


## Article

# An Automated Computational Fluid Dynamics Workflow for Simulating the Internal Flow of Race Car Radiators

Francesco Mangini <sup>1,\*</sup>, Matteo Vaccalluzzo <sup>1,†</sup>, Eugenio Bardoscia <sup>2,†</sup> and Andrea Bortoli <sup>2,†</sup>  
and Alessandro Colombo <sup>1,\*</sup> 

<sup>1</sup> Department of Engineering and Applied Sciences, University of Bergamo, Via Einstein, 24044 Dalmine, Italy

<sup>2</sup> Tatuus Racing S.p.a., Via Juan Manuel Fangio, 20045 Lainate, Italy; eugenio.bardoscia@tatuus.it (E.B.); andrea.bortoli@tatuus.it (A.B.)

\* Correspondence: francesco.mangini@unibg.it (F.M.); alessandro.colombo@unibg.it (A.C.)

† These authors contributed equally to this work.

**Abstract:** In this article, we present a software tool developed in Python, named T-WorkFlow. It has been devised to meet some of the design needs of Tatuus Racing S.p.a., a leading company in the design and production of racing cars for the FIA Formula 3 Regional and Formula 4 categories. The software leverages the open-source tools OpenFOAM and FreeCAD to fully automate the fluid dynamics simulation process within car radiators. The goal of T-WorkFlow is to provide designers with precise and easily interpretable results that facilitate the identification of the geometry, ensuring optimal flow distribution in the radiator channels. T-WorkFlow requires the radiator's geometry files in .stp and .stl formats, along with additional user inputs provided through a graphical interface. For mesh generation, the software leverages the OpenFOAM tools *blockMesh* and *snappyHexMesh*. To ensure uniform mesh quality across different configurations, and thus, comparable numerical results, various pre-processing operations on the specific geometry files are needed. After generating the mesh, T-WorkFlow automatically defines a control surface for each radiator channel to monitor the volumetric flow rate distribution. This is achieved by combining the OpenFOAM command *topoSet* with specific geometric information directly obtained from the radiator's CAD through FreeCAD. During the simulation, the software provides various outputs that automate the main post-processing operations, enabling quick and easy identification of the configuration that ensures the desired performance.



**Citation:** Mangini, F.; Vaccalluzzo, M.; Bardoscia, E.; Bortoli, A.; Colombo, A. An Automated Computational Fluid Dynamics Workflow for Simulating the Internal Flow of Race Car Radiators.

*Appl. Sci.* **2024**, *14*, 9930. <https://doi.org/10.3390/app14219930>

Academic Editor: Wei Huang

Received: 21 September 2024

Revised: 21 October 2024

Accepted: 24 October 2024

Published: 30 October 2024



**Copyright:** © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

**Keywords:** computational fluid dynamics; OpenFOAM; automated workflow; race car radiators

## 1. Introduction

Computational fluid dynamics (CFD) is an essential tool for the design of racing cars. It is used for designing external aerodynamics but also for evaluating the performance of internal components, such as radiators. The CFD design practice typically comprises four phases: (i) geometry pre-processing; (ii) generation of the computational grid; (iii) numerical simulation; (iv) results visualization and analysis, i.e., post-processing. This workflow often relies on highly specialized commercial software, and each phase may require manual intervention by the designer, consequently making the overall process costly in terms of both time and computational resources. Finally, interpreting and synthesizing numerical simulation results can be challenging and requires a deep understanding of fluid dynamic processes to draw meaningful conclusions.

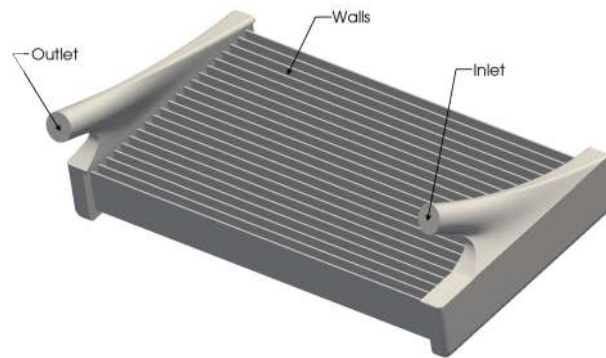
These considerations explain why some companies are interested in developing fully customizable open-source tools to enable the complete automation of the entire numerical simulation process with minimal effort from the user. These tools can also be designed to produce output that is already interpreted and synthesized efficiently, allowing the designer to make critical decisions quickly. As an example of this practice, the paper presents a utility, named T-WorkFlow, that has been developed to meet some of the design needs of

Tatuus Racing S.p.a. [1], a leading company in the design and production of racing vehicles for the FIA Formula 4 [2] and Formula 3 Regional [3] categories. Similar tools may contribute to both economic and time savings during the design phase by fully managing and standardizing the tasks involved in the process. T-WorkFlow has been developed to fully automate the internal fluid dynamics simulation of the radiators installed in Tatuus cars, further expanding the original work proposed for the company in [4]. In fact, the present effort focuses on streamlining the entire CFD process, intending to provide a customized, automated tool for the company to be used during the early stages of design. The ultimate objective is twofold: first, to minimize the time required for each numerical test by enabling the assessment of more configurations throughout the project, and second, to decrease significantly the chance of human error when performing repetitive tasks. Particular emphasis is placed on developing an automated meshing strategy that ensures consistent grid quality in different configurations. As is explained in Section 3, this requirement implies a number of pre-processing operations on the specific geometry. The tool, through the automation of the CFD analysis, allows the results to be standardized, making comparisons between different configurations more reliable and facilitating the work of designers in identifying the set-up that ensures the best flow distribution among the radiator channels, delivering accurate and easily interpretable results.

T-WorkFlow takes advantage of the widely known open-source OpenFOAM [5] CFD solver for the flow simulation. In the scientific literature, OpenFOAM [5] is widely employed in various applications, including external aerodynamics simulations [6–8], internal flow simulations [9–11], and even in the medical field [12]. Although the capabilities offered by open-source CFD solvers such as OpenFOAM [5] are undoubtedly significant and are contributing strongly to the advancement of fluid dynamics simulation in academic and industrial settings, the proper use of these tools often requires an experienced user. In this regard, T-WorkFlow completely manages the interface with the CFD solver, requiring the designer to input only a few simple parameters through a dedicated graphical interface. This approach simplifies and streamlines the interaction with the mesh generator and CFD solver, minimizing the manual intervention required from the designer. T-WorkFlow not only automatically generates the computational grid and the set-up for the flow simulation but also produces dedicated output to monitor, for example, the solution evolution at run-time. In fact, the tool can “capture” a cross-sectional control surface for each radiant channel, regardless of the CAD reference system, and monitor the volumetric flow rate distribution in each of them, along with the corresponding standard deviation value. The main post-processing operations are also fully automated, and the tool provides quantitative results to help the designer identify the radiator configuration that meets the desired performance. This includes achieving an almost uniform distribution of flow across channels for a given flow rate value at the inlet. This condition is of utmost importance as it enhances the overall efficiency of the heat transfer.

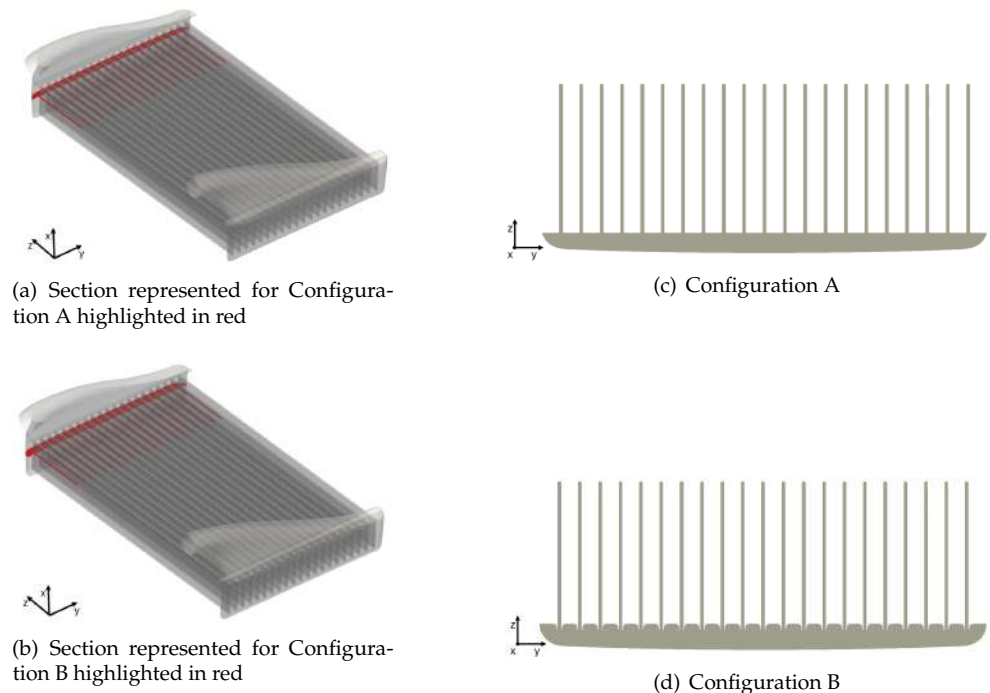
The radiator case study presented in this paper mainly serves as an example of the potential applications of similar automation tools in a real industrial design process. In fact, the main goal of this work is to gain and share experience in developing a, possibly open-source, complete CFD workflow that is as general as possible and that can be easily “extended” to the simulation of other car details, e.g., external aerodynamics, in the near future. It is important to note that this work focuses on process optimization; thus, assessing the accuracy of the numerical results is not the primary objective, as the company has previously established a correlation with the experimental data. This prior knowledge undoubtedly allowed for the confident selection of physical models and numerical strategies used in this research. In developing T-WorkFlow, we intentionally limited the number of user inputs as much as possible, e.g., by automating the choice of the parameters needed for mesh generation, to make the tool more user-friendly and suitable for the “repetitive application” in batch simulations and/or optimization cycles. Together with the few numerical user inputs to be set through a graphical interface (cf. Section 3.1), T-WorkFlow requires the CAD model of the radiator to be simulated in .stp format, along with the triangulations of the inlet, outlet, and wall surfaces in .stl format, as shown in Figure 1. These files are

essential for both mesh generation and automatically setting the boundary conditions in the OpenFOAM [5] dictionaries.



**Figure 1.** Boundary surfaces for a generic radiator model.

In this paper, we evaluate the performance of T-WorkFlow by simulating two radiator models. These are completely identical from the geometrical point of view, with the exception of the junction area between the channels and the inlet and outlet tanks. Configuration A exhibits radiant channels smoothly connected to the external surface of the tanks, while Configuration B is characterized by channels “partially immersed” in the water tanks. These differences are illustrated in Figure 2 through a cross-sectional detail near the tanks of the flow domain. Actually, each radiator mounted on Tatuus cars has a tank-channel junction similar to that of Configuration B. Configuration A, on the other hand, is a possible geometric simplification that leads to a reduction in the number of mesh elements and thus to computational savings; see Section 4. Therefore, the specific objective of the comparison presented in this paper is to understand whether the simplified Configuration A can be effectively used to study the performance of this radiator model without deviating significantly from the numerical results obtainable from the “real” Configuration B. Computational savings ensured by the use of Configuration A in place of Configuration B represent an aspect of foremost importance considering the high number of simulations and the tight deadlines characteristic of the design phase. To this purpose, in Section 4, performance indicators for these two configurations are presented and compared.



