



**Politecnico
di Torino**

Corso di laurea magistrale
in Ingegneria Energetica e Nucleare

Hydrogen Systems and Enabling Technologies

TESI DI LAUREA MAGISTRALE

A Study of Hydrogen Dispersion

Candidata:

Núria Tirado Creixell

Relatore:

Prof. Andrea Carpignano

Correlatori:

Prof. Federico Ustolin

Ingar Fossan

Emanuela Bruno

Raffaella Gerboni

Luglio 2026

Abstract

Fuelled by the energy transition, hydrogen is gaining popularity as a clean fuel. With hydrogen refuelling stations (HRSs) becoming more common as a result, several potential hazards arise due to the safety concerns related to hydrogen. It is therefore of the utmost importance to understand the consequences of a potential hydrogen leak in a HRS. Because the compressor tends to have a higher failure rate, this project focuses on hydrogen leaks originating there. This study investigates the consequences of both ignited and unignited leaks, and seeks to evaluate how the consequences differ depending on the type of ventilation —natural or forced— used in the compressor enclosure, the driving pressure, the hole size, and the direction of the leak. The effect of firewalls is also evaluated. The study uses computational fluid dynamics (CFD) and analyses a refuelling station envisioned in Voll, Trondheim, which features a three-stage compressor. The leak positions are tied to each of the stages —430, 630 and 900bar—. The hole sizes are defined according to the piping needed in each of the sections, and applying the fractional area approach: 1% area, 10% area and full bore are the considered scenarios. The directions are chosen based on a preliminary sensitivity study of the effects of leak direction. The results show that having forced ventilation is usually helpful when the leak is small, but counter-productive when it is large. Another finding is that higher pressure scenarios do not necessarily imply that the consequences will be higher. In such cases, even though the jet velocity is higher, the jet is always impinged. Moreover, a lower pressure also implies a higher flow rate, the consequences of which are usually worse. The full bore scenarios are found to have potential for the complete displacement of air inside the enclosure, which could lead to an overpressure caused by the pressure peaking phenomenon. The dynamic pressure outside of the protected area is low. The radiation caused by the fire has enormous consequences in the full bore scenarios, but is otherwise effectively contained by the firewalls in most cases. The height of the firewalls is found to be vital in protecting the nearby area, especially by containing the radiation from the fire. At a certain height its benefits in this regard plateau, and this needs to be balanced with the fact that a higher barrier also hinders the efficient dispersion of the cloud. In all, the findings in this study support engineers and regulators in ventilation and firewall design of future HRS.

Abstract (italiano)

Alimentato dalla transizione energetica, l'idrogeno sta acquisendo popolarità come combustibile pulito. Con la conseguente crescente diffusione delle stazioni di rifornimento di idrogeno (HRS), sorgono diversi potenziali pericoli dovuti alle problematiche di sicurezza legate all'idrogeno. È quindi di fondamentale importanza comprendere le conseguenze di una potenziale perdita di idrogeno in una HRS. Poiché il compressore tende a presentare un tasso di guasto più elevato, questo progetto si concentra sulle perdite di idrogeno originate in esso. Questo studio analizza le conseguenze sia delle perdite innescate sia di quelle non innescate e mira a valutare come tali conseguenze differiscano in funzione del tipo di ventilazione — naturale o forzata — utilizzata nell'involucro del compressore, della pressione di esercizio, della dimensione del foro e della direzione della perdita. Viene inoltre valutato l'effetto delle pareti tagliafiamma. Lo studio utilizza la fluidodinamica computazionale (CFD) e analizza una stazione di rifornimento prevista a Voll, Trondheim, dotata di un compressore a tre stadi. Le posizioni delle perdite sono associate a ciascuno degli stadi —430, 630 e 900bar—. Le dimensioni dei fori sono definite in base alle tubazioni necessarie in ciascuna delle sezioni, applicando l'approccio dell'area frazionaria: gli scenari considerati sono 1% dell'area, 10% dell'area e *full bore*. Le direzioni sono scelte sulla base di uno studio preliminare di sensibilità sugli effetti della direzione della perdita. I risultati mostrano che la ventilazione forzata è generalmente utile quando la perdita è piccola, ma controproducente quando è grande. Un altro risultato evidenzia che scenari a pressione più elevata non implicano necessariamente conseguenze maggiori. In tali casi, sebbene la velocità del getto sia più elevata, il getto impatta sempre su una superficie. Inoltre, una pressione inferiore implica anche una portata maggiore, le cui conseguenze sono generalmente peggiori. Si osserva che gli scenari di *full bore* hanno il potenziale di causare il completo spostamento dell'aria all'interno dell'involucro, il che potrebbe portare a una sovrappressione dovuta al *pressure peaking phenomenon*. La pressione dinamica all'esterno dell'area protetta è bassa. La radiazione causata dall'incendio ha conseguenze molto rilevanti negli scenari di *full bore*, ma negli altri casi è nella maggior parte delle situazioni efficacemente contenuta dalle pareti tagliafiamma. Si osserva inoltre che l'altezza delle pareti tagliafiamma è fondamentale per proteggere l'area circostante, in particolare contenendo la radiazione dell'incendio. Oltre una certa altezza, tuttavia, i benefici in questo senso raggiungono un plateau, e questo deve essere bilanciato con il fatto che una barriera più alta ostacola anche l'efficiente dispersione della nube. Nel complesso, i risultati di questo studio forniscono supporto a ingegneri e decisori nella progettazione dei sistemi di ventilazione e delle pareti tagliafiamma per le future HRS.

Contents

Introduction	9
1 State of the art	10
1.1 Evaluating the frequency of hydrogen incidents	10
1.2 Hole sizes	11
1.3 Pressure peaking phenomenon	12
1.4 Safety measures in HRS	13
1.4.1 Leak detection systems	13
1.4.2 Flow meters	13
1.4.3 Emergency shut-off	13
1.4.4 Fire suppression	13
1.4.5 Firewalls	14
2 Methodology	15
2.1 KFX	15
2.1.1 Governing equations	15
2.2 Description of the domain	17
2.2.1 Forced ventilation	18
2.3 Computational grid	19
2.4 Boundary conditions	19
2.5 Initial conditions	19
3 Results	22
3.1 Hole sizes	22
3.2 Forced ventilation sizing	22
3.3 Sensitivity studies	24
3.3.1 Grid	24
3.3.2 Time step	24
3.3.3 Height	26
3.3.4 Direction	26

3.4	Unignited leaks	29
3.4.1	Exposure to flammable gas	29
3.5	Ignited leaks	37
3.5.1	Dynamic pressure	37
3.5.2	Radiation	39
3.6	Safety barriers: effect of the firewall	44
4	Discussion	47
4.1	Overall severity rankings	47
4.2	Main effects analysis	47
4.2.1	Flammable area	48
4.2.2	Flammable volume	48
4.2.3	Dynamic pressure	48
4.2.4	Radiation	49
4.2.5	General conclusions	49
4.3	Scatter plot	52
	Conclusion and further work	53
	Bibliography	55
A	Additional figures	60
A.1	Exposure to flammable gas	60
A.2	Radiation	76
A.3	Consolidated analysis	87
B	Python scripts	89
B.1	Sensitivity studies	89
B.1.1	Sensitivity analysis	89
B.1.2	Ignition probability	101
B.2	Flammable area and volume	102
B.2.1	Area	102
B.2.2	Volume	122
B.3	Dynamic pressure	130
B.3.1	Pressure peaking phenomenon	144
B.4	Radiation	165
B.5	Consolidated analysis	179
B.5.1	Main	179
B.5.2	Plots	191

B.5.3	Scores	216
B.5.4	Utilities	222
C	Geometry files	224
C.1	Forced ventilation	224
C.2	Natural ventilation	230
D	List of simulations	236

