Human Factors and Simulation, Vol. 180, 2025, 23–31
https://doi.org/10.54941/ahfe1006385

Simulation for Artificial Intelligence Modeling and Assessment

Daniel Barber and Lauren Reinerman-Jones

Southwest Research Institute, San Antonio, TX 78238, USA

ABSTRACT

Recent developments in autonomous fighter jets and concepts such as an autonomous wingman are pushing the boundaries for human-autonomy teaming in high-risk military flight operations. Many of these concepts explore the use of aides to accelerate pilot decision-making and reduce cognitive demands. Artificial Intelligence (AI) is a key enabling technology for decision support and automation of flight processes. Machine learning (ML) techniques are the primary method of training and validating modern AI models which requires representative data of increasing size. Acquiring this data is often a major blocker to the development of AI models. This becomes even more challenging when the target domain is aircraft for the U.S. Department of Defense (DoD) where existing datasets may be classified and/or inaccessible. A second requirement in the development of an AI model is an operational environment to integrate, execute, and assess performance in a closed-loop system. The ability to assess the AI safely in a live environment can also be difficult as when technology hasn't yet fully matured. To address these challenges in the development of AI models for a decision support system, Southwest Research Institute (SwRI) leveraged the U.S. Air Force Research Laboratory's (AFRL) Advanced Framework for Simulation, Integration, and Modeling (AFSIM) as a solution. This paper explores lessons learned in using AFSIM and its recently added support for the Python programming language to create a testbed for generating data of sufficient size to train Artificial Neural Networks (ANNs) to perform decision support and demonstrated in a closed-loop manner with new/live data.

Keywords: Modeling & simulation, Artificial intelligence, AFSIM, Spiking neural network, Neuromorphics, Human system teaming, Decision support, Human autonomy interaction

INTRODUCTION

The DARPA Alpha Dogfight Trials were broadcast live on YouTube in the middle of the pandemic, whereby Artificial Intelligence (AI) controlled aircraft faced-off against pilot-controlled aircraft in a virtual environment and the AI won. The eight teams of AI developers each took different approaches in algorithm generation and data training. The team that outperformed the others used deep reinforcement learning and ignored dogfighting doctrine. Although this algorithm won the fight, pilots voted a different model that included doctrine and matched their own training as most reliable and trustworthy. The Alpha Dogfight Trials fed the DARPA Air Combat Evolution (ACE) program, moving the algorithms from simulation

to real aircraft and focusing on the importance of objective operator trust calibrated with the dogfighting agents with the agents' intent. In 2022, AI flew a live F-16 in a simulated dogfight, ushering in the future of autonomy in tactical aerospace and paving the way for the next generation of AI, which are cognitive and neuromorphic AI.

Cognitive AI aims to replicate human thought processes and strives to emulate the way humans learn, think, and make decisions going beyond basic data processing and pattern recognition. Neuromorphic AI seeks to build computational systems consisting of a biologically inspired architecture, which involves creating Artificial Neural Networks (ANNs) with specialized hardware and software that emulate the behavior of biological neurons. Neuromorphics enables real-time, parallel processing and low power consumption with all the learning power of other cognitive AI.

Techniques in Neuromorphics include spiking neurons rather than traditional artificial neurons that process continuous-valued inputs within what are called Spiking Neural Networks (SNNs). SNN neurons communicate using discrete spikes or pulses of activity, similar to the way neurons in the brain communicate through action potentials (Indiveri et al., 2011). Neuromorphics enables real-time, parallel processing. Algorithms can be implemented using software simulations that run on conventional hardware; however, they are optimal when using specialized hardware (chips) that do not require Graphics Processing Units (GPUs). Neuromorphics can be implemented alone or complementary to traditional AI. Neuromorphics are beneficial to Size, Weight, and Power (SWaP) constrained environments and when optimization of processing latency is critical (Schuman et al., 2017). Neuromorphics is still a nascent technology, requiring research to understand its' capabilities and potential. The Southwest Research Institute is exploring the potential of Neuromorphics to bring enhanced capabilities such as 3D correlation and tracking, signal classification in electronic warfare, visual perception, and tactical decision making/support to constrained hardware environments (e.g., military aircraft, small drones). The current case study seeks to explore the use of Neuromorphics to create a tactical decision support system for military pilots. Developing an SNN to achieve this objective requires two things: data sets large enough to build and evaluate models and an operational environment to integrate, execute, and assess performance of models in a closed-loop environment.