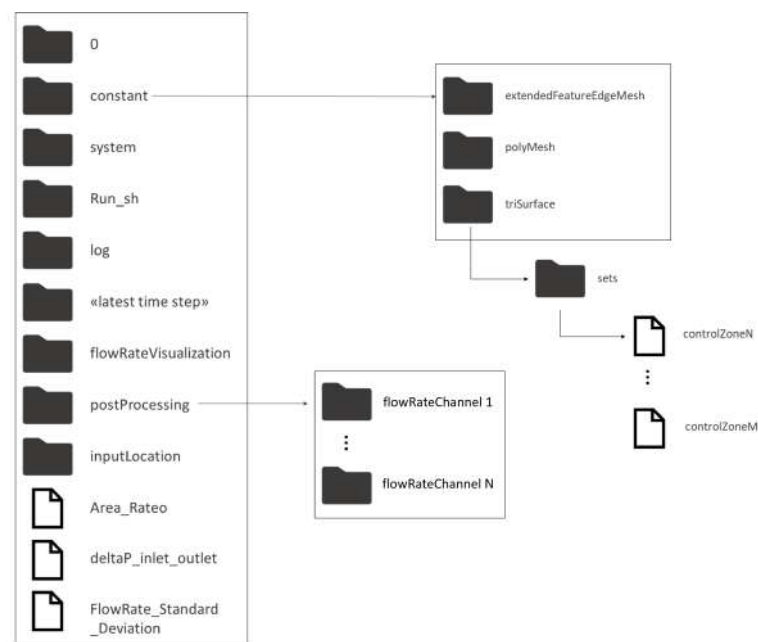
**Figure 2.** Cross-sectional detail near the tanks of the flow domain.

Let us finally mention that the CFD simulations carried out through T-WorkFlow in this work focus on the hydraulic performance of the different radiator configurations and not on their thermal exchange (“cold” simulations) as is usually implemented in the initial design phase.

## 2. Tools of the Trade

T-WorkFlow is a tool written in Python3 [13] that takes advantage of the open-source parametric 3D modeler FreeCAD (version 0.19 or later) [14] and the open-source CFD solver OpenFOAM10 [5]. We opted for the Python programming language as it readily provides access to many libraries. These are powerful and versatile tools that greatly accelerate development, minimize code duplication, and provide numerous advanced features without having to implement them from scratch. The main libraries employed by T-WorkFlow are as follows: (i) tkinter [15], available through the PSF License and used for developing a simple graphical interface for user interaction; (ii) numpy-stl [16], available through the BSD License and used to handle the .stl files provided as input to the tool, especially during the pre-processing phase; (iii) matplotlib [17], available through the Matplotlib license agreement and used to produce the summary plots provided as output by the tools.

Once the execution of T-WorkFlow is started, the first operation performed is the creation of a subdirectory within the working directory dedicated to the execution of the OpenFOAM simulation, named by default `Radiator_casenum`. The “casenum” token is simply an incremental index that starts at 0 and is updated each time the program is run to prevent the unintentional overwriting of previously run simulations. Within the case folder, the directory structure shown in Figure 3 is created, consisting of



**Figure 3.** Structure of the folders generated by T-WorkFlow.

- the subfolder **log**, where the log files generated during the simulation are saved;
- the subfolder **run\_sh**, where the shell scripts (.sh) generated by the tool and needed for its execution are saved;
- the subfolder **InputLocation**, where some text files needed for the workflow execution, as shown in Section 3, are saved;
- the subfolder **flowRateVisualization**, where the plots showing the volumetric flow rate distribution over the radiant channels at different iterations of the CFD solver are stored as images in .png format;
- the basic folders for an OpenFOAM case, i.e.:

- the subfolder **0**, containing the initial and boundary conditions;
- the subfolder **constant** contains files related to the computational grid, physical properties of the fluid (such as viscosity), and the turbulence model used for the simulation;
- the subfolder **system**, where all the dictionary files needed for the CFD solver set-up are saved.

In order to extract specific geometric information from the CAD model of the radiator, cf. Section 3.1, which is essential to ensure the correct functionality and flexibility of the workflow, we chose to use the FreeCAD modeler during the development of T-WorkFlow.

FreeCAD comes with its own Python console, allowing for some direct CAD operations on the virtual model via the command line. The execution of a specific set of operations can also be coded in Python language in dedicated files called macro. T-WorkFlow takes advantage of this feature by generating the `InputExtraction.FCMacro` file, which contains the instructions for extracting all the relevant geometric information from the model. For this purpose, T-WorkFlow directly opens the FreeCAD graphical interface to load the radiator `.stp` model and allow the user to execute the generated macro. The extracted data, which are discussed in detail in Section 3.1, are then automatically saved in the `InputLocation` folder.

The mesh can be either imported from external software, e.g., STAR-CCM+ [18], ANSYS [19], or Gmsh [20] or it can be generated using OpenFOAM internal tools like `blockMesh` and `snappyHexMesh`. Although the first option may make it easier to obtain a high-quality computational mesh, in order to achieve a fully automated process based on open-source tools, the second approach was followed for meshing. The grid generation strategy used in T-WorkFlow, which relies on a proper combination of `blockMesh` and `snappyHexMesh`, is detailed in Section 3.3.

T-WorkFlow involves executing multiple OpenFOAM programs from a Linux terminal, and each execution requires its own dictionary file that contains the desired specifications, e.g., the refinement level of a particular region of the mesh in the `snappyHexMeshDict`. To accomplish this, the tool first generates, for each command to be executed, its own dictionary file within the `system` folder and a dedicated `.sh` file containing the command-line instructions for its execution in the Linux terminal. The Python's `subprocess` module then schedules the execution of the command through the corresponding `.sh` file. Upon completion of each command, the corresponding `.sh` file is moved to the `run_sh` folder, while the log file generated by that specific operation is moved to the homonymous folder. In this way, the user can easily monitor the execution of each task by examining the log files. For executing OpenFOAM, it is essential that both `fvSchemes` and `fvSolution` files are properly set within the system directory, together with the other dictionary files. The `fvSchemes` and `fvSolution` files include numerical details, such as the methods used to discretize the terms of the governing equations and the algorithms used to solve the discrete system and advance the solution in time. Both these files are automatically generated by T-WorkFlow and their set-up is based both on the authors' experience in simulating similar flow problems with OpenFOAM and on the original work presented in [4].

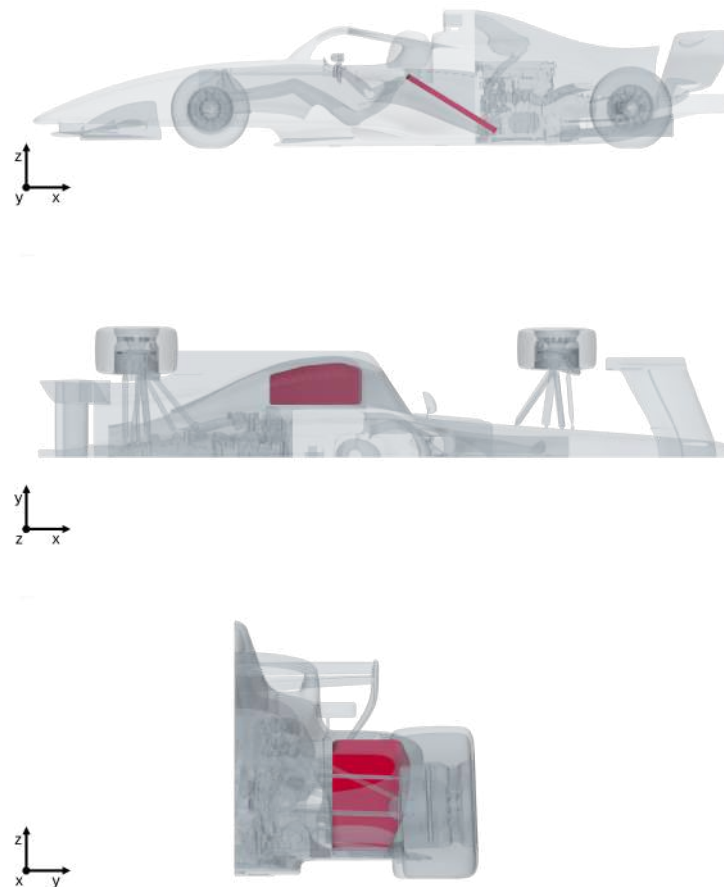
### 3. Workflow Description

This section describes the customized but rather general workflow developed in this work for CFD simulations, with a special focus on user/software interaction. As discussed in the previous section, the T-WorkFlow role is mainly concentrated in the early stages of the project, when the designer needs to evaluate and select from various radiator models or explore potential modifications before requesting their production. To streamline this process, automation of the computational tool is mandatory, enabling the designer to rapidly compare different configurations and identify the best solution within the tight deadlines characteristic of the design phase. Once the model is selected, the company can focus on more detailed flow and thermal analysis. For this reason, the tool has been designed to require as little input as possible, only using the data immediately available to the designer also for the automatic generation of the computational mesh.

The first input required from the user is the radiator geometry, provided as (i) the CAD file of the radiator in .stp format; (ii) the .stl files for the solid walls and the inlet and outlet surfaces, as shown in Figure 1. The latter is essential for generating the mesh and applying boundary conditions, while the .stp file is essential for identifying some “critical” points of the model by using FreeCAD.

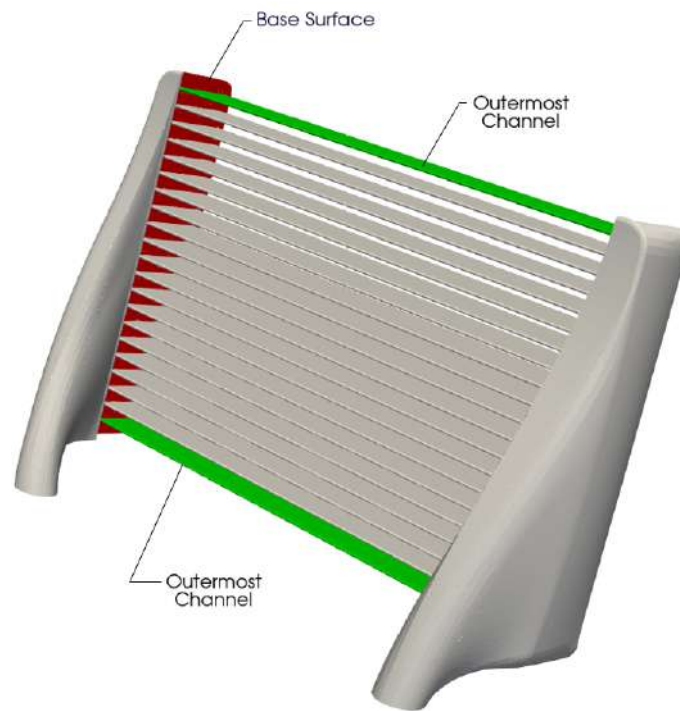
### 3.1. User Input Through FreeCAD and Graphical Interface

To allow T-WorkFlow to handle any configuration of the radiator, geometrical information related to the particular case under analysis is required before generating the mesh. To monitor the flow rate distribution in the radiant ducts during the simulation, cf. Section 3.4, it is necessary to generate a control surface for each duct cross-section, an operation that can be accomplished using the OpenFOAM command *topoSet*. To achieve this, the corresponding OpenFOAM dictionary file must specify a point on each control surface to be generated, along with its normal vector. Therefore, it is at least necessary to know the direction of the axis of the channels and the coordinates of two points corresponding to the outermost channels. It is important to note that in this type of application, the radiator geometry is typically provided according to the car’s reference system, as presented in Figure 4. This prevents making any assumptions about its positioning and orientation in space beforehand. Therefore, this information must be directly extracted from the CAD model of the specific configuration being analyzed. This operation is performed using the open-source FreeCAD [14] software, which is automatically launched when T-WorkFlow is executed and after the generation of the *InputExtraction.FCMacro* file. This macro contains all the Python commands necessary to extract and store the specific inputs required by the workflow.



**Figure 4.** CAD model of a car from Tatuus Racing S.p.a.—Example of the radiator positioning within the vehicle, colored in red. Images courtesy of Tatuus Racing S.p.a.

The *getSelectionEx()* method [21] from the FreeCAD's *Selection* module is here used to extract the various geometric details from the surfaces interactively selected by the user via the software graphical interface. Once the .stp file is loaded, the user only needs to perform the following two operations before running the macro: (i) select a surface from each of the outermost radiant channels; and (ii) one of the two surfaces at the base of all channels, whose normal vector is aligned with the axis of the ducts. An example of this operation is reported in Figure 5.