List of Figures

2.1	Example of grid.	16
2.2	Refuelling station design from [26]	17
2.3	Compressor in a (a) container, (b) shed.	18
2.4	Windrose for Voll, Trondheim, for the last 30 years, from [31].	20
3.1	Highest efficiency point of the fan at a given system pressure. Image generated by the AirBox software, red lines drawn on top of it to help locate the point in the axes.	23
3.2	Ignition probability for the different grid resolutions.	24
3.3	Volume between LFL and UFL over time for the different grid resolutions.	25
3.4	Volume between LFL and UFL over time for the different time step configurations.	25
3.5	Floorplan of the enclosure, pressure zones and leak positions.	27
3.6	Sensitivity study of the different heights for each of the locations.	28
3.7	Sensitivity study of the different directions for each of the locations.	30
3.8	Flammable area projection at steady state for 430bar, 10% area cases.	33
3.9	Flammable area over time for 430bar, 10% area cases.	34
3.10	Volume between LFL and UFL over time for 430bar, 1% area cases.	34
3.11	Volume between LFL and UFL over time for 900bar, 1% area cases.	34
3.12	Summary of the cloud projection of the flammable area and maximum reach at steady state for all the cases organised by a hierarchy of hole size, then pressure, then direction.	35
3.13	Summary of the flammable volume of the cloud at steady state for all the cases organised by a hierarchy of hole size, then pressure, then direction.	36
3.14	Dynamic pressure in steady state for 430bar, full bore, -Z, one of the most dangerous scenarios.	37
3.15	Results of the pressure peaking phenomenon model for the full bore scenarios.	38
3.16	Radiation in steady state for 430bar, 10% area cases. The score has units of m^2 , but doesn't directly represent physical area, but a severity-weighted exposed area.	42
3.17	Summary of the radiation scores at steady state for all the cases organised by a hierarchy of hole size, then pressure, then direction.	43
3.18	Cumulative areas affected by radiation, and volume of the flammable cloud.	44
3.19	Flammable area projection for each of the wall heights, applied to the worst case.	45
3.20	Radiation for each of the wall heights, applied to the worst case.	46

4.1	Raking on the cases based on their combined score. The ranking is truncated to 24 positions to improve visualisation.	48
4.2	Main effects of the different variables on the flammable area.	50
4.3	Main effects of the different variables on the flammable volume.	50
4.4	Main effects of the different variables on the dynamic pressure.	51
4.5	Main effects of the different variables on the radiation score.	51
4.6	Scatter confronting flammable area and volume, radiation score and dynamic pressure score.	52
A.1	Flammable area projection at steady state for 430bar, 1% area cases.	61
A.2	Flammable area projection at steady state for 430bar, full bore cases.	62
A.3	Flammable area projection at steady state for 630bar, 1% area cases.	63
A.4	Flammable area projection at steady state for 630bar, 10% area cases.	64
A.5	Flammable area projection at steady state for 630bar, full bore cases.	65
A.6	Flammable area projection at steady state for 900bar, 1% area cases.	66
A.7	Flammable area projection at steady state for 900bar, 10% area cases.	67
A.8	Flammable area projection at steady state for 900bar, full bore cases.	68
A.9	Flammable area over time for 430bar, 1% area cases.	69
A.10	Flammable area over time for 430bar, full bore cases.	69
A.11	Flammable area over time for 630bar, 1% area cases.	69
A.12	Flammable area over time for 630bar, 10% area cases.	70
A.13	Flammable area over time for 630bar, full bore cases.	70
A.14	Flammable area over time for 900bar, 1% area cases.	70
A.15	Flammable area over time for 900bar, 10% area cases.	71
A.16	Flammable area over time for 900bar, full bore cases.	71
A.17	Volume between LFL and UFL over time for 430bar, 10% area cases.	71
A.18	Volume between LFL and UFL over time for 430bar, full bore cases.	72
A.19	Volume between LFL and UFL over time for 630bar, 1% area cases.	72
A.20	Volume between LFL and UFL over time for 630bar, 10% area cases.	72
A.21	Volume between LFL and UFL over time for 630bar, full bore cases.	73
A.22	Volume between LFL and UFL over time for 900bar, 10% area cases.	73
A.23	Volume between LFL and UFL over time for 900bar, full bore cases.	73
A.24	Summary of the cloud projection of the flammable area and maximum reach at steady state for all the cases organised by a hierarchy of pressure, then hole size, then direction.	74
A.25	Summary of the flammable volume of the cloud at steady state for all the cases organised by a hierarchy of pressure, then hole size, then direction.	75
A.26	Radiation in steady state for 430bar, 1% area cases.	76
A.27	Radiation in steady state for 430bar, full bore cases.	77
A.28	Radiation in steady state for 630bar, 1% area cases.	78
A.29	Radiation in steady state for 630bar, 10% area cases.	79
A.30	Radiation in steady state for 630bar, full bore cases.	80
A.31	Radiation in steady state for 900bar, 1% area cases.	81
A.32	Radiation in steady state for 900bar, 10% area cases.	82

A.33 Radiation in steady state for 900bar, full bore cases.	83
A.34 Summary of the radiation scores at steady state for all the cases organised by a hierarchy of pressure, then hole size, then direction.	84
A.35 Summary of the radiation scores at steady state for all the cases organised by a hierarchy of hole size, then pressure, then direction.	85
A.36 Summary of the radiation scores at steady state for all the cases organised by a hierarchy of pressure, then hole size, then direction.	86
A.37 Heatmap with all the cases of the study.	87
A.38 Raking on the cases based on their combined score.	88

List of Tables

1.1 Properties of hydrogen gas compared to other fuels measured at standard temperature and pressure (STP). From [2]	11
2.1 Quality specifications of the grid.	19
3.1 Hole sizes for the different scenarios. In the 900bar medium leak case, the 0.8mm hole is crossed out to clarify that it is the correct theoretical value, but in practice the study used a 1.0mm hole because of technical limitations. . .	22
3.2 Leak locations.	26
3.3 Expected structural damage caused by an explosion depending on peak over-pressure, from [36]	39
3.4 Radiation thresholds used in the severity score, from [37, 38].	40
3.5 Most and least severe observed directions for the different pressure levels and types of ventilation.	41

Introduction

In the last few years, hydrogen has gained popularity as a potential nonpolluting and renewable energy vector. Depending on the technology being used to produce it, it could also aid in increasing energy security. However, it is surrounded by a number of safety concerns, as well as the shadow of past accidents. The compressor is typically a component with high failure rates, and the characteristics of hydrogen may exacerbate the problem. This means that it is reasonable to think that hydrogen leaks will be more likely in the compressor enclosure, so its design should be informed.

This work methodologically explores the consequences of hydrogen leaks in the compressor enclosure using CFD, considering both natural and forced ventilation designs, analysing how the consequences vary with pressure, hole size, and direction, with the goal to answer the following research questions:

1. What is the time-dependent area exposed to flammable gas and the combustible volume generated by an unignited leak?
2. What is the dynamic pressure at ground level for an unignited leak?
3. What is the heat radiation the surroundings are exposed to in the case of an ignited hydrogen leak?
4. What is the effect of the firewall, and what is its optimal height?

The answers to these questions should paint a clearer picture of what the risks are and what design aspects should be prioritised.

The text is structured as follows: the work starts with a brief literature review, followed by a description of the methodological aspects that have shaped the project. The results are then presented, from the chosen hole sizes and sensitivity analyses that were used to shape the project, to the answers to the research questions presented above. The findings are collected in the discussion chapter, which attempts to combine them and further analyse them. In the appendixes, the reader will find a large number of figures which illustrate the generated data and the scripts that were used in the postprocessing, amongst other material.

Chapter 1

State of the art

Hydrogen is the lightest gas. It has a low ignition energy and a wide flammability range (from 4 to 75%), and is buoyant and odourless [1]. The lower flammability limit is not exceptional, but the fact that the flammability interval is so wide makes it prone to ignition in more situations. Its flame is also almost invisible, which makes its detection more difficult [2]. The flame is difficult to suppress and the fastest amongst common fuels at propagating: its laminar speed can exceed 3m/s [3]. Its small molecular size and high diffusivity help in its dispersion in the atmosphere in the event of a leak, but also aid its leakage through micro-cracks, faulty seals or porous materials. Because of its low density, it is buoyant, and can easily accumulate under ceilings if the leak happens in a confined space [4]. The properties of hydrogen are summarised in table 1.1, which also includes the characteristics of other common fuels for comparison.

As hydrogen gains popularity as an automotive fuel, an increase in the number of refuelling stations can be expected, and as a result, an increase of accidents in these sites may also be expected [5]. In the case of transportation by piping, safety concerns include hydrogen embrittlement, ensuring compatibility with existent infrastructure, and managing leaks in densely populated areas [6]. Research is also underway to improve the materials for high pressure hydrogen storage, since even small cracks due to hydrogen embrittlement can have serious consequences [5]. This degradation process affects high-strength steels, nickel-based alloys and titanium. In polymers, like the types used in sealing applications, it can cause cracking, swelling, or permeability changes [2]. The concerns regarding leaks, overpressurisation and material degradation are present at every stage of the hydrogen value chain. Other less frequent but potentially dangerous hazards are the risk of ice formation or equipment damage from cryogenic hydrogen leaks and exposure to toxic chemical carriers like ammonia or hydrides that could be used for storage and transport [2].