AIRCRAFT MODELING & SIMULATION FOR AUTONOMY

The state-of-the-art in AI applications apply Machine Learning (ML) techniques with massive data sets to train and validate models. One of the most popular Large Language Models today, ChatGPT, leverages data from publicly available sources, academic research, user-generated content, third-party partnerships, and human trainers, (OpenAI, 2024). Gathering the necessary data for model development using ML becomes increasingly difficult within military domains such as aircraft for the U.S. Department of Defense (DoD) where existing data sets are classified or simply unavailable. Moreover, even if existing data sets were made available, they would

not cover the full spectrum of situations and edge-cases aircraft may encounter. For example, encounters with different aircraft or tactics yet to be employed in real environments. This limitation is further exacerbated when exploring concepts where some technologies have yet to be fully developed or deployed (e.g., autonomous swarms, combat wingman). To develop an SNN for a decision support system, there exists a need for simulations capable of delivering the appropriate level of fidelity for sensors and the environment matched to real-world systems while also able to cover the necessary spectrum of scenarios for data set generation. In addition to data set generation, there must be methods to integrate, exercise, and assess the SNN (or any other AI approach) built from this data within the simulation environment with new data generated in real-time. To address these two challenges for AI modeling, the team leveraged the U.S. Air Force Research Labs (AFRL) Advanced Framework for Simulation, Integration, and Modeling (AFSIM).

Supporting Decision Making Autonomy

AFSIM is an object-oriented C++ library used to create simulation modeling platform interactions in a geographic context. AFSIM incorporates top-level objects within the framework, called platforms, representing systems and their attached attributes (e.g., sensors) supporting advanced modeling of ground, air, space, surface, subsurface vehicles in addition to buildings or living things, and interactions among platforms including sensor detections, collisions, and communications to name a few. AFSIM's core applications deliver weapon engagement analysis support, mission analysis/baseline simulation application, sensor coverage and antenna gain plot creation, weapon model development support, and post processing and report generation, (Defense Systems Information Analysis Center, 2023). Moreover, AFSIM provides advanced scripting capabilities for generation of scenarios modeling aircraft, such as a generic fighter or unmanned air system (UAS), for generation and capture of sensor inputs used in the decision-making processes of a pilot, operator, or in this instance a decision support system.

A key benefit for use of AFSIM for creation of a decision support system is the incorporation of Endsley's model of situation awareness (SA), which facilitates use of an Boyd's observe, orient, decide, and act (OODA) loop, (Endsley, 1995; McIntosh, 2011). This enables simulated aircraft information processors to consume data in ways analogous to how human's make decisions. As a result, AFSIM delivers a framework for capture of data, reasoning on it, and implementing actions which directly supporting development of decision support aides for a pilot in addition to creating models for processing sensor data for sensemaking, classification, signal matching, and more. AFSIM supports a "closed-loop" decision making cycle following components of an OODA loop, see Figure 1.

These components include situation awareness processing and rule-based AI techniques (e.g., behavior trees, finite state machines) that can be substituted or paired with other approaches. For example, a decision support SNN running on Neuromorphic hardware consuming the same inputs as the

SA processor (Observe) and learns relationships between those inputs and possible actions (Orient) to then map the state of the environment to actions (Decide) for implementation (Act).

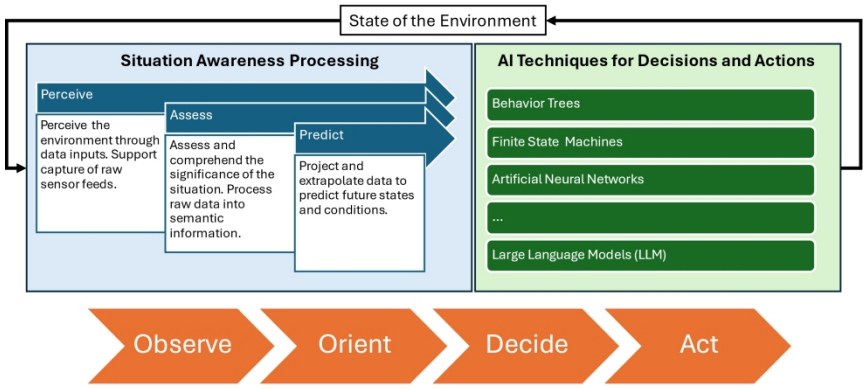

Figure 1: Information processing approach using AFSIM.