**Figure 5.** Surfaces to be selected in sequence using the FreeCAD graphical interface are highlighted in green and red.

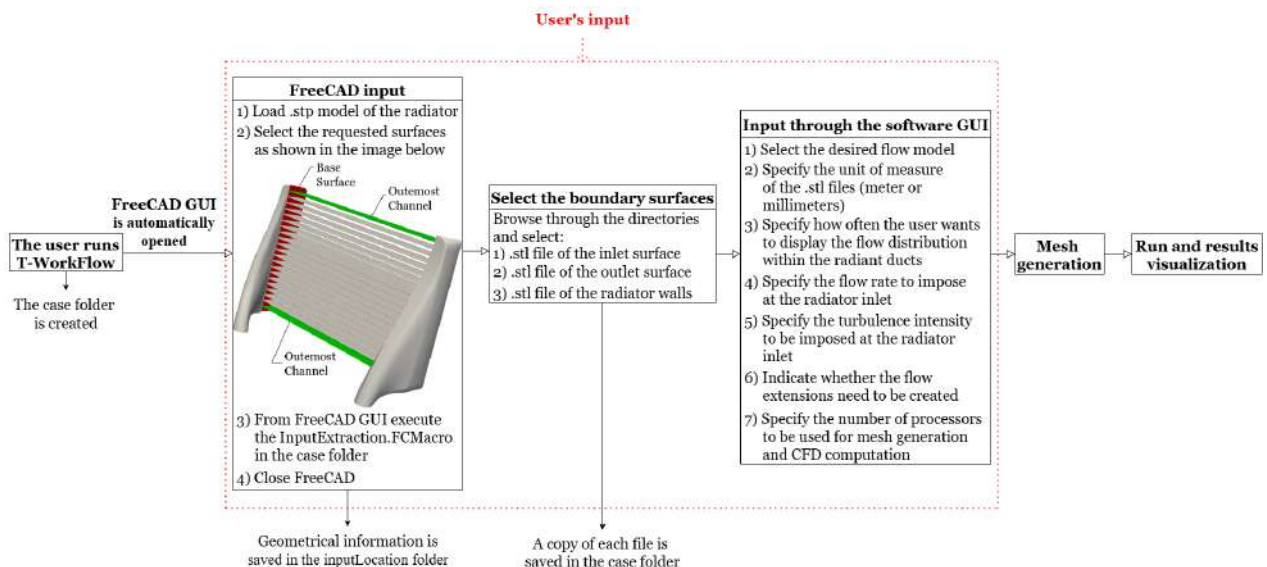
From selection (i), the software computes the barycenter coordinates of the two outermost radiant channels, while from selection (ii), it extracts the normal vector and barycenter coordinates of the corresponding base surface. In addition, T-WorkFlow automatically identifies two opposite vertices on the base surface of the radiant ducts as the pair of points on the surface with the maximum Euclidean distance among all possible combinations of points. These geometric details are also essential for the pre-processing operations related to mesh generation, as is discussed later in Section 3.2. The last instructions within the macro are dedicated to determining the number of channels of the radiator and calculating the cross-sectional area of each radiant duct. These metrics are then used to (i) roughly estimate the expected Reynolds number within the channels, thereby automating the definition of prism layer extension through simple fluid dynamics correlations, cf. Section 3.3.2; (ii) compute the ratio between the channel cross-sectional area represented by the computational grid and the actual cross-sectional area defined in the CAD model to estimate the discretization error; and (iii) define a “detection grid” to be used to establish the control surfaces for monitoring the flow rate within the channels. For this purpose, a cutting plane is automatically defined such that its normal is aligned with the axis of the ducts, and the barycenters of the two outermost channels lie on it. The number of ducts in the radiator is then determined by the number of distinct surfaces on this plane, and the cross-sectional area of each of them is extracted using the *getSelectionEx()* [21] method.

Upon execution of the macro, the relevant geometrical information extracted from the CAD model is saved in the *InputLocation* folder. The user can then exit the FreeCAD interface and provide the additional inputs required by T-WorkFlow through a dedicated

graphical interface, developed with Python's `tkinter` library to facilitate the interaction with the software. The following inputs are requested:

1. select the desired flow model between: (i) laminar; (ii)  $k-\epsilon$  [22]; (iii)  $k-\omega$  [23]; (iv)  $k-\omega$  SST [24];
2. select the `.stl` files for the radiator's inlet, outlet, and walls, needed to set up the fluid domain and define the boundary conditions;
3. specify the measurement unit used for the generation of the `.stl` files;
4. specify how often, in terms of iterations of the CFD solver, the user wants to visualize the flow distribution within the radiant ducts;
5. specify the volumetric flow rate to be imposed at the radiator inlet;
6. specify the turbulence intensity ( $Tu\%$ ) to be imposed at the radiator inlet;
7. indicate whether the inlet and outlet surfaces should be extruded along their normal direction (flow extension) to regularize the flow entering the radiator;
8. specify the number of processors to be used for parallel grid generation and parallel CFD computation (partitions). Note that the number of processes used in the two tasks may be different.

Some of the information obtained through the graphical interface is used to generate the dictionary files in the `system` folder and to create the files in the `0` folder for imposing the desired boundary conditions, as detailed in Section 3.5. Figure 6 visually summarizes the operations performed by T-WorkFlow, with particular attention to the user's input phase. The software requires minimal inputs, streamlining the process while maintaining flexibility to test the models under any desired conditions. T-WorkFlow has been developed for technical office personnel, who should have no difficulty providing the small amount of required information. During the input phase, the most common error users may encounter is the incorrect selection of the surfaces highlighted in Figure 5 via the FreeCAD interface. If this happens, when the `InputExtraction.FCMacro` file is executed, an error message will appear on the FreeCAD GUI. The user will then need to reselect the three required surfaces. The needed geometrical data will only be saved in the `InputLocation` folder once the `InputExtraction.FCMacro` is executed successfully.



**Figure 6.** Summary of T-WorkFlow main operations and user interaction.

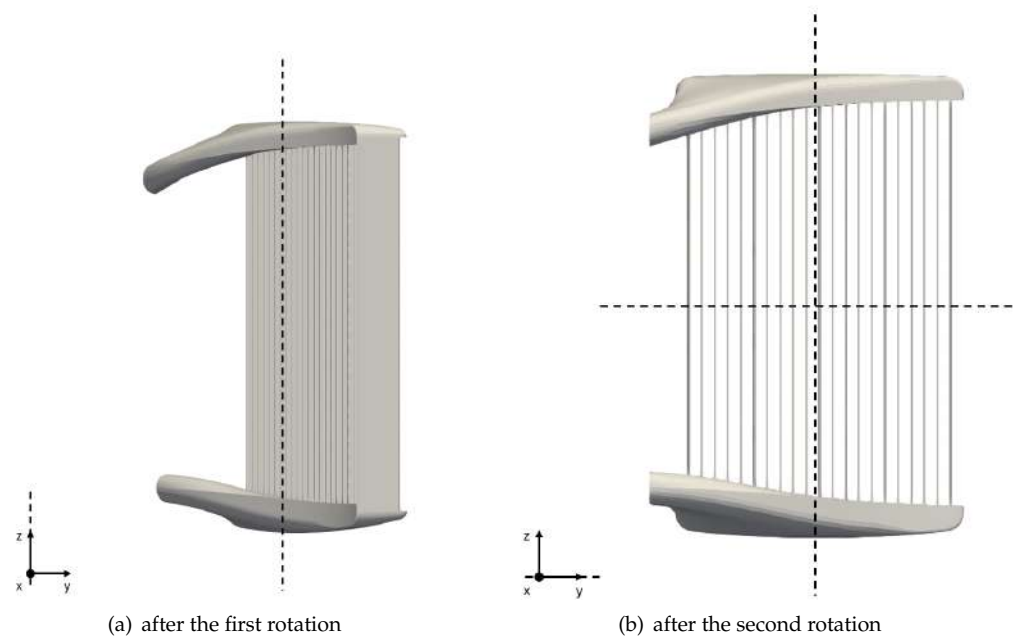
### 3.2. Geometry Processing

T-WorkFlow uses the *blockMesh* and *snappyHexMesh* tools part of the OpenFOAM suite for the generation of the computational mesh, as detailed in Section 3.3. *snappyHexMesh* is a fully parallel mesh generator that takes as input an existing (background) mesh, usually generated by *blockMesh*, and automatically and recursively splits its hexahedra to capture

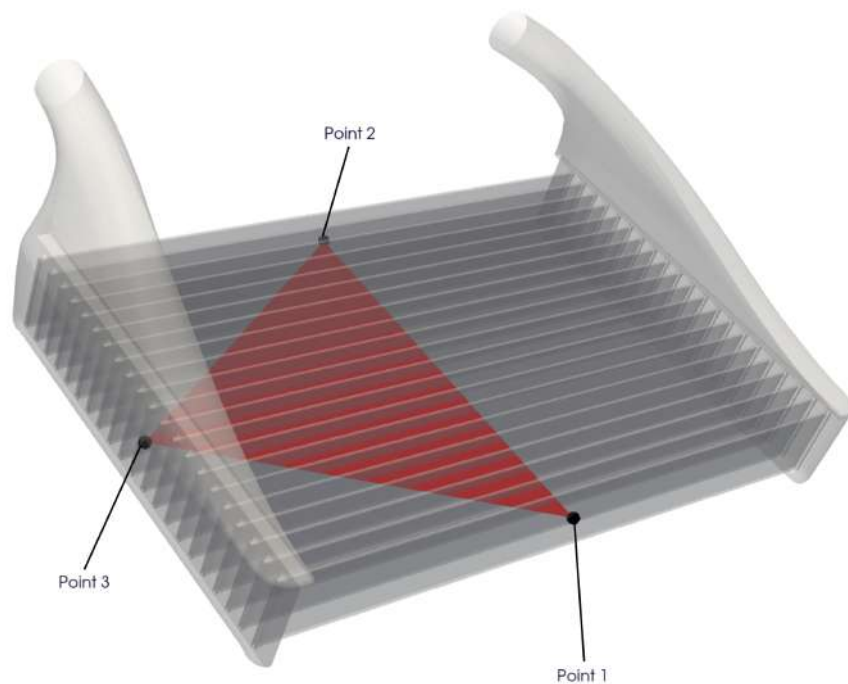


the surface of the CAD model accurately. Before creating the mesh, some pre-processing operations need to be performed on the .stl files provided as input. It is essential to scale these files properly to ensure that their dimensions are expressed in meters, as OpenFOAM assumes this unit of measurement for .stl files. Depending on the CAD export settings, these data may, in fact, be in different units, usually millimeters. To perform this scaling for the inlet, outlet, and wall surfaces, T-WorkFlow uses the OpenFOAM command *surfaceMeshConvert*. Our empirical evidence indicates that the position and orientation of the CAD model in the frame of reference of the initial (background) Cartesian grid created by *blockMesh* have a significant impact on the quality of the mesh produced by *snappyHexMesh*, as well as the time it takes to generate the grid. In fact, the sides of the background grid generated by *blockMesh* are aligned by design with the Cartesian directions. Therefore, the .stl geometry files, which are typically provided in the vehicle reference system, cf. Figure 4, need to be properly rotated to fit the domain. We achieved the best results by applying two consecutive transformations: (i) align the ducts axis with a Cartesian direction and (ii) align the vector connecting the barycenter of the outermost channels with a second Cartesian direction. This process is schematically represented in Figure 7. These transformations are performed using the OpenFOAM *surfaceTransformPoints* command, which automatically defines the axis and angle of rotation based on the directions of a vector before and after the transformation. Hence, to execute transformation (ii), it is necessary to know the coordinates of the points extracted by using the FreeCAD macro, cf. Section 3.1, after the first rotation (i). To this end, before applying any rotation, T-WorkFlow generates the *triangulation\_not\_rotated.stl* file, containing the surface defined by the points extracted by the FreeCAD macro, as represented in Figure 8. Rotation (i) is then applied to both the .stl files related to the radiator surface and to *triangulation\_not\_rotated.stl* using the OpenFOAM command *surfaceTransformPoints*. This leads to the creation of the *triangulation\_1.stl* file containing the coordinates of the points originally extracted using the FreeCAD macro after the first rotation. By reading this file, it is therefore possible to define the vector connecting the centroids of the two outermost channels, which is then used to apply rotation (ii) to both the .stl files related to the radiator surface and to *triangulation\_1.stl*. In this way, in addition to obtaining what is schematically represented in Figure 7, the file *triangulation.stl* is generated, from which the coordinates of points 1, 2, and 3, represented in Figure 8, are obtained as a result of the applied rotations. T-WorkFlow then applies the same transformations to the gravitational acceleration vector through appropriately defined rotation matrices. The impact of these operations on the generation of the computational grid is shown in Figure 9 for a detail of the cross-section of a channel.