1.1 Evaluating the frequency of hydrogen incidents

Recent risk assessments have determined that the highest risk of leakage comes from the fuel supply unit, followed by the storage, pressurisation, unloading, and transport units [7]. Hydrogen leakages have been modelled using a variety of techniques, including deep learning

Table 1.1: Properties of hydrogen gas compared to other fuels measured at standard temperature and pressure (STP). From [2]

Property	Hydrogen	Gasoline vapour	Methane
Density (kg/m ³)	0.0899	3	0.68
Flammability range (%)	4–75 (high risk)	1.4–7.6 (low risk)	5–15 (moderate risk)
Minimum ignition energy (mJ)	0.02 (easily ignitable)	0.29 (safer)	0.28 (safer)
Flame visibility (scale 1-5)	1 (invisible flame)	5 (highly visible)	3 (moderately visible)
Laminar flame speed (m/s)	2.65–3.25 (high)	0.4–0.5 (slow)	0.35–0.45 (slow)
Gas dispersion rate in air (cm ² /s)	0.61 (very fast)	0.05–0.07 (slow)	0.16 (moderate)

and Bayesian dynamic networks [8–10]. The most critical elements appear to be joints, seals, protection of pipes against external impacts and prevention of corrosion and accumulation of particles in pipes [8]. In addition, human error and corrosion are two very important factors contributing to hydrogen leakage accidents [11]. A review of incidents which happened on industrial hydrogen processing plants selected 32 incidents and focused on valves, piping, flanges, storage tanks and process vessels. It identified five root causes: engineering (33.3%), maintenance (23.5%), design error (15.7%), operator error (15.7%) and management (11.8%) [12]. Another study concerning 120 incidents in the period 1999-2019 found that the main common cause was failure of pipes, valves and filters [13]. A study on the HIAD 2.0. database concluded that the most common causes were maintenance operations and equipment failures [14]. Another HySET thesis which was developed in parallel to this one [15] performed a thorough analysis on hydrogen incidents in HRS utilising the information found in other databases, and one of the findings was that the highest fraction of reported incidents had started in the compressor, followed by the tube trailer, the filling hose, and the shut-off valve. Regarding the ignited incidents only, the compressor is again ranked first, followed by the tube trailer, the electrolyser, and storage.

1.2 Hole sizes

[16] proposed to study the effects of leaks in hole diameters of 0.5, 0.7 and 1mm. [17] studied the release characteristics of highly pressurised hydrogen through the same selection of hole sizes. These were the sizes of holes chosen because when KIST (the Korea Institute of Science and Technology) measured the concentration of a released hydrogen jet from a highly pressurised chamber, to mimic the environment of a high pressure vessel, they used these

hole sizes. The hole diameters were chosen in such a way that the hole area increased by approximately two for each hole size. They could not use bigger holes because the duration of the jet would have been too short. In fact, they failed to measure the hydrogen concentration at 400bar (the highest pressure they were studying) at 1mm [18]. Another study [19] found that for an isentropic choked leak, for hydrogen stored at 690 bar, a pinhole leak as small as $0.4\mu\text{m}$ could support a stable flame.

Another approach is to define the hole sizes based on the fractional area of the section of the pipe, as described in HyRAM [20], which is a wider methodology for assessing hydrogen safety in a variety in applications. It's often used in quantitative risk analysis (QRA) applications. The release is distretised in fractional areas of system pipe flow area: 0.01%, 0.1%, 1%, 10%, 100%.

1.3 Pressure peaking phenomenon

The pressure peaking phenomenon (PPP) becomes relevant in this study because of its proximity to dynamic pressure.

The PPP is a peak in the pressure transient which can be powerful enough to compromise the enclosing structure. In some cases, this pressure peak can be of an order of magnitude higher than the overpressure that is eventually reached when the enclosure is fully filled with the leaked gas. It is pronounced only in gases that are lighter than air, eg. hydrogen and helium, when they are released in enclosing structures with one or more vents, when the release is large enough that the air is replaced by the lighter gas. Its magnitude depends on the release flow rate, hydrogen inventory, enclosure volume and ventilation area [21].

A study [21] proposed a model to predict the development of overpressure with time within the enclosure. This model assumes uniform concentration, so that the leaked gas mixes perfectly with the previous atmosphere inside the enclosure. It also assumes that there is no air ingress.

$$m_{encl}^{t+\Delta t} = m_{encl}^t + (\dot{m}_{nozz}^t - \dot{m}_{vent}^t)\Delta t \quad (1.1)$$

$$n_{encl}^{t+\Delta t} = n_{encl}^t + \left(\frac{\dot{m}_{nozz}^t}{M_{H_2}} - \frac{\dot{m}_{vent}^t n_{encl}^t}{m_{encl}^t} \right) \Delta t \quad (1.2)$$

$$P_{encl(t_{a+1})} = \frac{n_{encl}^{t+\Delta t} RT}{V} \quad (1.3)$$

$$\dot{m}_{vent}^{t+\Delta t} = C \left(\frac{m_{encl}^{t+\Delta t} A}{V} \right) \left(\frac{2(P_{encl}^{t+\Delta t} - P_O)V}{m_{encl}^{t+\Delta t}} \right)^{\frac{1}{2}} \quad (1.4)$$

where m is the mass, M is the molecular mass, n is the number of moles, R is the universal gas constant, T is the temperature, t is time, V is the volume, and C is the discharge coefficient, for which the paper uses a value of 0.6.

1.4 Safety measures in HRS

Only the effect of the firewall is included in this study, but an overall view of the safety systems is nevertheless provided for the sake of completion.

1.4.1 Leak detection systems

The sensors used to detect hydrogen are based on principles such as thermal conductivity and electrochemical reactions, and detection based on semiconductors. The aim is to detect hydrogen at very low concentrations. They should be strategically positioned, close to areas prone to leaks, such as dispensers, storage tanks and pipelines. Their effectivity depends on their sensitivity and response time, and they need to work consistently under a range of environmental conditions. Maintenance is also very important, and the operational time of each type of sensor should always be taken into consideration [22].

1.4.2 Flow meters

Flow meters are relevant to safety because the correct evaluation of the rate of hydrogen transfer is highly relevant to avoid overpressure issues. A few technologies are used to this end, based on the Coriolis effect, ultrasounds, electromagnetic conductivity, and pressure difference. The Coriolis flow meter is highly precise and calculates the flow by evaluating its impact on a vibrating tube. The ultrasonic flow meter measures how much time it takes for ultrasonic pulses to travel. It's very resilient and doesn't need maintenance, but it's not as precise as the Coriolis flow meter. Electromagnetic flow meters aren't recommended because of hydrogen not being conductive. Differential pressure meters measure the flow by measuring the decrease in pressure caused by an obstruction in the flow path. They're economical, but their precision can be affected by variations in the environmental conditions [22].

1.4.3 Emergency shut-off

Emergency shut-off systems are designed to stop the flow of hydrogen in case of an emergency. They can be manual or automatic —triggered by the safety systems of the station, typically controlled by the gas management panel—. Breakaway couplings —installed on the hoses to safely detach them in case they were pulled excessively— also fall under this category. These systems are important because they aim to quickly stop the release of hydrogen and in so doing minimise the chances of ignition and explosion, therefore protecting both the refuelling station and the people and surroundings [22].

1.4.4 Fire suppression

Commonly used methods for fire suppression in hydrogen refuelling stations include water mist systems, dry chemical suppressants, and gas-based suppression. Water mist systems are highly effective, also at preventing reignition, and work by cooling the area and dragging the oxygen down, so the fire is suffocated. Dry chemicals disrupt the chemical reaction of

the fire. They are good at stopping a fire but not as effective in preventing reignition. Gas-based suppression systems work by displacing oxygen or hindering the chemical reactions that support the fire. Their effectivity is moderate and can only be used in enclosed areas [22].

1.4.5 Firewalls

The outer temperature of the hydrogen flame exceeds 2100°C. Firewalls have a beneficial effect by impeding the heat radiation caused by a hydrogen fire, but they can also make it difficult for the smoke or gas to disperse [5].