Integrating Data Capture Tools and AI Applications

A recent development in feature releases of AFSIM facilitates rapid data capture and integration of AI through the addition of support for the Python programming language. Although AFSIM's native scripting language provide means of data logging and data links to external software, there are limitations and constraints for developers. For example, using AFSIM native scripts, one can write raw data in binary formats to files on disk, but this requires creating a duplicate implementation of a reader in different programming languages used in third-party software. Any changes in data requirements or format would also require updates to both the AFSIM script and third-party applications as well, limiting re-use of code. Second, data links into AFSIM are primarily geared toward sensor or control inputs, existing messages, and not custom commands (i.e., decision support recommendation) without building new C++ plugins. Although functional, this approach has limited flexibility for changing what kind of data needs to be collected, when it can be shared, injecting new data into AFSIM, and doing so in a way compatible with best-in-class AI programming tools and techniques written in Python. Therefore, leveraging the power of Python support enables reuse of code between third-party AI applications and AFSIM simulations. AFSIM includes two methods for using Python: 1) AFSIM in Python and 2) Python in AFSIM. The first approach, AFSIM in Python, supports running AFSIM simulations directly from Python, while the second method, Python in AFSIM, enables execution of Python modules from within AFSIM applications. For this effort, the team selected the second approach as it had more flexibility, better re-use of Python code, and direct use of AFSIM tools and visuals to demonstrate capabilities.

Using the aforementioned "Python in AFSIM" approach, several Python modules and tools were created to 1) enable capture of aircraft perception

data for AI development and 2) sending decision support information for use within the simulation. The industry standard and open-source universal messaging library ZeroMQ (ØMQ) was selected for inter-process communication between standalone applications and AFSIM due to low latency and asynchronous capabilities, (ZeroMQ, 2024). An AFSIM scripting module was created for re-use across AFSIM scenarios that loads Python modules dynamically, injecting simulation state and events into them for sending data to AI programs or applications recording data in bulk for use in training SNN models. This AFSIM script also checks for incoming messages from external programs (i.e., neuromorphic decision aide) and passes them to simulated entities. This approach resulted in minimal programming within AFSIM outside of scenario creation as all code driving interactions occurred withing re-usable Python modules, Figure 2.

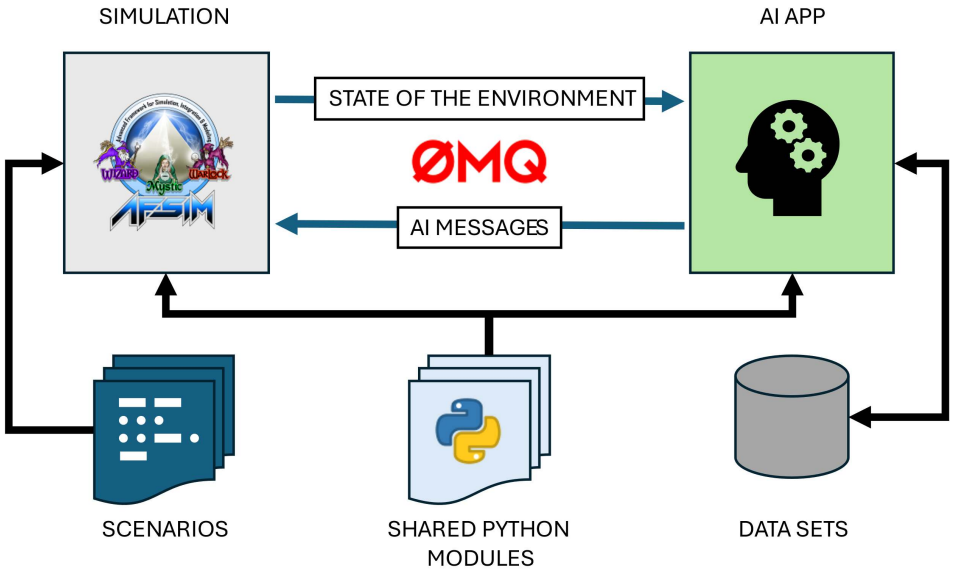

Figure 2: Integration of AFSIM, AI, and Python software.