The last pre-processing operation performed by T-WorkFlow involves splitting the .stl file of the wall surface, see Figure 1, into two additional files: (i) *channels.stl*, which represents only the surfaces of the radiating channels; (ii) *walls.stl*, which includes the surfaces of the tanks as well as the inlet and outlet channels. This subdivision may already be present in the original .stl file provided as input, depending on how it was exported by the designer. If not, the software will automatically perform this operation. To this end, T-WorkFlow defines a parallelepiped based on the information previously obtained through the FreeCAD's *macro*, ensuring that it encloses only the radiating channel region, thus allowing for subdivision. This operation is crucial for refining the mesh only in areas where it is necessary when using *snappyHexMesh*. A critical region that requires a large mesh density is the tight radius of curvature at the ends of the channels, as shown in Figure 9. By partitioning the wall region in this way, as discussed in Section 3.3.2, refinement can be concentrated on this specific region. This avoids unnecessary refinements elsewhere, limiting the overall number of mesh elements and computational cost.

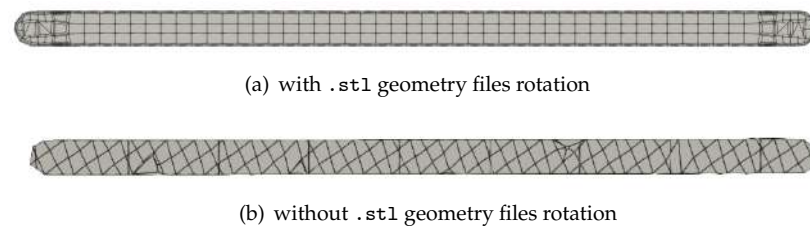


**Figure 7.** T-WorkFlow—Effect of the two rotations on the radiator positioning.



**Figure 8.** Example of a radiator model before the rotations, with the surface `triangulation_not_rotated.stl` highlighted in red.

These pre-processing operations are necessary to obtain a mesh, generated using the OpenFOAM tools *blockMesh* and *snappyHexMesh*, that meets the requirements defined by Tatuus Racing S.p.a., cf. Section 3.3. Automating these operations ensures significant time savings within the overall simulation process while greatly reducing the chance of human errors.



**Figure 9.** Comparison between mesh given by the same *snappyHexMeshDict* in a cross-section of the channels.

### 3.3. Mesh Generation

The most significant challenge encountered during the development of T-WorkFlow was the automatic generation of the computational mesh, which posed two critical constraints. Firstly, to meet the specific requirements of Tatuus S.p.a. concerning available computational resources and to minimize the time required for the overall CFD process, the mesh should not exceed  $8 \times 10^6$  cells. Secondly, to enable meaningful comparisons between CFD results of different radiator models, it is critical to ensure that the automatically generated mesh has a consistent level of quality, regardless of the specific geometry. In this regard, the pre-processing operations described in Section 3.2 were essential to achieve the highest possible mesh quality while respecting these constraints.

In this work, the grid generation process is based on the *blockMesh* and *snappyHexMesh* tools from the OpenFOAM suite, which require properly defined input files (dictionaries) that are automatically generated by T-WorkFlow, as is discussed below.

#### 3.3.1. Automated Configuration of the *blockMeshDict*

An underlying grid is required as input to use *snappyHexMesh*, which will then be refined using an octree-based algorithm. The input mesh is here created by the OpenFOAM utility *blockMesh*, which splits a bounding box of the fluid domain into a Cartesian grid. The parameters for this utility are specified in the *blockMeshDict* file located in the *system* directory. This configuration file defines the coordinates of the bounding box vertices and the number of subdivisions along each edge. To automatically obtain the coordinates of the box, T-WorkFlow uses the *numpy-stl* Python library [16]. This library extracts the minimum and maximum values along each Cartesian axis from the provided *.stl* files. The quality of the final mesh generated by *snappyHexMesh* is significantly influenced by the “splitting strategy” used by *blockMesh*. A basic condition is to try to keep the aspect ratio of the hexahedral cells as close to unity as possible; ideally, the grid should be made of cubes. To achieve this, the length of each edge of the bounding box must be divisible by a single factor, obtaining an integer result. For this purpose, the coordinates of the vertices of the box are adjusted appropriately. In T-WorkFlow, this common divisor is set to 30% of the inlet diameter, a value automatically captured from the *inlet.stl* files using the *numpy-stl* library. Our numerical experiments have shown that this value balances the computational efficiency of *snappyHexMesh* and the quality of the resulting mesh. This methodology helped standardize the mesh generation process and ensure a consistent grid quality across various geometries.

#### 3.3.2. Automated Configuration of the *snappyHexMeshDict*

*snappyHexMesh* is one of the grid generators available in the OpenFOAM suite. Starting from the *.stl* files of the fluid domain and the base grid created using *blockMesh*, it is capable of systematically discretizing even complex geometries. The generation of the *snappyHexMeshDict* is automated by T-WorkFlow, and this file is made by five parts, which are briefly described below: (i) *geometry*; (ii) *castellatedMeshControls*; (iii) *snapControls*; (iv) *addLayersControls*; (v) *meshQualityControls*. In the *geometry* section, the entities that define the fluid domain are specified. This includes identifying the names of the corresponding *.stl* files, which must be located in the *constant/triSurface*

folder. Once these entities have been defined, the progressive refinement phase of the underlying grid defined by *blockMesh* begins. This is carried out according to the information provided in *castellatedMeshControls* and follows an octree-refinement-based algorithm. The refinement levels selected for the different surfaces are fixed and have been chosen to balance the quality of the resulting grid, the time required for its generation, and the total number of cells, according to the specifications of Tatuus Racing S.p.a. In this phase, particular attention is given to certain regions of the radiator that require a larger mesh density to be properly represented. To this end, through the OpenFOAM command *surfaceFeatures*, it is possible to extract from the *.stl* files indicated in the corresponding dictionary some features of the surface where a higher mesh refinement is required. Particularly, this command generates the *.eMesh* files in *constant/triSurface* folder, which can then be referenced in the *feature* section of *castellatedMeshControls*, along with the corresponding refinement level. The maximum level of refinement is imposed on the contour of the radiator channels to properly capture their very pronounced curvature (see Figure 9a). Another region to be refined is the edge of the tanks to accurately discretize the junction between the radiator channels and the tanks themselves, as shown in Figure 10. To achieve this, the region of the radiator channels alone was included in a dedicated *.stl* file during the pre-processing phase, cf. Section 3.2. In this way, through the OpenFOAM command *surfaceFeatures*, it's possible to generate the *channel.eMesh* file to which the highest level of refinement is effectively applied. Once the surface refinement phase is completed, *snappyHexMesh* removes the cells not contained in the fluid domain, i.e., all but those cells included in the internal region of the input *.stl* files. The internal region is identified by means of the coordinates of the point provided through the *locationInMesh* keyword in *castellatedMeshControls*, and the software automates the writing of these coordinates. This operation is based on the coordinates of Point 3, represented in Figure 8, which corresponds to the barycenter of the "Base Surface", shown in Figure 5. These coordinates are known due to the information obtained through the FreeCAD macro, cf. Section 3.1. Specifically, Point 3 of Figure 8 is translated in the direction opposite to the normal of the surface on which it resides by a small fraction of the length of the vector connecting it to one of the barycenter of the outermost channels. Following this approach the coordinates of a point within one of the radiator tanks, which are subsequently assigned to the *locationInMesh* keyword within *castellatedMeshControls*, are obtained. Upon completion of the refinement, the snapping phase begins. This process involves a series of operations aimed at conforming the generated mesh to the reference geometry, thereby enhancing its alignment with the computational domain boundaries defined by the input *.stl* files. During this phase, the vertices of the existing mesh are translated towards the target surface to align the hexahedral cells with the geometric interface. This operation is guided by minimizing the distance between the mesh points and the surface, with particular attention given to maintaining mesh quality by limiting cell distortion. This process is iterative and continues until the mesh quality parameters satisfy those specified in the *meshQualityControls* section. Both the *snapControls* and *meshQualityControls* parameters are rather standard and are taken according to the guidelines provided in the various OpenFOAM tutorials.

The final step in the grid generation process is the generation of a layer of prism at the walls to solve the boundary layer region accurately. For the automatic generation of this layer, *T-WorkFlow*, using the inputs from Section 3.1 and simple fluid dynamics relationships, estimates the boundary layer thickness within the radiator channels and consequently generates an appropriate clustering of the prisms layer by setting the parameters *firstLayerThickness* and *nSurfaceLayers* within the *addLayersControls* section. The thickness of the first layer of cells is given as a dimensional value obtained by means of the simplified assumptions discussed below, thus allowing for the complete automation of this process. To obtain a representative value of the fluid velocity in a radiator channel

and, consequently, the “local” Reynolds number, it is assumed that the volumetric flow rate at the inlet is uniformly distributed across the radiating ducts.

$$Q_{in} = \sum_{i=1}^N Q_i = N Q_i = N U_i A_i \iff U_i = \frac{Q_{in}}{N A_i}; \quad (1)$$

where  $Q_{in}$  is the volumetric flow rate at the inlet,  $N$  is the number of radiator channels,  $U_i$  is the velocity component in the direction of the channel axis, and  $A_i$  is the cross-sectional area of a single duct. The cross-sectional area of the channels and their number have been automatically provided by the FreeCAD macro execution, cf. Section 3.1, while the inlet volumetric flow rate is given as input by the user, cf. Section 3.1. Once the velocity in the single channel is estimated, it is possible to compute the “local” Reynolds number, as follows:

$$Re = \frac{\rho D U_i}{\mu}, \quad (2)$$

where  $D$  is the reference dimension of a single duct here, defined as  $D = \sqrt{A_i}$ ,  $\rho$  is the fluid density, and  $\mu$  is the dynamic viscosity. Thus, following the correlation provided in [25] for the turbulent flat plate, it is possible to estimate

$$C_f = [2 \log_{10}(Re) - 0.65]^{2.3}, \quad (3)$$

$$\tau_w = \frac{1}{2} \rho U^2 C_f, \quad (4)$$

$$u_\tau = \sqrt{\frac{\tau_w}{\rho}}, \quad (5)$$

where  $C_f$  is the skin friction coefficient,  $\tau_w$  is the shear stress in the tangential direction to the wall, and  $u_\tau$  is the friction velocity. The desired height of the first wall prism is usually expressed in terms of wall units,  $y^+ = y u_\tau / \nu$ , depending on the turbulence model and the possible use of wall functions, e.g.,  $y^+ \approx 3$  for  $k - \omega$  following [26]. By imposing the  $y^+$  value it results in

$$y_H = 2 \frac{y^+ \mu}{u_\tau \rho}, \quad (6)$$

where  $y_H$  is the dimensional height of the first prism, while  $\frac{y^+ \mu}{u_\tau \rho}$  is the height of the wall adjacent cell centroid. The boundary layer thickness is then estimated as a function of the Reynolds number, following [27].

$$\delta_{99} = \begin{cases} \frac{4.91D}{\sqrt{Re}} & \text{if } Re < 5 \times 10^5 \\ \frac{0.38D}{Re^{1/5}} & \text{if } Re > 5 \times 10^5 \end{cases}. \quad (7)$$