A study [23] found that the ability of the firewall to hinder lateral flame spread is heavily dependent on how tall the wall is relative to the leak: if their heights are the same, the flame cannot be stopped. Another study [5] used OpenFOAM to simulate scenarios of 35, 45, 70 and 90MPa, the most dangerous scenario being the complete rupture of the pipeline, a leakage hole of 10mm at a height of 1m. The firewalls considered in this study are the ones required by the Chinese National Standard GB 50203-2021 and are 240mm thick and constructed of fireproof solid bricks. They found that the buoyancy of hydrogen causes the flame to float upwards, and the flame above the firewall increased with firewall height whilst restricting its lateral range. They found that the firewall height should be 2m taller than the leakage point, and have widths of 2.5m if the storage pressure was 35-45MPa, and 3m if 70-90MPa. The safe distances ranged from 8.4 to 9.3m, and the leakage hole had to be kept from 4 to 7m away from the firewall.

Sandia National Laboratories performed a set of experiments to evaluate the effectiveness of barrier walls to reduce hazards [24]. They concluded that a 3-wall 135°C configuration was the most effective at mitigating the effects of overpressure, thermal radiation, and unignited jet release.

Chapter 2

Methodology

2.1 KFX

KFX is a software owned by DNV which is specifically developed for safety-related applications and uses computational fluid dynamics to simulate dispersion and fire scenarios [25]. It utilises finite volume methods to solve the Navier-Stokes equations.

By default, the solvers used in KFX are QTDMA for transport equations and STONE for pressure correction and partial differential equations. However, if the STONE solver fails, it automatically switches to QTDMA. For the difference scheme, it uses a mix between first and second order scheme. Usually, a mix of 90% second order and 10% first order is used. This is done to improve false diffusion.

A grid is created specifically for each case. An example of grid can be seen in figure 2.1. In the simulations that use forced simulation, a custom cell was used, and the grid was refined manually in these cases to make sure the placement of the fan was correct in the simulations.

2.1.1 Governing equations

For the gas phase, the program uses equations (2.1) to (2.8). The equations are presented in time averaged form, unless stated otherwise.

Species mass fraction. Mass balance equation.

$$\frac{\partial \bar{\rho} \tilde{Y}_l}{\partial t} + \frac{\partial \bar{\rho} \tilde{u}_j \tilde{Y}_l}{\partial x_j} = - \frac{\partial}{\partial x_j} (\bar{\rho} Y_l V_{lj}) - \frac{\partial}{\partial x_j} (\bar{\rho} u_j'' Y_l'') + \bar{\rho} \tilde{R}_l + \bar{\rho} \tilde{R}_{\text{liq},l} \quad (2.1)$$

where ρ is the gas density, Y_l is the species mass fraction, and V_{lj} is the molecular diffusion velocity of species l in the direction j . R_l is the chemical source term and $R_{\text{liq},l}$ is the source due to liquid phase transition.

Continuity equation. Conservation of mass.

$$\frac{\partial \bar{\rho}}{\partial t} + \frac{\partial \bar{\rho} \tilde{u}_j}{\partial x_j} = \bar{\rho} \tilde{R}_{\text{liq}} \quad (2.2)$$

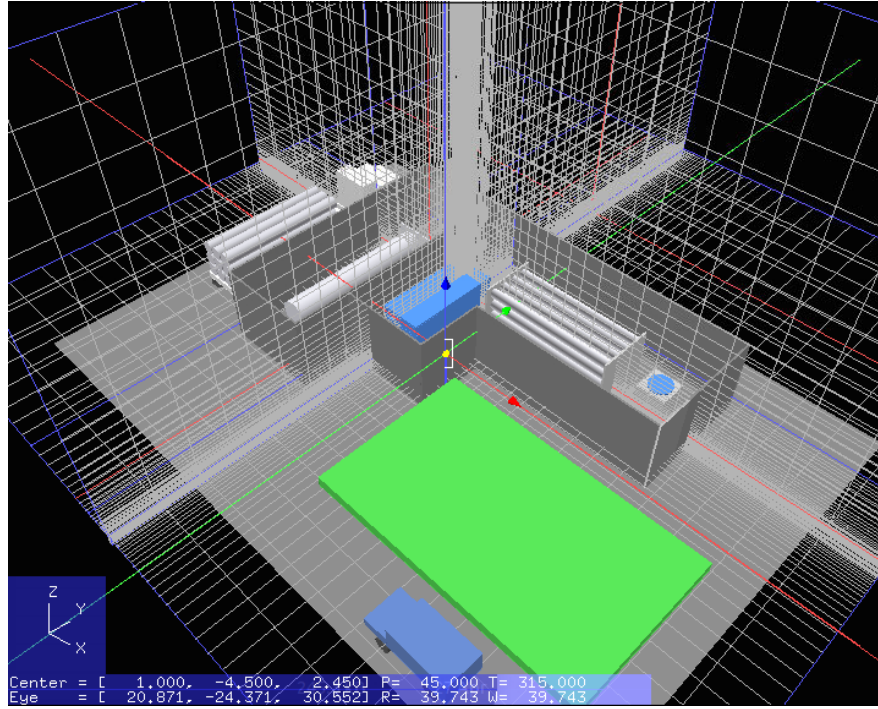


Figure 2.1: Example of grid.

where ρ is the density of the gas and R_{liq} is the source due to liquid phase transition.

Momentum equations. Conservation of momentum along the coordinate directions are given by the Navier-Stokes equations, expressed in cartesian tensor notation.

$$\frac{\partial \bar{\rho} \tilde{u}_i}{\partial t} + \frac{\partial \bar{\rho} \tilde{u}_j \tilde{u}_i}{\partial x_j} = -\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j} \left(\bar{\tau}_{ij} - \overline{\rho u_j'' u_i''} \right) + \bar{\rho} f_i + \bar{\rho} \tilde{F}_{\text{liq},i} \quad (2.3)$$

where f_i are mass forces by which the gas is influenced, and τ_{ij} is the stress tensor:

$$\bar{\tau}_{ij} = \mu \left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i} \right) + \left(\kappa - \frac{2}{3} \mu \right) \left(\frac{\partial \tilde{u}_k}{\partial x_k} \right) \delta_{ij} \quad (2.4)$$

and F_{liq} is the source due to liquid phase interaction, p is the absolute pressure, κ is the second viscosity coefficient, and δ_{ij} is the Kronecher delta.

Energy equation for compressible gas flows. Ideal gas behaviour is assumed.

$$\frac{\partial}{\partial t} (\bar{\rho} \tilde{e}_T) + \frac{\partial}{\partial x_j} (\bar{\rho} \tilde{u}_j \tilde{e}_T) = \frac{\partial}{\partial x_j} \left((\bar{\tau}_{ij} - p) u_j \right) + \frac{\partial}{\partial x_j} \left(k \frac{\partial T}{\partial x_j} - \overline{\rho u_j'' e_T''} \right) + \bar{Q}_{\text{gs}} + \bar{Q}_{\text{Rad}} + \bar{\rho} \tilde{S}_{\text{liq}} \quad (2.5)$$

where the total energy is

$$e_T = e + \frac{1}{2} u_i u_i \quad (2.6)$$

and the internal energy is

$$e = \sum_l Y_l e_l(T) \quad (2.7)$$

General gas phase transport equation. The model form of this equation is

$$\underbrace{\frac{\partial(\bar{\rho}\tilde{\Phi})}{\partial t}}_{\text{I}} + \underbrace{\frac{\partial(\bar{\rho}\tilde{u}_j\tilde{\Phi})}{\partial x_j}}_{\text{II}} = \underbrace{\frac{\partial}{\partial x_j}\Gamma_{\Phi,\text{eff}}\frac{\partial\tilde{\Phi}}{\partial x_j}}_{\text{III}} + \underbrace{\bar{\rho}\tilde{S}_{\Phi}}_{\text{IV}} \quad (2.8)$$

where **I** is the transient term, **II** is the convective term, **III** is the diffusive term, which includes both laminar diffusion and diffusive effects caused by turbulence, and **IV** is the source term for the property Φ , and includes other effects not included in the previous terms.