Simulating Aircraft Scenarios for Model Development

Building upon the infrastructure previously described, the team constructed a simulation of a pilot flying a generic fixed-wing aircraft to generate data and provide a closed-loop testbed for creation of tactical decision support AI with AFSIM. The simulated pilot/aircraft supported three decisions: ROUTE, FIGHT, and EGRESS. The ROUTE task instructs the pilot to follow a pre-defined mission route towards a target. The FIGHT task suggests the need to engage enemy aircraft. Finally, the EGRESS task informs the pilot that it should return to their home location as quickly as possible. For all tasks, if within range of a target, the pilot can be notified of ability to launch a GPS navigated bomb to destroy it. For all scenarios, the primary mission goal was to destroy a ground target and survive any engagements with

enemy aircraft. Several unique scenarios were created to capture an equal number of data samples for each command under a variety of different conditions. To best support ML using supervised learning for an AI model (i.e., SNN), it is important that the number of each decision type were equated in generated data sets to avoid biasing the model toward one decision or another while supporting generalizability. Data set ground truth labels, what the correct decision should be at a given time, was determined using pre-defined heuristics within AFSIM scripts that controlled aircraft behavior. The heuristics acted as "Subject Matter Expertise" as to what the right decision should be at each time step. Therefore, in the absence of AI control, AFSIM scripted logic controls aircraft behavior and defines "ground truth." Ground truth data is necessary so that AI can "learn" what the output should be for a given input. The final simulation implemented behaviors using the scripted "ground truth" logic or decisions resulting from a trained SNN running on Neuromorphic hardware during runtime.

To deliver variation and required data distribution, combinations of scenarios including either one versus zero (1v0), one versus one (1v1), or one versus two (1v2) enemy aircraft were created. These combinations were selected to support capture of unique inputs and transitions capable of triggering a change in decision recommendation while also achieve an appropriate distribution of each decision state. For example, in 1v0 scenario, with no enemy aircraft, the FIGHT task would never occur resulting in only the ROUTE and EGRESS recommendations occurring. However, in a 1v1 situation where the aircraft believed it could successfully engage and still bomb the ground target you would capture all three categories (ROUTE->FIGHT->EGRESS). In 1v2 scenario the AI decision aide (or script) may decide it is outmatched and recommend EGRESS or try sneak in and bomb the target before flying home, each of which results in a different duration of ROUTE and EGRESS decision states.

Table 1: Sample scenarios with amount of time within a scenario each decision is recommended.

# Friendly vs # Enemy	Scenario Description	ROUTE	FIGHT	EGRESS
1v0	ROUTE to target, drop bomb, then EGRESS home.	50%	0%	50%
1v1	ROUTE to target. Switch to FIGHT defeat enemy aircraft, resume ROUTE, then EGRESS home after bombing target.	33%	34%	33%
1v1	ROUTE to target. Switch to FIGHT defeat enemy aircraft, EGRESS when/if bingo fuel/weapons.	32%	50%	18%

Continued

Table 1: Continued

# Friendly vs # Enemy	Scenario Description	ROUTE	FIGHT	EGRESS
1v2	ROUTE to target. EGRESS if unable to bomb without engagement of two (2) enemy aircraft.	25%	0%	75%
Totals		32%	36%	32%

LESSONS LEARNED

There were several challenges encountered during the implementation of the simulation approach presented using AFSIM. The first challenge was related to stability and early stage of documentation surrounding Python support in the current feature releases of AFSIM. As previously described, AFSIM supports either "Python in AFSIM" or "AFSIM in Python." Initially, the latter method was explored as a means to directly interact with AFSIM libraries, automate simulation execution, and capture data from within a Python environment, but this approach did not support access to all data due to how C++ interface bindings are exposed to Python. This challenge became more pronounced as development proceeded and the need for additional data was identified to support training an SNN. Therefore, the "Python in AFSIM" approach was selected as any data exposed within an AFSIM script can be explicitly delivered to a Python module. An additional result of using this approach was the need to convert AFSIM data into re-usable Python data structures that could be shared using inter-process communication techniques (i.e., ZeroMQ). To support both data set logging and serialization/deserialization steps needed for data interchange, Google Protocol Buffers were leveraged, (Google, LLC., 2024). Protocol Buffers allow developers to define data types as "messages" independent of a specific programming language. A Protocol Compiler application called "protoc" reads these definitions to generate versions of the data types in multiple programming languages (e.g., C++, Python, Java). This approach supported generation of data types for use in AFSIM, Python, and future applications written in other programming languages while including built-in capabilities for data interchange with ZeroMQ and data set logging. Although more challenging to use due to the need for inter-process communication in this application, there was an added benefit in that AFSIM's tools were leveraged to demonstrate AI with real-time 3D visualizations. Without this capability, an additional visualization would be needed to showcase the resulting integrated solution.