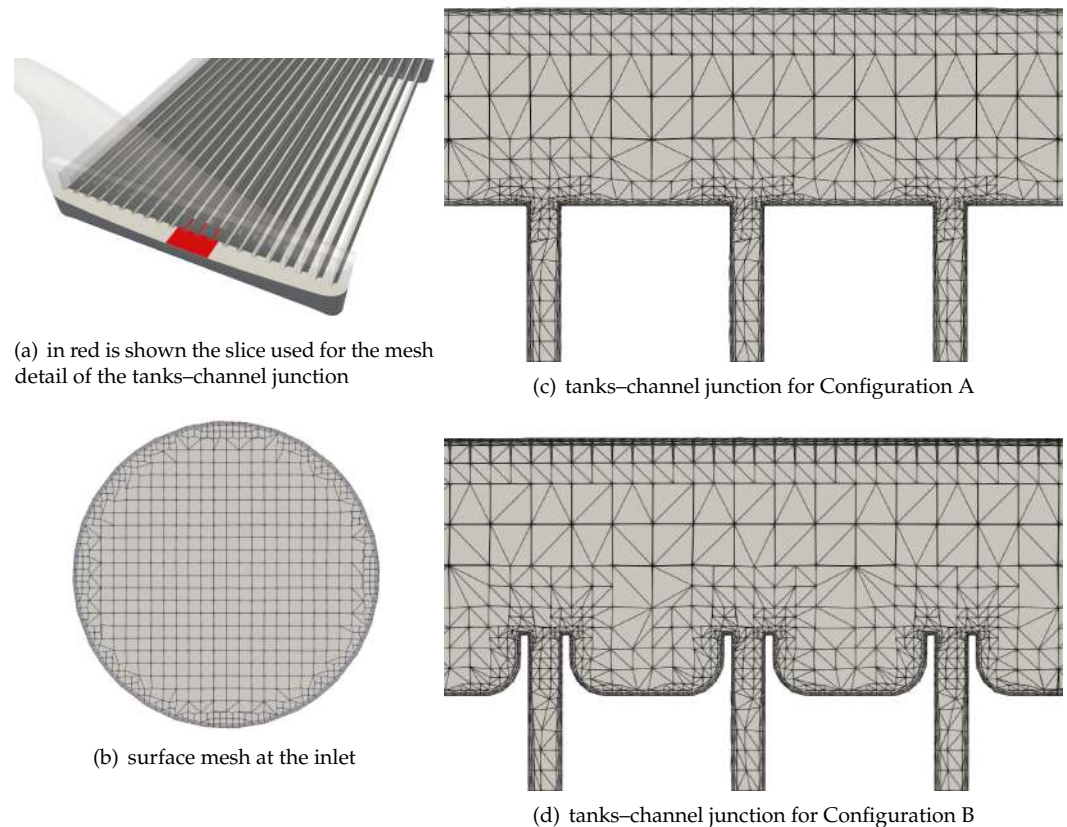
At this point, to properly define the prism layer, it remains to determine their number in order to ensure that their thickness is greater or equal to the boundary layer thickness estimated through Equation (7). To this end, an expansion ratio, which is defined as the ratio between the heights of two consecutive prism layers, is first set. In T-WorkFlow, this parameter is fixed equal to  $r = 1.3$ , which is a suitable value for achieving homogeneous prisms. The total height of the prisms  $y_T$  resulting from the generation of  $M$  layers can then be evaluated through the following geometric series

$$y_T = \sum_{k=0}^{M-1} y_H r^k, \quad (8)$$

where  $k = 0$  indicates the prism layer adjacent to the wall. When  $y_T$  is found to be greater or equal to the estimated boundary layer thickness, the process of introducing prisms is stopped, and the number of layers  $M$  is determined. This value is then assigned to `nSurfaceLayers` in `addLayerControls`.

The final mesh in the radiator channel area, in the region of the junction between the channels and tanks and for the inlet are presented in Figures 9a and 10. In these images, we can appreciate the mesh refinement along the edges of these regions, achieved through the specific generation of the `.eMesh` files using the OpenFOAM `surfaceFeatures` tool.

When performing internal fluid dynamics simulations, it is often advisable to extend the inlet and outlet patches perpendicular to their surface by approximately six times their diameter. This operation ensures a better interaction of the flow with the boundary conditions and allows for the proper development of the incoming boundary layer. Often, the inlet and outlet extrusions are already present in the input `.stl` files. If this is not the case, the user can request for their generation in the graphical interface during the input phase, as shown in Section 3.1, and, once the grid has been generated, T-WorkFlow will extrude the inlet and outlet patches using the OpenFOAM `extrudeMesh` command.



**Figure 10.** Surface mesh at the radiator inlet and cross-sectional detail of the volume mesh of the flow domain near the tanks.

### 3.4. Automatic Detection of the Radiator Channels and Volumetric Flow Rate Monitoring

After generating the computational grid, the radiator ducts are automatically identified to monitor the volumetric flow rate in each of them during the simulation. This operation is performed by using the OpenFOAM command `topoSet`, through which T-WorkFlow defines a control surface for each radiant channel. In the corresponding dictionary, it is necessary to specify the coordinates of a point belonging to each control surface and its normal. The latter is known due to information obtained via the FreeCAD macro, cf. Section 3.1. To identify each channel, the software first creates a vector connecting the barycenter of the two outermost ducts. Subsequently, along this vector, a number of evenly spaced points

equal to 10 times the (known) number of radiant channels are defined. For each of these points, a control surface `faceZoneSet` of OpenFOAM is created using the `topoSet` command. If the point used to define the control surface is actually inside a channel, the corresponding surface file consisting of adjacent mesh faces will be written. However, if the point provided to `faceZoneSet` is in the region between two channels, the nearest enclosed surface to that point will be considered to create the control surface. As this procedure will produce more than a control surface for each radiant duct, the software compares the files related to these surfaces in the `constant/polyMesh/sets` and retains only one copy for each group. This results in a single file for each channel, ensuring that the corresponding control surfaces can be effectively used to monitor the volumetric flow rate in the channels.

After defining the surfaces through which the flow rate of each channel will be evaluated, the method used by OpenFOAM to compute this quantity during the simulation needs to be set. For this purpose, the `surfaceFieldValue` function from the OpenFOAM library `libfieldFunctionObjects` is used. This function calculates the sum of the volume fluxes across the faces of a specified mesh region. In particular, T-WorkFlow automatically appends the following block of instructions at the end of the `controlDict` file for each captured control surface, as well as for the inlet and outlet surfaces

```
flux_pipe_{i + 1}
{
  type          surfaceFieldValue;
  libs          ("/opt/openfoam10
/platforms/linux64GccDPInt32Opt/lib/libfieldFunctionObjects.so");
  log           true;
  writeControl  runTime;
  writeInterval {delta};
  writeFields   true;
  surfaceFormat none;
  regionType    faceZone;
  name          controlZone{faces[i]};
  operation     orientedSum;
  fields        (phi);
}
```

where `flux_pipe_{i + 1}` is the volumetric flow rate through the single duct; `delta` is the interval, measured in solver iterations, between successive outputs of the flow rate, as specified by the user (cf. Section 3.1); and `controlZone{faces[i]}` is the control surface related to that specific channel.

### 3.5. Simulation Set-Up

Once the definition of the control surfaces for monitoring the volumetric flow rate in each radiant channel has been completed, T-WorkFlow executes the CFD solver. In this regard, the `simpleFoam` solver from OpenFOAM based on the SIMPLE algorithm [28] was chosen, as it is suitable for steady-state incompressible flow simulations, also considering the Reynolds-Averaged Navier–Stokes (RANS) simulations closed by different turbulence models. In this regard, in the T-WorkFlow graphical interface, we limited the user selection among the (i)  $k - \epsilon$  [22]; (ii)  $k - \omega$  [23]; (iii)  $k - \omega$  SST [24] models (cf. Section 3.1). The boundary conditions automatically imposed at the different surfaces of the radiator are tabulated in Table 1, where the subscript “WF” stands for “Wall Function”, while the subscript “ifd” stands for “internal field” and  $U_x; U_y; U_z$  are the velocity vector components imposed as uniform at the inlet. These are derived from the volumetric flow rate at the inlet specified by the user (cf. Section 3.1), assuming the velocity vector is parallel to the normal of the inlet. The direction of the normal and the area of the inlet are automatically captured due to the `numpy-stl` library.

**Table 1.** Boundary conditions imposition.

Boundary		$\frac{p}{\rho}$ [m <sup>2</sup> /s <sup>2</sup> ]	$U$ [m/s]	$k$ [m <sup>2</sup> /s <sup>2</sup> ]	$\epsilon$ [m <sup>2</sup> /s <sup>3</sup> ]	$\omega$ [1/s]
Inlet	Type: Value:	zeroGradient /	fixedValue $U_x; U_y; U_z$	fixedValue $k_{ifd}$	fixedValue $\epsilon_{ifd}$	inletOutlet $\omega_{ifd}$
Outlet	Type: Value:	fixedValue uniform 0	zeroGradient /	inletOutlet $k_{ifd}$	inletOutlet $\epsilon_{ifd}$	inletOutlet $\omega_{ifd}$
Walls	Type: Value:	zeroGradient /	noSlip /	kqR <sub>WF</sub> $k_{ifd}$	epsilon <sub>WF</sub> $\epsilon_{ifd}$	omega <sub>WF</sub> $\omega_{ifd}$

The turbulence model variables  $k_{ifd}$ ,  $\epsilon_{ifd}$ , and  $\omega_{ifd}$  are imposed as uniform on the different surfaces and they are evaluated following the indications in [29], as follows:

$$k_{ifd} = 1.5 \|\mathbf{U}_{in}\|^2 Tu^2, \quad \epsilon_{ifd} = C_\mu \frac{k_{ifd}^{1.5}}{l}, \quad \omega_{ifd} = \frac{\epsilon_{ifd}}{C_\mu k_{ifd}}, \quad (9)$$

where  $\|\mathbf{U}_{in}\|$  is the magnitude of the inlet velocity vector,  $Tu$  is the turbulence intensity as imposed by the user (cf. Section 3.1),  $C_\mu = 0.09$  is a model coefficient, and  $l = 0.1D_{in}$ , with  $D_{in}$  being the inlet diameter, is a reference length.

### 3.6. Output Generated

The ultimate goal of T-WorkFlow is to provide the designer with a tool that can streamline and speed up the process of selection of the best configuration among those available. In this regard, the designer needs specific outputs that help evaluate both qualitatively and quantitatively the flow distribution within the radiant channels and the overall pressure drop across the radiator. During the software development, we chose to monitor the evolution of these two parameters throughout the simulation, as they are considered fundamental and representative of the radiator performance, especially in the initial design phase, for which the use of T-WorkFlow is intended. Distributing the fluid uniformly within the radiator channels ensures a more effective heat exchange, while minimizing pressure losses allows for more power to the engine, which can be converted into “thrust”. In a design loop where performance must be maximized, improvements related to these two parameters enable the minimization of the heat exchange surface, allowing for the installation of a smaller and lighter radiator. This, in turn, ensures a less bulky sidepod, leading to reduced aerodynamic drag and, thus, more available power. Additionally, T-WorkFlow produces the text file `Area_ratio`, which is intended to assist the designer in assessing the ability of the computational grid, automatically generated, to accurately reproduce the actual area of the radiant channels. Within this file, the following parameter is given as a percentage:

$$A_{rt} = \frac{\sum_{i=1}^N A_{m,i}}{A_{eff}}, \quad (10)$$

where  $N$  indicates the number of channels,  $A_m$  is the area of the computational grid relative to the control surface defined for the  $i$ -th channel, and  $A_{eff}$  is the area of the cross-section of all the radiant channels obtained from the CAD model via the FreeCAD macro. T-WorkFlow also provides as output the following set of files to monitor during the simulation of the evolution of pressure losses and fluid distribution in the radiator channels:

1. `deltaP_inlet_outlet`;
2. `flowRate_Standard_Deviation`;
3. `charts` update within `flowRateVisualization`.