2.2 Description of the domain

This thesis builds on a previous HySET student's work [26] by focusing on the compressor and evaluating different scenarios. The geometry for the refuelling station then developed has therefore been reused and adapted, and can be seen in figure 2.2 and in 3D [here](https://youtu.be/BM2A4hM146A) or on <https://youtu.be/BM2A4hM146A>. The location was also defined in that study: Voll, Trondheim. The compressor is a three stage compressor, with pressure values of 430bar, 630bar and 900bar. Since three stage compressor can be conceptualised as three smaller compressors in series, three zones are naturally established: a low pressure side, an intermediate pressure zone in the middle, and a high pressure side, each with their respective leak location (see figure 3.5). Because the compressor is not placed symmetrically inside the container, the location of the high pressure side is established to be the one furthest from the wall, the reason being that the cloud should be able to disperse more easily this way.

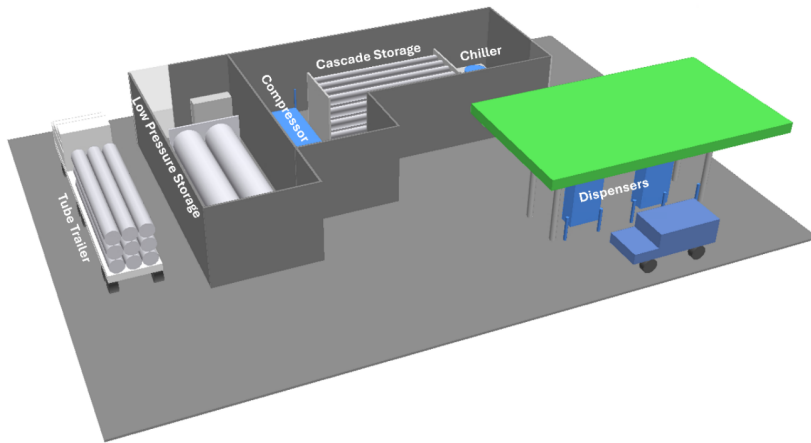


Figure 2.2: Refuelling station design from [26]

One of the objectives of this project is to evaluate the extent of the effect of the ventilation system of the compressor in the event of a hydrogen leak. To study this, two configurations are considered: the first is the most common one, where the compressor is placed inside of a container (figure 2.3a). The use of a container implies that noise control is easier, but it also

has the drawback that, in the event of a leak, hydrogen is more likely to accumulate. This is why the container has vents, and makes use of forced ventilation (see section 2.2.1). The second configuration consists of the compressor being placed in a simple shed (figure 2.3b). The inside dimensions and height of the shed are the same as in the case of the container: 5m in length, 2m in width, 2m in height. The roof is V-shaped to prevent the accumulation of hydrogen at the top and aid its evacuation. Lastly, the effect of the height of the firewall is also studied. The geometry files used in this project can be found in the appendix.

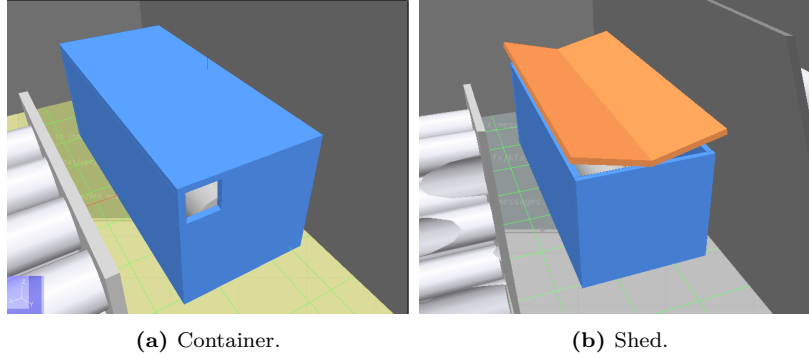


Figure 2.3: Compressor in a (a) container, (b) shed.

Each stage of the compressor uses a different size of piping. The pipe diameters to be used in each stage are established based on a report from the Idaho National Laboratory [27]. The hole sizes are defined according to the fractional pipe area approach from [20]. In this study, only medium leaks (1% of the pipe section), large leaks (10% of the pipe section) and full bore (100%) scenarios are considered because of technical limitations —i.e. the program is not able to successfully simulate leaks smaller than this because the pipes are already quite small—. However, it is worth mentioning that different works organise the leaks in different ways. According to the guidelines published by the government of Hong Kong [28], the diameters chosen for this study fall under the 8-16mm bracket and are closest to what they categorise as very small and small leaks.

2.2.1 Forced ventilation

The required forced ventilation for the container is calculated according to IEC 60079-10-1 [29] and following the requirements in NFPA 2-2020 (6.18.1) [30]. The purpose of this methodology is to size a ventilation system that is able to disperse a credible and sustained leak under normal (or near normal) operation. For the case of the compressor, it is reasonable to assume that (1) the leaks are expected to be secondary —i.e. they are not expected in normal operation, but are nevertheless possible—, and (2) the upstream pressure is that of the compressor. Because of these reasons, all of the leaks considered in these study are too large for designing the mechanical ventilation, and using those would lead to an unrealistic ventilation system. The calculations are detailed in section 3.2.

2.3 Computational grid

The grid used for the study is about 300 000 cells. KFX creates a grid which is reasonably close to the number given by the user, and the grid is generally created for each specific case, which in practice means that all the grids will be different to some degree. However, a grid independence test was done for one of the scenarios (see section 3.3.1).

The quality specifications of the chosen grid can be found in table 2.1. As a general orientation to assess grid quality, it is desirable to have an aspect ratio as close as possible to 1. Therefore, we can conclude that the chosen grid is of decent quality. However, as discussed, not exactly the same grid was used for each of the scenarios, so the specifications may vary between them.

Table 2.1: Quality specifications of the grid.

Grid Quality Parameters	
Largest stretch in the x-direction	1.04588 at i=18
Largest stretch in the y-direction	1.04289 at j=9
Largest stretch in the z-direction	1.04151 at k=38
Largest aspect ratio	4.93208 (39,2,11)
Average aspect ratio	1.93295
Number of nodes	313600 (80 * 80 * 49)
Domain low coordinate	(-20,-20,1.83187e-15)
Domain high coordinate	(20,20,20)

2.4 Boundary conditions

The boundary conditions refer to the wind conditions. Only one wind condition was taken into consideration in this study, which corresponds to the most likely wind direction and speed in Voll according to the wind rose for the last 30 years (see figure 2.4). Specifically, the wind direction was calculated to be given by an angle of 202.5° relative to north, and a wind speed of 9.7m/s.

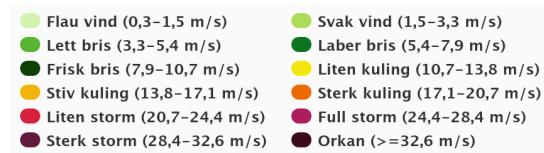
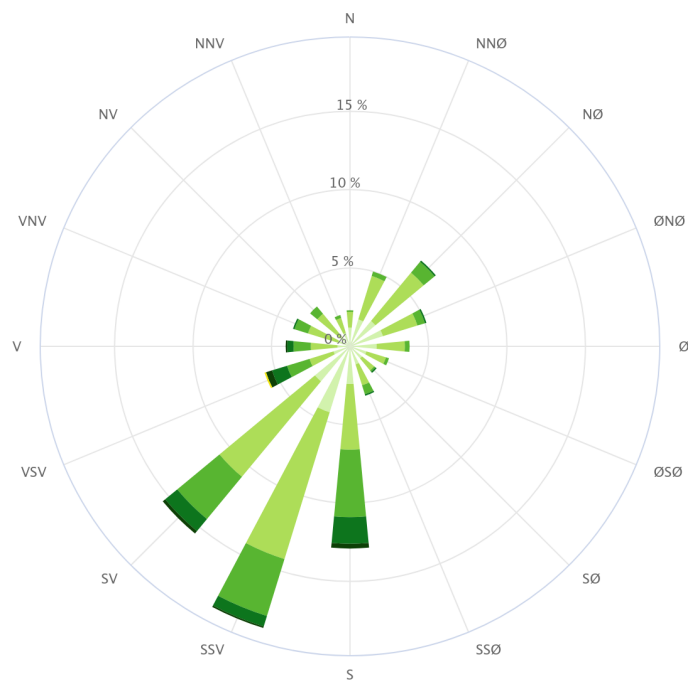
2.5 Initial conditions

The initial conditions are given by each of the leaks considered in this study. The parameters that are varied are the following:

- **Ventilation type:** Natural ventilation (shed) or forced ventilation (container).
- **Pressure:** The studies considers leak in each of the stages of a three stage compressor: 430, 630 and 900bar. Therefore, each of the pressures is linked to a position.

Vindrose for Trondheim – Voll (68860) i perioden; 2.1996–2.2026.

