Project Pipeline:

Concluded on 18—01—2025: This study blends math, automation, and flow physics to explore how Von Kármán ogives behave across subsonic to hypersonic regimes. The objective of this project titled **“Flow Study of Von Kármán Ogive Nose-Cones”** was to predict drag performance, shock formations, and flow separation of same profile with Fitness Ratios – 1,1.5, 2, 2.5, 3, 3.5 4 and 4.5 under Mach numbers – 0.3, 0.5, 0.8, 1, 1.5, 2, 3, 4 and 5.

The project started with generating ogive coordinates using a Python script. This script takes inputs for nose cone length and fineness ratio (L/D) and returns X-Y-Z coords of the 2D profile. The ogive shape was calculated using the analytical equation for Von Kármán profiles, known for optimal supersonic drag characteristics. The X-axis represented axial length, Y was radial distance from center-line, and Z was kept zero to represent a planar profile. These values were then exported as a formatted .txt file for CAD import. You can find the python code in the repository.

SolidWorks was used to construct the geometry. The point file was imported as a 2D sketch using “Curve from XYZ points tool” and revolved about the X-axis to form a smooth 3D solid of the ogive. The surrounding flow domain was also modeled—large enough to let shocks evolve without being affected by boundaries. Only a quarter of the domain was modeled, taking advantage of symmetry to cut down mesh size and sim time.

The model was saved as a .STEP file and imported into Salome, an open-source CAD + mesh platform. Salome provided full control for clean meshing. Inside Salome, I used the 1D-2D-3D Netgen algorithm. A small refinement body was created near the body to serve as a trigger for local mesh density. This ensured we had a fine mesh around the nose while keeping the outer domain coarser. Tetrahedrals were used for the domain, with refined near-wall cells.

To handle wall effects and viscous layers, I used extrusion techniques to add prisms around the ogive surface. Layer height, count, and growth rate were set manually to match y+ needs for the k-ω SST turb model. Target y+ was <1 to ensure the low-Re behavior was captured well. The transition from prism layers to the outer tet mesh was kept smooth.

After meshing, the grid was exported as .unv. I used SimFlow 5.0, a GUI layer for OpenFOAM, to manage the simulation. SimFlow recognized the boundary groups from the unv file and handled the conversion internally.

I used rhoCentralFoam solver due to its capability to handle high-speed compressible flows and shocks. The fluid was set as ideal gas (air), with $\gamma = 1.4$. Inlet BC was set to supersonicInlet where Mach, P, and T were specified. At the outlet, pressureOutlet was used. The ogive wall used noSlip BC. Symmetry planes were defined for the domain faces to mimic the rest of the 3D domain.

For turbulence, I picked the SST model, known for its performance under adverse pressure gradients and strong shock/boundary layer interaction. It uses a blend of k-ε (in the free stream) and k-ω (near the wall) to handle different zones efficiently.

Numerical schemes used were mostly 2nd-order upwind for convective terms and linear for diffusion. Time stepping was implicit with fixed Courant number. Under-relaxation was applied for U, P, and k/ω to ensure stability. Convergence was tracked using residual plots. Target convergence were set to 1e-5 for momentum and turbulence residuals.

After running the base shape, I re-used the Python script to regenerate points for other fineness ratios. The script only needed length and L/D ratio as inputs. It re-calculated base radius, ran the equation, and generated new point files. These were brought back into SW, modeled into new domains, and followed the same pipeline—export to Salome, mesh, convert, and simulate.

This parametric approach allowed a fast sweep across geometric configs. Using Salome for meshing was critical—Netgen gave a great balance between automation and control. Having prism layers with manual extrusion instead of fully unstructured meshes made boundary resolution sharper and helped turb model accuracy. ParaView came in handy to ensure each mesh looked clean, especially near the nose tip where pressure gradients were highest.

SimFlow provided a consistent setup for solver selection, BCs, and convergence control. However, for some advanced tweaks, I still accessed the system/ and constant/ dirs directly to change schemes and solution control parameters.

From a tools perspective, each software played a clear role: Python for ogive generation, SW for CAD, Salome for generating mesh, SimFlow for physics and solver setup, and ParaView for visualization.

SimFlow's GUI was stable across all runs, and even handled hypersonic setups without crashing. rhoCentralFoam solver proved resilient. The only time I faced issues was at Mach ~0.8, where convergence slowed due to transitional flow conditions. Adding pseudo-transient terms helped stabilize those cases.

The project gave me strong insight into shock behavior, mesh strategies, solver tuning, and automation of CAD workflows. It also showed how to think parametrically—once the base pipeline was set, running more configs was just a matter of feeding in new inputs.

Using SW for modeling, Salome for high-quality tetra + prism meshing, SimFlow for solver integration, and ParaView for mesh/flow validation ensured the pipeline was robust and reproducible. Post-proc in ParaView let me extract slices, velocity contours, pressure plots, and even animation of shock formation over timesteps. Exporting graphs directly from ParaView saved time.

Turbulence modeling via SST was stable, even at high Re numbers. For $M > 3$, I considered using DES or LES, but they would've required much finer meshes and smaller timesteps. For now, RANS was sufficient.

More than just results, I learned how to own the full pipeline—from scripting and geometry, to meshing, solving, and setup management. That's what makes flow studies powerful—not just solving equations but understanding which ones, when, and how.

Every run built my confidence in tweaking mesh parameters, solver setups, and interpreting physical behavior. Instead of staring at charts, I saw how changes in fineness ratio altered flow behavior directly.

This wasn't just about the study. It was about learning a repeatable method to analyze flow around any high-speed body. From missiles to re-entry vehicles, the same principles apply—refined mesh, smart BCs, and solver tuning.

Overall, the project made me better at design + analysis workflows. It showed me how good results need both technical rigor and flexibility to change things quickly when problems arise. Whether it's convergence issues, mesh exports, or BC tweaking, every part matters.

18—01—2025