These are updated during the simulation whenever the solver reaches an iteration count that is a multiple of a user-defined value specified through the graphical interface

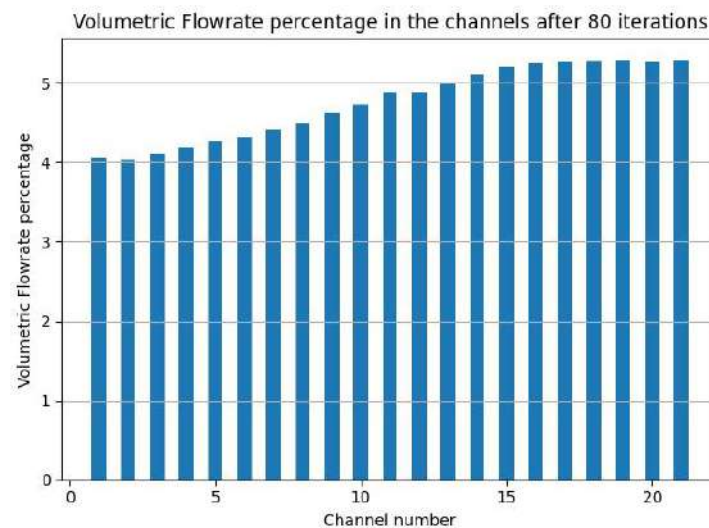


(see Section 3.1). The first file contains the pressure drop  $\Delta\hat{p}_{in-out} = 2(\bar{p}_{in} - \bar{p}_{out}) / \|\mathbf{U}_{in}\|^2$ , evaluated across the inlet and outlet surface and normalized by a dynamic pressure computed with the magnitude of the velocity at the radiator inlet.  $\bar{p}_{in}$  and  $\bar{p}_{out}$  are, here, the integral averages of the pressures evaluated on the inlet and outlet surfaces. Notice that the incompressible solver *simpleFoam* solves for the  $\bar{p} = p/\rho$  variable, where  $\rho$  is the fluid density. To readily have access to the  $\Delta\hat{p}_{in-out}$  parameter is essential for the designer, as it facilitates the quick identification of the radiator model that ensures low pressure losses. The `FlowRate_Standard_Deviation` file contains the value  $Q_{std}$ , which represents the dimensionless standard deviation of the flow rate over the ducts

$$Q_{std} = \sqrt{\frac{\sum_{k=1}^N \left(Q_k - \frac{Q_{in}}{N}\right)^2}{N}} \frac{1}{Q_{in}}, \quad (11)$$

where  $Q_k$  is the volumetric flow rate in the  $k$ -th channel,  $Q_{in}$  is the volumetric flow rate at the inlet, and  $N$  is the number of radiant channels. The mean value for the computation of the standard deviation in Equation (11) has been set to  $Q_{in}/N$ , i.e., an “ideal case” where the flow is evenly distributed among all the channels. `FlowRate_Standard_Deviation` therefore indicates, quantitatively and synthetically, how much the distribution of the volumetric flow rate over the different channels deviates from the “ideal” case.

Finally, `T-WorkFlow` populates the `FlowRateVisualization` folder with a series of graphical outputs in `.png` format, generated at runtime. These images provide the designer with a simple visual representation of the actual flow distribution over the channels; see Figure 11. These plots are created by using the well-known Python library `matplotlib` [17] and accessing at runtime the volumetric flow rate values of each control surface. In particular, Figure 11 shows the dimensionless volumetric flow rate  $\hat{Q}_k = Q_k/Q_{in}$  as a percentage relative to the total inlet flow rate to the radiator.



**Figure 11.** Example of a plot generated at run-time by the software containing the volumetric flow rate distribution as percentage values in the different channels.

It is worth noting that these operations, while they might appear trivial, allow for the full automation of the main post-processing tasks necessary for the designer to identify the best configuration among those tested. `T-WorkFlow` was developed to automatically manage all interactions with `OpenFOAM`, dramatically simplifying the set-up of the case and specific numerical configurations. This enables even non-expert users of this open-source CFD tool to efficiently produce accurate results and make informed decisions within the tight deadlines typical of this type of car design.

#### 4. Numerical Results

In this paragraph, examples of the results provided by T-WorkFlow for the analysis of the flow within a radiator are reported. For this purpose, the two radiator configurations shown in Figure 2 were given as input to the tool. All the results shown in this section are obtained using the  $k - \omega$  [23] as turbulence model. The simulation ends when the steady-state condition, identified by the values of some norm of the residuals of the spatial discretization, is reached. Notice that the dimensional value of the inlet flow rate is not provided for confidentiality reasons. However, the exact value is not essential for the purposes of this paper, as our focus is on demonstrating the capabilities of T-WorkFlow in handling the entire CFD workflow through comparative simulations of two configurations. For this reason, the analysis emphasizes the relative performance of the configurations rather than absolute values. In particular, as mentioned in Section 1, the goal of the present comparison is to assess if Configuration A, which represents a possible geometrical simplification of the “real” Configuration B, can be effectively used to provide computational savings without compromising the validity of the numerical results. Ideally, these two configurations should have the same performance. The fluid considered for all the simulations is water at ambient conditions, as Formula 3 [3] and Formula 4 [2] regulations prescribe using “simple” fluids to keep costs down and not dangerous to humans and the environment since they can be dispersed in the event of an accident.

Regarding the automated mesh generation with the *snappyHexMesh* utility, the main results from the OpenFOAM *checkMesh* command for the mesh diagnostic are provided in Table 2 for the analyzed configurations.

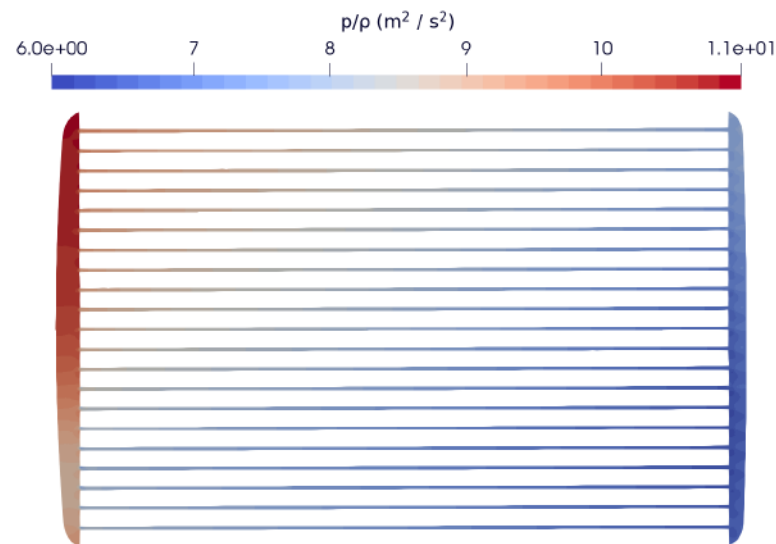
**Table 2.** Main information from the OpenFOAM *checkMesh* diagnostic tool for meshes for the two configurations.

Parameter	Configuration A	Configuration B
Number of cells	6,666,090	7,655,500
Max cell openness	$8.09 \times 10^{-16}$	$7.12 \times 10^{-16}$
Max aspect ratio	31.14	25.26
Minimum face area [m <sup>2</sup> ]	$1.02 \times 10^{-9}$	$1.25 \times 10^{-9}$
Maximum face area [m <sup>2</sup> ]	$1.67 \times 10^{-5}$	$1.65 \times 10^{-5}$
Min volume [m <sup>3</sup> ]	$5.26 \times 10^{-13}$	$1.22 \times 10^{-13}$
Max volume [m <sup>3</sup> ]	$6.68 \times 10^{-8}$	$6.58 \times 10^{-8}$
Mesh non-orthogonality		
Max [°]	69.96	69.70
Mesh non-orthogonality		
Average [°]	10.84	11.08
Max skewness	3.94	3.90

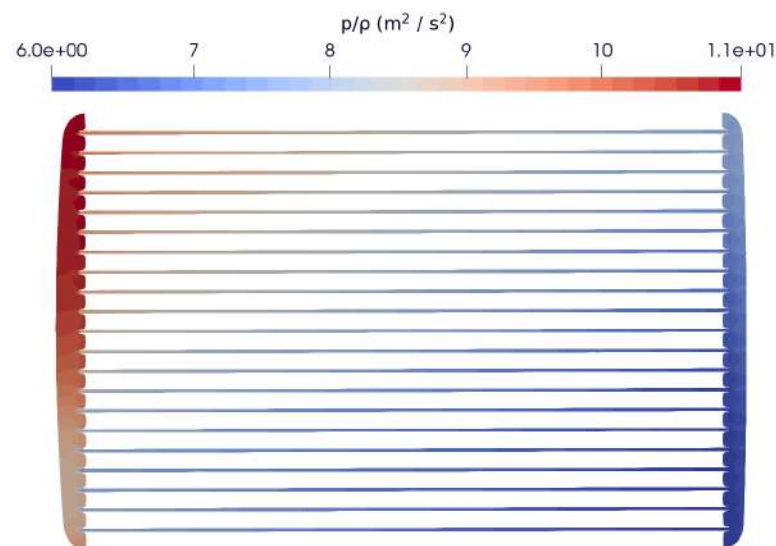
As clearly evidenced by Table 2, mesh quality is comparable between the two configurations. As expected, the mesh of Configuration A is characterized by fewer cells than those of Configuration B, i.e., about 13% fewer elements. This is the key factor suggesting that using the geometry of Configuration A instead of Configuration B could result in significant computational savings. This difference in the number of cells is due to the different ways in which the channels are connected to the water tanks in the two configurations, see Figure 2. The more complex junction characterizing Configuration B leads to the extraction of a greater number of features through the *surfaceFeatures* command, resulting in a higher level of refinement on a greater number of cells than in Configuration A, as clearly evidenced in Figure 10. Moreover, due to the *postProcess* option of *simpleFoam*, it is possible to obtain the average value at of  $y^+$  related to the generated mesh. For the converged solution of Configuration A, this parameter is found to be  $y_A^+ = 0.57$ , while for Configuration B,  $y_B^+ = 0.53$ . These results support the validity of the methodology used for the automatic generation of prism layers with *snappyHexMesh*, cf. Section 3.3.2. Another parameter related to mesh quality is  $A_{rt}$ , computed according to Equation (10). For both the generated meshes, this

parameter turns out to be  $A_{rt} \approx 99.2\%$ , indicating an excellent ability of the automatically generated computational grid to accurately copy the CAD geometry of the radiating ducts.

To effectively determine whether Configuration A can be used in place of Configuration B to conduct numerical investigations of radiator operation, we now present the leading performance indicators, i.e., pressure losses and flow distribution within the radiator channels, obtained using T-WorkFlow for both geometries. Figure 12, generated through the open-source ParaView software [30,31], shows the pressure distribution on the plane longitudinally intersecting the channels and passing through the barycenter of a tank.



(a) Configuration A



(b) Configuration B

**Figure 12.** Pressure distribution on a plane parallel to the Cartesian  $yz$ -plane, passing through the barycenter of a tank.

The pressure contours in Figure 12 suggest that pressure losses across the radiators are larger for Configuration B. The  $\Delta\hat{p}_{in-out}$  values, defined in Section 3.6, confirm this observation, but the difference between  $\Delta\hat{p}_{in-out,A} = 1.686$  and  $\Delta\hat{p}_{in-out,B} = 1.709$  is actually minimal, i.e.,  $\approx 1\%$ . Regarding the flow rate distribution within the radiator channels, the plot provided by T-WorkFlow for the converged solution for both radiator configurations is shown in Figure 13. The analysis of this figure reveals a sim-

ilar flow rate distribution between the two configurations. This is confirmed by the FlowRate\_Standard\_Deviation value, automatically evaluated by T-WorkFlow according to Equation (11), i.e.,  $Q_{std,A} = 4.0 \times 10^{-3}$  and  $Q_{std,B} = 4.2 \times 10^{-3}$ .