Stille (0,0–0,2 m/s) = 2,2 %



Highcharts.com

Figure 2.4: Windrose for Voll, Trondheim, for the last 30 years, from [31].

- **Height:** In the intermediate pressure section, the leak is assumed to be on top of the compressor. However, in the low and high pressure sections, three heights were compared and the worst one was chosen (see section 3.3.3).
- **Direction:** All six directions ($-X$, $+X$, $-Y$, $+Y$, $-Z$, $+Z$) were initially considered. A sensitivity analysis was done to establish which were the most relevant directions for each of the locations from a safety point of view, in order to optimise the number of simulations (see section 3.3.4).
- **Hole size:** As described in section 3.1, the hole sizes correspond to a leak that is 1%, 10% and 100% of the area of the pipe.

The combination of this variables gives a total of 54 simulations, which are to be run twice (ignited and unignited). The complete list of simulations can be found in the appendix. These combinations are input to KFX, where a few additional internal variables (flowrate, equivalent diameter, expanded velocity) are calculated before creating the necessary files to send to the cluster and proceed with the simulation.

Chapter 3

Results

The results in this chapter, unless stated otherwise, refer to the steady state.

3.1 Hole sizes

As described in section 2.2, the hole sizes are calculated according to the fractional area approach, based on the size of piping for each pressure region. The medium leak corresponds to a hole of 1% of the area, the large leak represents 10% of the area, and the full bore represents a scenario where the pipe breaks completely. The hole diameters chosen for each of the pressure levels are detailed in table 3.1.

Because of the technical limitations, even though the medium leak at 900bar corresponds to a hole whose diameter is 0.8mm, it is simulated with a 1.0mm hole instead because it proved to be too small to be simulated successfully.

Table 3.1: Hole sizes for the different scenarios. In the 900bar medium leak case, the 0.8mm hole is crossed out to clarify that it is the correct theoretical value, but in practice the study used a 1.0mm hole because of technical limitations.

Pressure [bar]	Hole diameter [mm]		
	Medium leak	Large leak	Full bore
430	1.2	3.8	12.0
630	1.0	3.2	10.0
900	0.8 1.0	2.5	8.0

3.2 Forced ventilation sizing

To design the ventilation, as a starting point, the requirements in NFPA 2-2020 (6.18.1) [30] are used. In this directive, the minimum ventilation is $0.0051\text{m}^3/\text{s}$ per m^2 of floor area. Since the floor area of the container is 10m^2 , the minimum required ventilation is $0.051\text{m}^3/\text{s}$. With

this number in mind, we look for a fan that we could realistically use for this application. The chosen fan is a Ø400mm Novenco ZerAx AZN ATEX fan [32]. This range of fans are unidirectional ducted axial flow fans, and they have versions suitable for installation in explosive environments. Using the AirBox software from the manufacturer [33], we can find the highest efficiency point, which corresponds to a system resistance of approximately 304Pa, and a volumetric flow of 1.45m³/s, with an efficiency of 86% (figure 3.1).

The resistance of the system is assumed to be 304Pa because that is the point that corresponds to the highest efficiency point of the fan. This value would account for inlet pressure losses, but also losses inside the container, which can be significant due to the amount of machinery present inside the container, even though they cannot be calculated accurately without a detailed equipment layout. The container would realistically house the three compression stages, intercoolers, piping, instrumentation, and other auxiliary equipment, which means that the actual free volume for airflow is significantly less than the total inside volume. In any case, this is a conservative assumption, and if the system resistance is less than the assumed one, the fan would simply operate with lower efficiency.

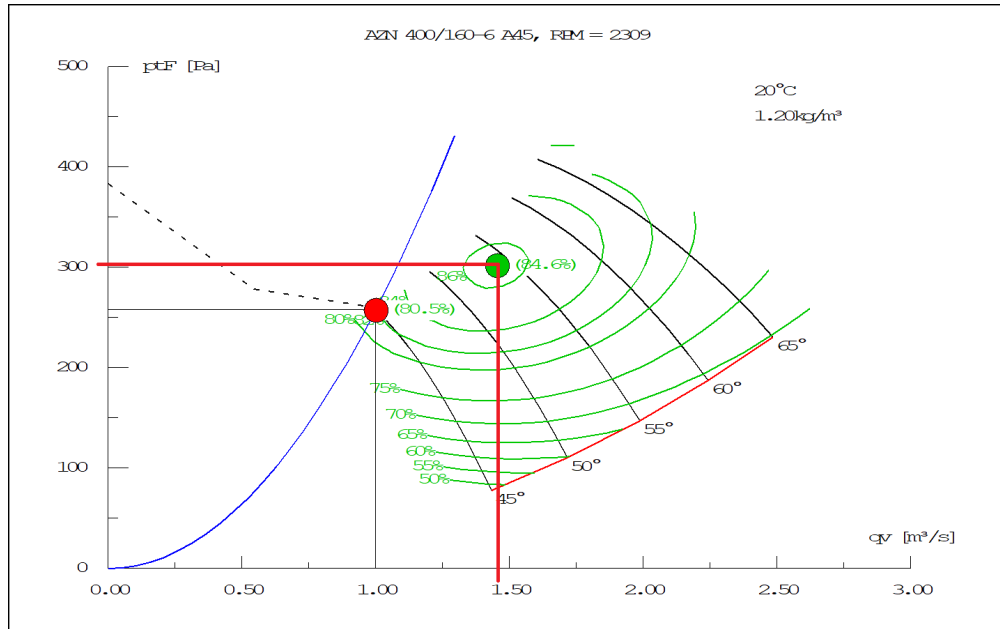


Figure 3.1: Highest efficiency point of the fan at a given system pressure. Image generated by the AirBox software, red lines drawn on top of it to help locate the point in the axes.

In order to be able to run the simulations, the information obtained from the manufacturer needs to be translated into air speed and mass flow rate so these values can be used to configure the custom cell. The inlet area of the fan is

$$A_{inlet} = \pi \times 0.2^2 = 0.1257m^2$$

The air speed at the inlet is therefore

$$u_{inlet} = \frac{1.45}{0.1257} = 11.5m/s$$

We can calculate the mass flow of air using its density at 15°C and sea level [34] and the volumetric flow

$$\dot{m}_{air} = 1.45 \times 1.225 = 1.776 kg/s$$

3.3 Sensitivity studies

3.3.1 Grid

To choose a proper grid size, which ensures an accurate depiction of the results with the minimum computational time, a grid sensitivity study was done. The same scenario was run with 7 different grid resolutions, with the number of cells ranging from approximately $1 \cdot 10^5$ to $7 \cdot 10^5$. Figure 3.2 shows the calculated ignition probabilities at $t=75s$, whilst figure 3.3 shows the volume between the lower flammability limit (LFL) and the upper flammability limit (UFL) over time for the different cases. In light of these results, a grid with about $3 \cdot 10^5$ cells is chosen, as it appears to provide the best accuracy to computational time ratio available.

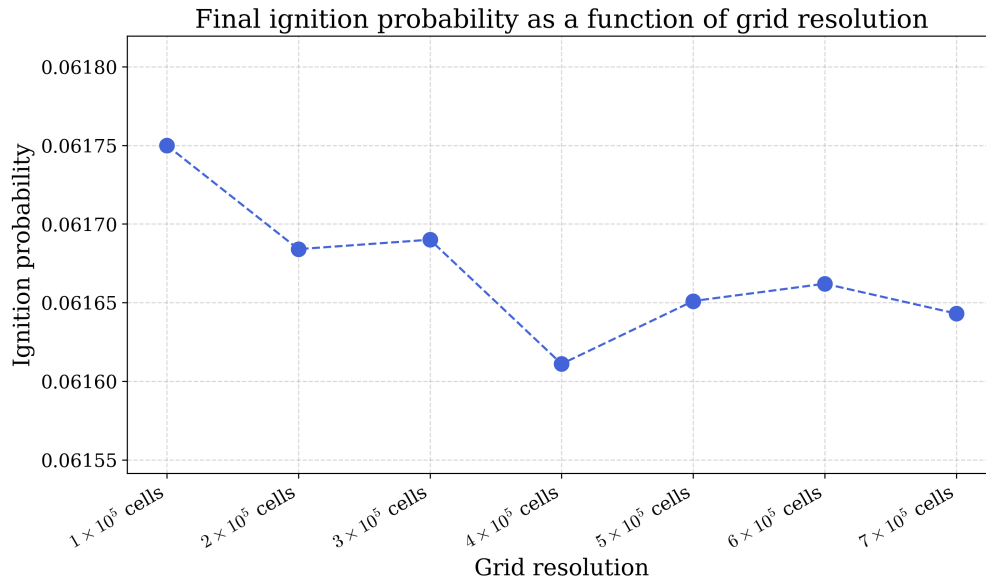


Figure 3.2: Ignition probability for the different grid resolutions.