Other issues to overcome using Python in AFSIM were early stages of documentation and software issues. As a preface, the authors do not want to overstate the significance of these issues. A feature release of AFSIM was used for the current effort (version 2.2408.0) and documentation and software bugs are very likely to be resolved in future stable releases. As a newly developing feature within AFSIM, the documentation on how to enable Python functionality is distributed across different sets of documentation and

examples and not in a single location. Second, two methods of executing Python code in AFSIM scripts are documented, but one of the two methods does not currently work in all AFSIM applications. For example, AFSIM includes two main programs for simulation execution: Mission and Warlock. Mission is a command line tool that runs scenarios in hyper time or wall-clock time. Warlock has a graphical user interface (GUI) and visualizes the scenario as it runs using wall-clock time. The authors were only able to get the Mission application to fully supported both Python approaches, while Warlock would "crash" when using a method where Python code is written directly within the scripts. Once discovered, this challenge was overcome using the alternative Python integration method using native script objects.

CONCLUSION

In summary, several lessons were learned related to how AFSIM can support AI development and integration with third party software applications like those in this project. Although new and not free of errors, the recent inclusion of Python into the AFSIM ecosystem delivers previously unseen simulation flexibility and extensibility. Previous approaches to extending AFSIM required development of C++ plugins which bring their own challenges with respect to debugging and cross-platform support (i.e., Windows vs Linux OS). Python is much more platform agnostic and is the state-of-the-art/best practice programming language for AI research and development. The present effort greatly benefited from the ability to rapidly add new functionality, change data capture needs, and integrate different AI techniques for creation of a tactical decision support system. Moreover, these capabilities can be quickly adapted to different AI applications in other domains AFSIM supports: (e.g., ground, sea, space, track correlation, sensing). Finally, AFSIM supports both UNCLASSIFIED and CLASSIFIED efforts, making it an ideal starting point for transitioning products to existing and future Department of Defense (DoD) customers. A specific recommendation for future work is the development of tools to support more automated scenario generation for creating data sets. For example, as a proof of concept and approach for model development, many of the scenarios were very similar with primary changes being to starting locations of aircraft, target, and route. It's possible a "template" scenario could be created and then used to produce multiple unique situations through a "random" modification of a core set of parameters (e.g., initial location and speed, number of weapons). This type of tool, integrated with AFSIM through Python modules developed under this effort, paired with Monte Carlo simulation capabilities of AFSIM, could then generate, run, and modify the template to produce a significantly larger number of behaviors and events for machine learning data sets.

ACKNOWLEDGMENT

The authors would like to acknowledge the contributions of Brian Millikan and Howard Yanxon in the advancement of this effort.

REFERENCES

Defense Systems Information Analysis Center. (2023, 10 4). *AFSIM*. Retrieved from DSIAC: https://dsiac.dtic.mil/models/afsim/.

Endsley, M. R. (1995, March). Toward a Theory of Situation Awareness in Dynamic Systems. *The Journal of Human Factors and Ergonomics Society, 37*(1), 32–64. doi: 10.1518/001872095779049543.

Google, LLC. (2024). *Protocol Buffers - Google's data interchange format*. Retrieved 01 03, 2025, from https://github.com/protocolbuffers/protobuf.

Indiveri, G., Linares-Barranco, B., Hamilton, T. J., van Schaik, A., Etienne-Cummings, R., Delbruk, T.,... Boahen, K. (2011). Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience, 5*. doi: 10.3389/fnins.2011.00073.

OpenAI. (2024, 11 1). *How ChatGPT and our foundation models are developed*. Retrieved 02 12, 2025, from OpenAI: https://help.openai.com/en/articles/7842364-how-chatgpt-and-our-foundation-models-are-developed.

Schuman, C., Potok, T., Patton, R., Birdwell, D., Dean, M., Rose, G., & Plank, J. (2017, May 19). *A Survey of Neuromorphic Computing and Neural Networks in Hardware*. Retrieved 01 09, 2025, from https://arxiv.org/abs/1705.06963.

ZeroMQ. (2024). *ZeroMQ*. Retrieved 02 12, 2025, from ZeroMQ: https://zeromq.org/.