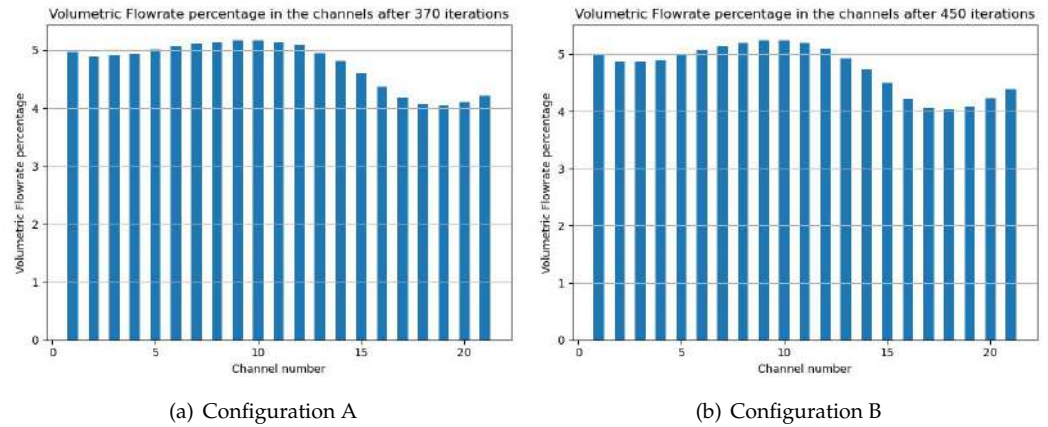


Figure 13. Flow rate distribution across the radiant ducts.

The trend observed in Figure 13 is further corroborated by the distribution of the velocity component associated with mass transport, as shown in Figure 14. This distribution is shown on a plane normal to the axial direction of the ducts, passing through their barycenters. From the analysis of Figure 14, it is clear that the two configurations show an almost identical behavior.

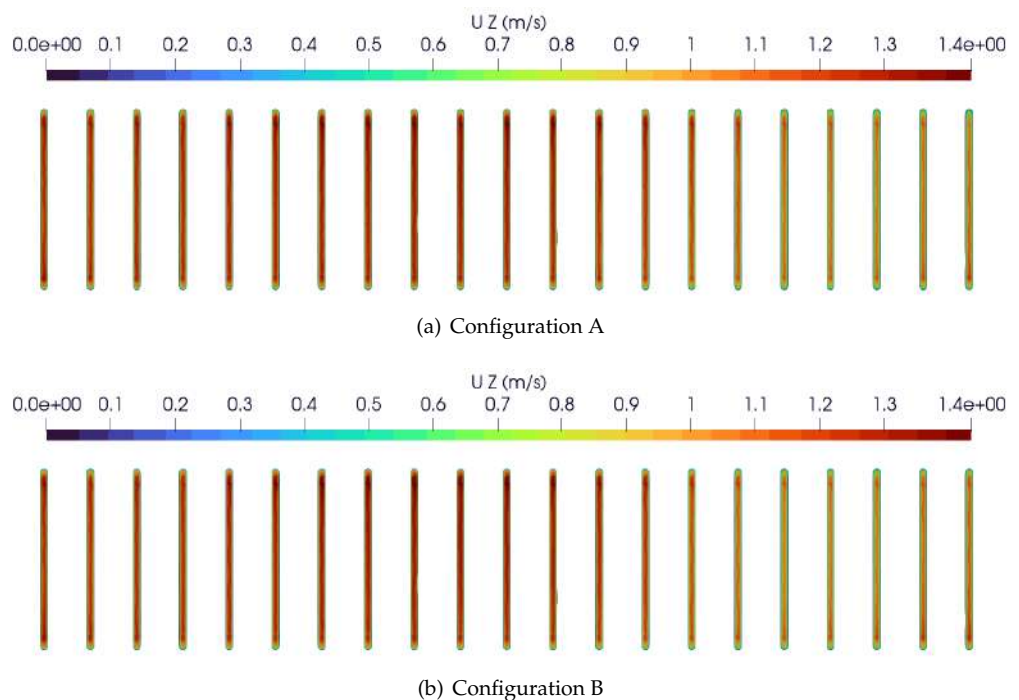
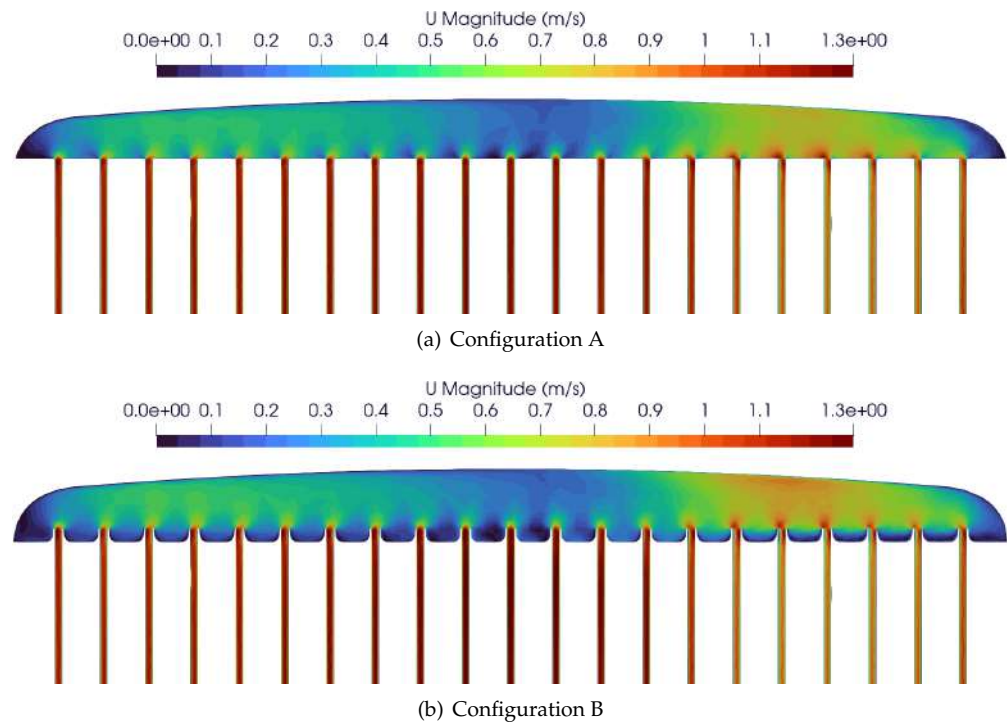


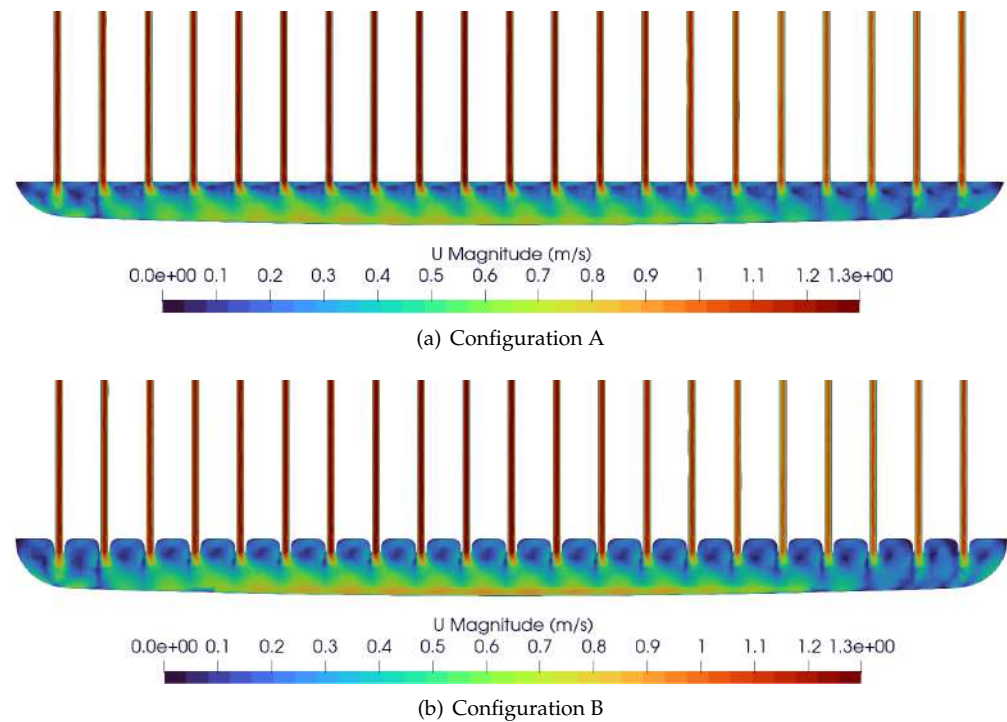
Figure 14. Distribution of the velocity component related to mass transport in the channels on a plane with normal parallel to their axis direction and that passes through their barycenters.

Details of the numerical solution in the region of the inlet and outlet tanks are presented hereafter to highlight the different flow behavior where the differences between the geometries occur, see Figure 2. In Figures 15 and 16, the distribution of the velocity vector magnitude is presented, composed solely of the velocity components lying on the yz-plane for both the tanks. As expected, by comparing Figure 15 with Figure 16, the main

differences between the two configurations lie in the pronounced recirculation regions at the radiant duct connection with the outlet tank of Configuration B, rather than in the inlet region.



**Figure 15.** Distribution in the inlet tank of the velocity vector module, composed solely of the velocity components lying on the plane parallel to the Cartesian  $yz$ -plane and that passes through the barycenter of this tank.



**Figure 16.** Distribution in the outlet tank of the velocity vector module, composed solely of the velocity components lying on the plane parallel to the Cartesian  $yz$ -plane and that passes through the barycenter of this tank.

Streamlines colored with the magnitude of the velocity vector are displayed in Figures 17 and 18 for both configurations, focusing on the region of the tanks. In the outlet reservoir of Configuration B, the flow recirculation at the connection between the radiating ducts and the tank is clearly evident. This distinct flow behavior with respect to Configuration A results from the geometric differences between the two models. However, based on the performance indices, it does not lead to significant differences in the overall hydraulic performance of the two configurations. For this reason, Configuration A can be effectively used in production runs to guarantee computational savings without significantly compromising the accuracy of the numerical results. In fact, it is worth mentioning that the numerical simulation performed for Configuration A converged in  $\approx 370$  iterations, while that of Configuration B reached a steady-state condition after  $\approx 450$  iterations. This 18% reduction in the number of iterations demonstrates the computational savings that can be obtained by using the simplified geometry.

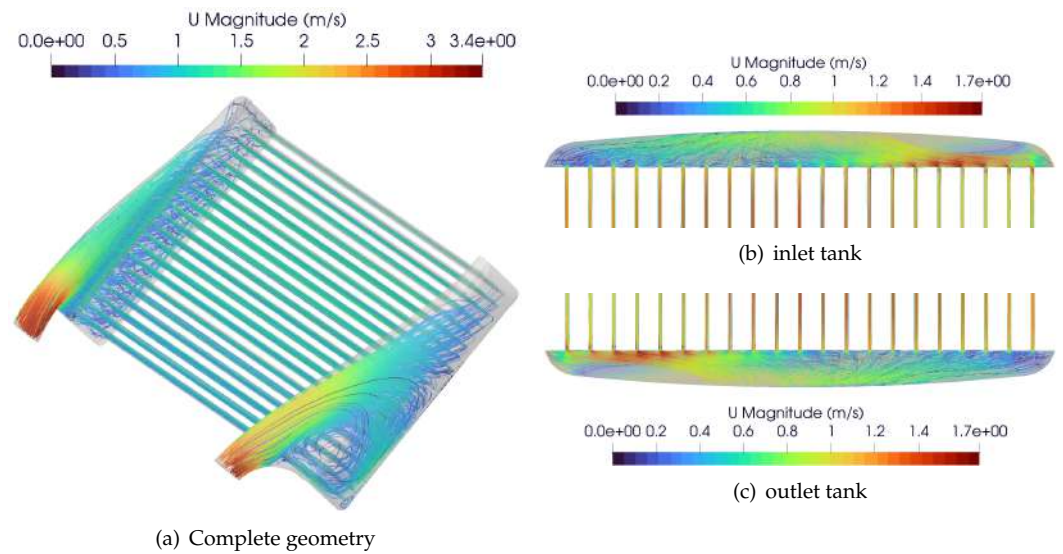


Figure 17. Streamlines colored with the velocity vector magnitude for Configuration A.