3.3.2 Time step

The same thing was done in order to determine the time step and hence minimise computational time. It was found that a time step of 0.05s produced good results in general —see figure 3.4—, but in some specific cases, a smaller time step had to be used because of stability issues.

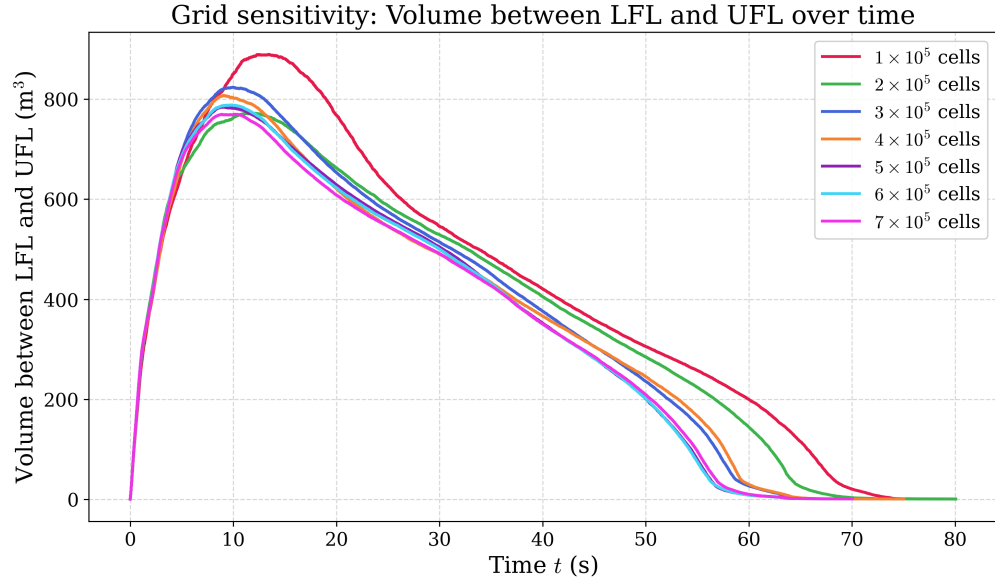


Figure 3.3: Volume between LFL and UFL over time for the different grid resolutions.

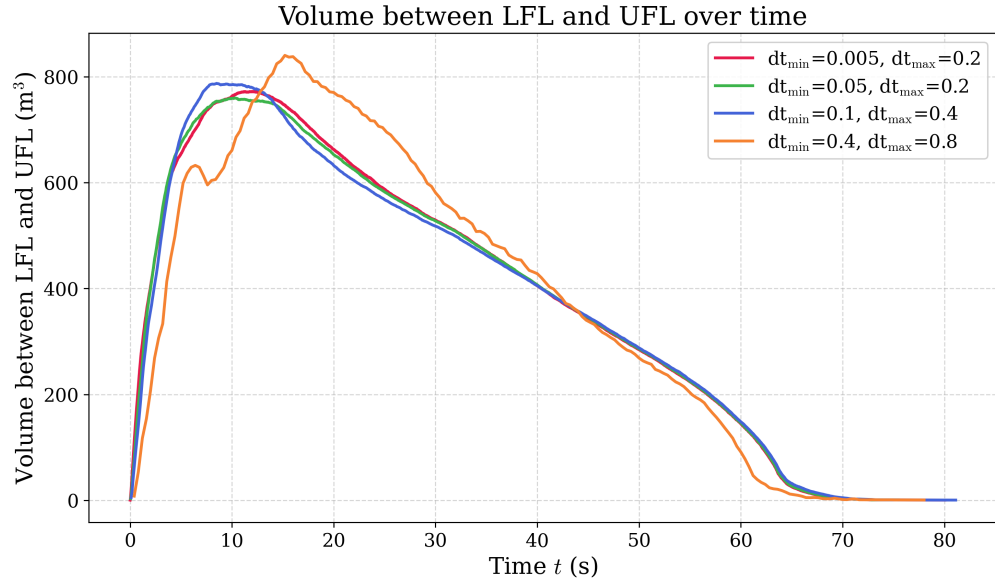


Figure 3.4: Volume between LFL and UFL over time for the different time step configurations.

3.3.3 Height

As described previously, the use of a three-stage compressor naturally implies that different pressure levels are going to be located in distinct areas inside the compressor enclosure (see figure 3.5). The designated leak points are detailed in table 3.2. To choose the height for the low and high pressure areas, a sensitivity study was done (see figure 3.6). All of the cases were run in the $-\hat{j}$ direction and at 900bar. In the case of the intermediate pressure, the chosen height is at the top of the compressor. It would not be realistic to consider the leak below this point because that space is occupied by the compressor itself, and it also does not seem reasonable to have a leak higher up because the leak is coming from the compressor by definition of the current problem. This is why it was not included in the sensitivity analysis.

Table 3.2: Leak locations.

Region	Pressure	Coordinates
Low pressure	430bar	(-2.50, -3.75, 1.20)
Intermediate pressure	630bar	(-2.50, -1.70, 1.00)
High pressure	900bar	(-2.50, 0.40, 0.70)

3.3.4 Direction

In order to optimise the number of simulations to be done, a sensitivity analysis focusing on the effect of the direction was done for each of the leak locations.

The studies for locations (-2.5, 0.4, 0.7) and (-2.5, -3.75, 0.7), which refer to the high pressure and low pressure leaks, respectively, were both done at 900bar. This is because they were carried out in the beginning of the project, and at that point, the study did not take into account particular points for each of the pressure levels. However, a reorganisation of the simulations was done at a later date, and the different pressure zones were defined. At that moment, therefore, the sensitivity study for the third point, (-2.5, -1.7, 1.0), was defined as the location of the intermediate pressure leak, and in this case the analysis was done using a full bore leak at the intermediate pressure. This latter analysis does not consider the $-Z$ direction because the leak point is defined to be on top of the compressor, so the $-Z$ direction would imply that the leak is going into the compressor: that does not make physical sense and was therefore excluded. Whilst this reasoning can seem somewhat confusing, had other values been used, the results should be proportional: one would expect that the most safety-concerning directions be the same in all the cases, the geometry being the driving factor: this is the reason why it wasn't repeated. The results of the analysis are shown in figure 3.7.

