Model Operation Names

The input and output operation names mentioned above are available to find using TensorFlow's built in SavedModel command line interface (CLI) tool. An example of viewing the default serving input and outputs, along with their expected shapes may look like:

```
$ saved_model_cli show –dir {save_path} --tag serve --signature_def serving_default
```

The save path is the path to the directory of the saved model. The directory should contain at least an "assets" directory, "saved_model.pb" file, and "variables" file.

The operation names can also be viewed using the SavedModel class in the tensorflow.core.protobuf.saved_model_pb2 module.

## Multiplatform Tools

This tool was developed for both Windows and Linux development and deployment. To do this, the CMake platform (Ref. 11) was used, which generates build files for various platforms and other tools. Detailed instructions on how to compile and run the software are present in the code repository's documentation, and briefly described here. Additionally, the repository documentation describes the prerequisites for building and using the library. The CMake tool will automatically download the C/C++ TensorFlow language bindings necessary for linking, however they may not initially be on the active library path. Additional information is given in the prerequisites section of the TFExec library documentation. If you are incorporating this library into another project, additional steps will need to be taken to appropriately link TFExec. Windows compiling is intended to utilize the Microsoft Visual Studio software. Once the project is made using CMake, the solution will need to be opened with Visual Studio and built.

## Future Work

This library is built to be both fast and maintainable, so that other developers can extend this work for future needs. However, there are several potential extensions. Automated testing could be implemented. Many tests were conducted by compiling and running example programs that each tested a specific feature of the library. However, this could be replaced by a unit testing framework like GoogleTest (Ref. 12) to ensure that changes to the code base are automatically tested in a continuous integration setup.

The recurrent neural network hidden state is incredibly important in the microsurrogate, as past strains and outputs affect the current output step. Tests should be specifically designed to ensure that the state is acting as expected, i.e. it is independent of samples but not of batches since the model needs to run at every time step and for each sample dimension, i.e. dimension corresponding to a fiber, have the internal state update (and not reset). To accomplish this, it may be necessary to save this state at each step and load it back into the model before execution of the next step. The TensorFlow documentation provides information on how to solve these issues in Keras models with cross-batch statefulness (Ref. 13). An alternative is to not save the hidden state in GPU memory and pass each previous timestep as input to collect the next output.

Finally, it is worth noting that while TFExec is capable of evaluating surrogates in parallel on GPUs, limitations from the calling software may limit this. For instance, in a finite element context, if a surrogate model is utilized to represent integration point behavior, parallel processing could be utilized to evaluate the surrogate across a batch of integration points. Some commercial software tools, however, only expose data for one integration point at a time. In this case, overhead associated from mapping between processors and GPUs could become significant. While this strategy is inefficient from a surrogate model

evaluation standpoint, there is still an added benefit for utilizing TFExec for cases where the baseline physics-based model is prohibitively expensive or unable to be evaluated due to computational limitations.

## Conclusion

TFExec provides a fast and efficient interface for integrating TensorFlow-based machine learning surrogates into Fortran-based simulation codes. By enabling low-latency inference, batch processing, and multiple models in memory, TFExec addresses key challenges in incorporating ML models into computational physics workflows. This capability significantly accelerates multiscale simulations while preserving accuracy, making high-fidelity modeling more practical. Beyond composite materials, TFExec has broader applications in computational fluid dynamics, thermomechanical analysis, and other physics-based simulations, ensuring that established scientific software can leverage modern ML techniques for improved performance and scalability.

## References


1. J. Stuckner, M. Piekenbrock, S.M. Arnold, and T.M. Ricks, "Optimal experimental design with fast neural network surrogate models," Comput. Mater. Sci., vol. 200, no. December 2020, p. 110747, 2021, doi: 10.1016/j.commatsci.2021.110747.
2. J. Stuckner, S. Graeber, B. Weborg, T.M. Ricks, and S.M. Arnold, "Tractable multiscale modeling with an embedded microscale surrogate," AIAA Scitech 2021 Forum, 1963, 2021.
3. P.A. Gustafson, E.J. Pineda, T.M. Ricks, B.A. Bednarcyk, B.L. Hearley, and J. Stuckner, "Convolutional Neural Network for Enhancement of Localization in Granular Representative Unit Cells," AIAA J., vol. 61, no. 4, pp. 1863–1875, 2023, doi: 10.2514/1.J061918.
4. A. Sorini, E. Pineda, J. Stuckner, and P. A. Gustafson, "A convolutional neural network for multiscale modeling of composite materials," 2021, doi: 10.2514/6.2021-0310.
5. B.A. Bednarcyk and S.M. Arnold, "MAC/GMC 4.0 User's Manual: Keywords Manual. Volume 2," 2002.
6. E.J. Pineda, B.A. Bednarcyk, T.M. Ricks, S.M. Arnold, and G. Henson, "Efficient Multiscale Recursive Micromechanics of Composites for Engineering Applications," Int. J. Multiscale Comput. Eng., vol. 19, no. 4, pp. 77–105, 2021, doi: 10.1615/INTJMULTCOMPENG.2021039732.
7. J.O. Andersson, T. Helander, L. Höglund, P. Shi, and B. Sundman, "Thermo-Calc & DICTRA, computational tools for materials science," Calphad, vol. 26, no. 2, pp. 273–312, Jun. 2002, doi: 10.1016/S0364-5916(02)00037-8.
8. R.T. Biedron et al., "FUN3D Manual: 13.7." 2020, Accessed: Mar. 18, 2025. [Online]. Available: http://www.sti.nasa.gov.
9. G.V. Candler et al., "Development of the US3D code for advanced compressible and reacting flow simulations," 53rd AIAA Aerosp. Sci. Meet., 2015, doi: 10.2514/6.2015-1893.
10. "serizba/cppflow: Run TensorFlow models in C++ without installation and without Bazel." https://github.com/serizba/cppflow (accessed Mar. 18, 2025).
11. "CMake - Upgrade Your Software Build System." https://cmake.org/ (accessed Mar. 18, 2025).
12. "google/googletest: GoogleTest - Google Testing and Mocking Framework." https://github.com/google/googletest (accessed Mar. 18, 2025).
13. "Working with RNNs | TensorFlow Core." https://www.tensorflow.org/guide/keras/working_with_rnns#cross-batch_statefulness (accessed Mar. 18, 2025).