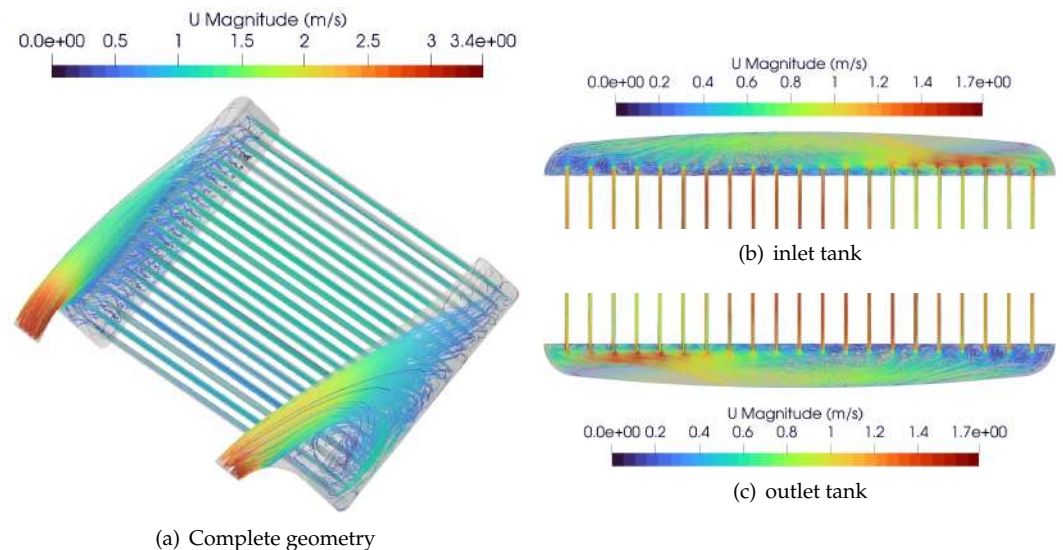


Figure 18. Streamlines colored with the velocity vector magnitude for Configuration B.

### 5. Conclusions

This paper presents a software tool designed to fully automate and manage the workflow for CFD simulations of radiators mounted in race cars. Software tools like the one

proposed here could provide significant value to automotive companies. In fact, companies can lower costs by using open-source software and reduce the time needed for designers to obtain reliable and standardized CFD results through the customization and streamlining of the process. In this specific case, the tool has been designed in the most general way possible so that any radiator geometry the company wants to analyze can be used as input. Great care was taken to develop a procedure ensuring the generation of computational meshes of comparable quality regardless of the specific radiator configuration. This allows for consistent comparison between numerical results from different geometries, thus enabling even non-expert users to make informed decisions within the tight deadlines typical of the race car design. For this purpose, during the development, it was necessary to compare various pre-processing operations to enhance the quality of the computational grid while reducing its generation time using *snappyHexMesh*. T-WorkFlow fully manages the interaction with OpenFOAM, automates post-processing, and provides real-time outputs to quickly identify the most effective configuration in evenly distributing the flow across the radiating ducts. The paper also provides a practical example by applying the software to two different radiator configurations and includes a comparative analysis of their hydraulic performance. In particular, it was successfully demonstrated that a possible geometrical simplification in the tank–channel junction can lead to important computational savings without significantly affecting the accuracy of the numerical results.

In future work, several improvements could be implemented in T-WorkFlow, some simple and immediate and others more complex, to increase the impact of the tool on the design practice of the company. The potential new features include but are not limited to (i) allowing users to select different levels of mesh refinement, with the tool automatically adjusting the *snappyHexMeshDict*; (ii) automatically scheduling multiple simulations for a parametric study involving various geometries or operating conditions; (iii) applying the existing workflow to simulate other components, such as intercoolers. More ambitious goals involve integrating the T-WorkFlow into a complete shape optimization process by utilizing open-source tools like Dakota [32], as shown in [33]. Lastly, a particularly intriguing possibility lies in integrating machine learning methods to support the numerical simulation practice for ground vehicles, as explored, for example, in [34].

**Author Contributions:** Conceptualization, F.M., M.V., E.B., A.B. and A.C.; Methodology, F.M., M.V., E.B., A.B. and A.C.; Software, F.M., M.V., E.B., A.B. and A.C.; Validation, F.M., M.V., E.B., A.B. and A.C.; Writing—original draft, F.M., M.V., E.B., A.B. and A.C.; Writing—review & editing, F.M., M.V., E.B., A.B. and A.C.; Supervision, A.C. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding authors.

**Conflicts of Interest:** Authors Eugenio Bardoscia and Andrea Bortoli were employed by the company Tatuus Racing S.p.a. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

## References

1. Tatuus S.p.a. Tatuus Official Website. Available online: <https://tatuus.it/> (accessed on 27 August 2024).
2. Federation Internationale de l'Automobile. FIA F4 Championship. Available online: <https://www.f4championship.com> (accessed on 27 August 2024).
3. Federation Internationale de l'Automobile. FIA Formula 3 Championship. Available online: <https://www.fiaformula3.com> (accessed on 27 August 2024).

4. Bottau, F. Design and Optimization of a Race Car Cooling System. Master's Thesis, Aerospace Engineering/Ingegneria Aerospaziale, Università di Bologna, Bologna, Italy, 2018. Available online: <https://amslaurea.unibo.it/id/eprint/16520> (accessed on 27 August 2024).
5. Weller, H.G.; Tabor, G.; Jasak, H.; Fureby, C. A tensorial approach to computational continuum mechanics using object-oriented techniques. *Comput. Phys.* **1998**, *12*, 620–631. [[CrossRef](#)]
6. Guerrero, A.; Castilla, R.; Eid, G. A Numerical Aerodynamic Analysis on the Effect of Rear Underbody Diffusers on Road Cars. *Appl. Sci.* **2022**, *12*, 3763. [[CrossRef](#)]
7. Aguerre, H.J.; Gimenez, J.M.; Escribano, F.; Nigro, N.M. Aerodynamic study of a moving Ahmed body by numerical simulation. *J. Wind Eng. Ind. Aerodyn.* **2024**, *245*, 105635. [[CrossRef](#)]
8. Navó, A.; Bergada, J.M. Aerodynamic Study of the NASA's X-43A Hypersonic Aircraft. *Appl. Sci.* **2020**, *10*, 8211. [[CrossRef](#)]
9. Luo, W.; Yang, Z.; Jiao, K.; Zhang, Y.; Du, Q. Novel structural designs of fin-tube heat exchanger for PEMFC systems based on wavy-louvered fin and vortex generator by a 3D model in OpenFOAM. *Int. J. Hydrogen Energy* **2022**, *47*, 1820–1832. [[CrossRef](#)]
10. Singh, R.J.; Gohil, T.B. The numerical analysis on the development of Lorentz force and its directional effect on the suppression of buoyancy-driven flow and heat transfer using OpenFOAM. *Comput. Fluids* **2019**, *179*, 476–489. [[CrossRef](#)]
11. Yan, K.; Wang, Y.; Yan, J. Topology Optimization of Two Fluid Heat Transfer Problems for Heat Exchanger Design. *CMES—Comput. Model. Eng. Sci.* **2024**, *140*, 1949–1974. [[CrossRef](#)]
12. Spasov, G.H.; Rossi, R.; Vanossi, A.; Cottini, C.; Benassi, A. A Critical Analysis of the CFD-DEM Simulation of Pharmaceutical Aerosols Deposition in Upper Intra-Thoracic Airways: Considerations on Aerosol Transport and Deposition. *Pharmaceutics* **2024**, *16*, 1119. [[CrossRef](#)] [[PubMed](#)]
13. Van Rossum, G.; Drake, F.L. *Python 3 Reference Manual*; CreateSpace: Scotts Valley, CA, USA, 2009.
14. FreeCAD. Version 0.19.2. 2023. Available online: <https://www.freecadweb.org> (accessed on 23 August 2024).
15. Van Rossum, G. *The Python Library Reference, Release 3.8.2*; Python Software Foundation: Wilmington, DE, USA, 2020.
16. Wojciechowska, W. *numpy-stl*. 2023. Available online: <https://github.com/WoLpH/numpy-stl> (accessed on 23 August 2024).
17. Hunter, J.D. Matplotlib: A 2D graphics environment. *Comput. Sci. Eng.* **2007**, *9*, 90–95. [[CrossRef](#)]
18. Siemens Digital Industries Software. Simcenter STAR-CCM+. Available online: <https://www.plm.automation.siemens.com/global/en/products/simcenter/STAR-CCM.html> (accessed on 24 August 2024).
19. Ansys Inc. Ansys Fluent. Available online: <https://www.ansys.com/products/fluids/ansys-fluent> (accessed on 25 August 2024).
20. Geuzaine, C.; Remacle, J.F. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *Int. J. Numer. Methods Eng.* **2009**, *79*, 1309–1331. [[CrossRef](#)]
21. FreeCAD Community. *getSelectionEx() Method*; FreeCAD, 2023; Method from Version 0.19.2 of FreeCAD. Available online: [https://wiki.freecad.org/Selection\\_API](https://wiki.freecad.org/Selection_API) (accessed on 23 August 2024).
22. Launder, B.E.; Spalding, D.B. The Numerical Computation of Turbulent Flows. *Comput. Methods Appl. Mech. Eng.* **1974**, *3*, 269–289. [[CrossRef](#)]
23. Wilcox, D.C. *Turbulence Modeling for CFD*; DCW Industries: La Canada, CA, USA, 1998; Volume 2, pp. 103–217.
24. Menter, F.R.; Kuntz, M.; Langtry, R. Ten Years of Industrial Experience with the SST Turbulence Model. In *Proceedings of the Turbulence, Heat and Mass Transfer 4, Antalya, Turkey, 12–17 October 2003*; Hanjalić, K., Nagano, Y., Tummers, M., Eds.; Begell House, Inc.: Danbury, CT, USA, 2003; pp. 625–632.
25. Schlichting, H.; Gersten, K. *Boundary-Layer Theory*; McGraw-Hill Book Company: New York, NY, USA, 1979.
26. Menter, F.R. Two-equation eddy-viscosity turbulence models for engineering applications. *AIAA J.* **1994**, *32*, 1598–1605. [[CrossRef](#)]
27. Cengel, Y.; Cimbala, J. *Fluid Mechanics: Fundamentals and Applications*; McGraw-Hill Education: New York, NY, USA, 2006.
28. Patankar, S.V.; Spalding, D.B. Numerical calculation of fluid flow. *Comput. Fluids* **1972**, *1*, 1–26.
29. NASA Langley Research Center. Turbulence Modeling Resource. 2024. Available online: <https://turbmodels.larc.nasa.gov/> (accessed on 24 August 2024).
30. Ahrens, J.; Geveci, B.; Law, C. ParaView: An End-User Tool for Large Data Visualization. In *Visualization Handbook*; Elsevier: Amsterdam, The Netherlands, 2005; ISBN 978-0123875822.
31. Kitware, Inc. ParaView. Available online: <https://www.paraview.org> (accessed on 27 August 2024).
32. Adams, B.; Bohnhoff, W.; Dalbey, K.; Ebeida, M.; Eddy, J.; Eldred, M.; Hooper, R.; Hough, P.; Hu, K.; Jakeman, J.; et al. *A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.15 User's Manual*; Sandia Technical Report SAND2020-12495; Sandia National Laboratories: Albuquerque, NM, Mexico, 2021.
33. De Donno, R.; Ghidoni, A.; Noventa, G.; Rebay, S. Shape optimization of the ERCOFTAC centrifugal pump impeller using open-source software. *Optim. Eng.* **2019**, *20*, 929–953. [[CrossRef](#)]
34. Eiximeno, B.; Miró, A.; Rodríguez, I.; Lehmkühl, O. Toward the Usage of Deep Learning Surrogate Models in Ground Vehicle Aerodynamics. *Mathematics* **2024**, *12*, 998. [[CrossRef](#)]

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.