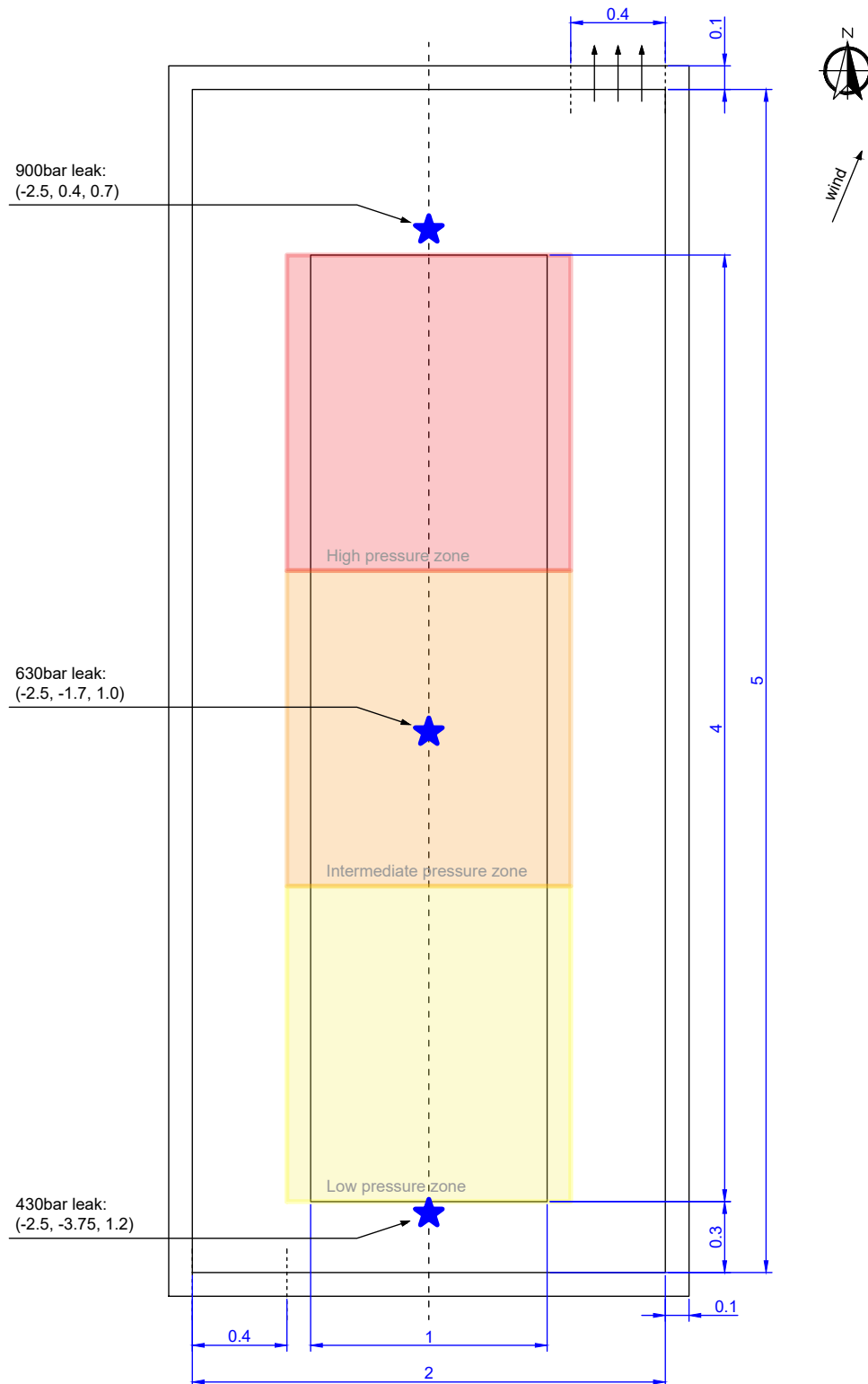


Figure 3.5: Floorplan of the enclosure, pressure zones and leak positions.

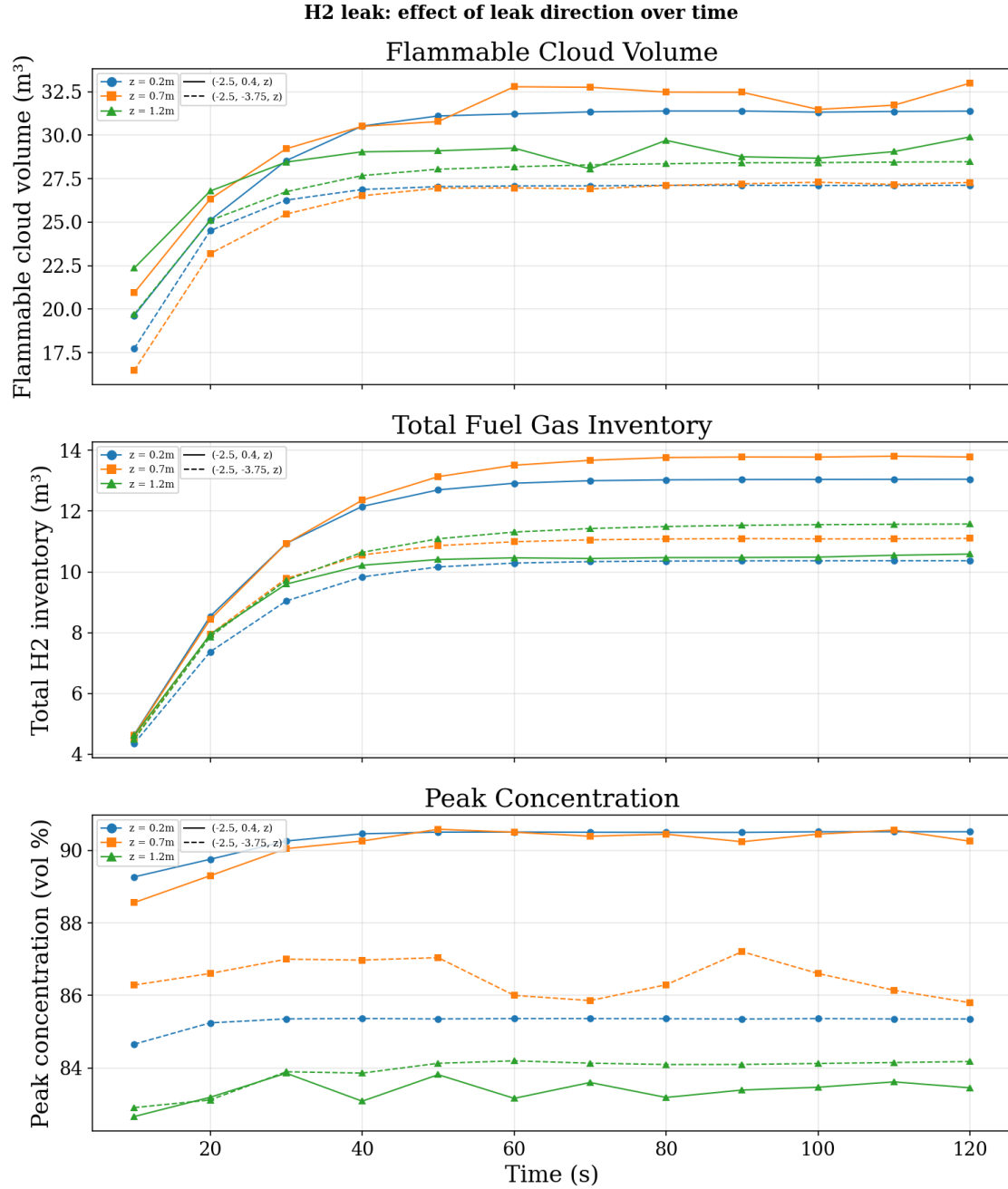


Figure 3.6: Sensitivity study of the different heights for each of the locations.

Based on the sensitivity analysis, the directions chosen for the study are:

$$430\text{bar} \begin{cases} -Y \\ +Z \\ -Z \end{cases} \quad 630\text{bar} \begin{cases} +X \\ -X \\ -Y \end{cases} \quad 900\text{bar} \begin{cases} -X \\ +Y \\ -Y \end{cases}$$

3.4 Unignited leaks

3.4.1 Exposure to flammable gas

The first question evaluates the size of the cloud that can ignite based, firstly, on its projection on the XY plane, and secondly, on its flammable volume.

For the calculation of the area, the script moves through the domain and for each (X,Y) pair referring to a cell, checks if there is any cell in all Z that has a hydrogen concentration that is in the flammability range. If it finds it, it accounts for its XY area —not all cells have the same area— and continues to the next pair. This is done like this so the result is the actual projection, and it's not oversized. The script also checks which one of these cells is the furthest one from the leak source to give an idea of how far the cloud extends from the leak point. All of the projections are calculated when the steady state has been reached. Regarding the flammable volume, the raw results are plotted.

The plots are organised by pressure and hole size. Under this filtering six cases emerge, three from natural ventilation and three from forced ventilation. In the case of the area, these six cases are presented as a ranking, from most to least dangerous, using floorplans. In this case, the danger of the scenario is defined as proportional to the area of the cloud, so the larger the area the more dangerous the scenario. The flammable volume is a plot with the same cases.

Some of the results can be found in this section (figures 3.8, 3.9 for area, 3.10 and 3.11 for volume) and the rest in the appendix (figures A.1 to A.25). Figure 3.12 shows the values of the area and maximum reach, whilst figure 3.13 focuses on the volume. Both show all the scenarios and refer to the steady state. They are organised following a hierarchy of hole size, pressure, and direction: each subplot refers to a hole size, and the different cases are organised side by side with small brackets that delimit the different pressure levels. Inside each pressure, the relevant directions are shown. A similar figure to these but with a different hierarchy, with pressure at the top followed by the hole size, can be found in the appendix, figures A.24 and A.25.

An important observation is that whilst the volume of the cloud increases as the leak flow increases, this is not necessarily true for the projected area, as it depends on the shape of the cloud. It's also worth keeping in mind that the conclusions explained below aren't necessarily general. This is because in the study the position of the leak is tied to its pressure and each position is therefore surrounded by a slightly different geometry, which has a certain impact

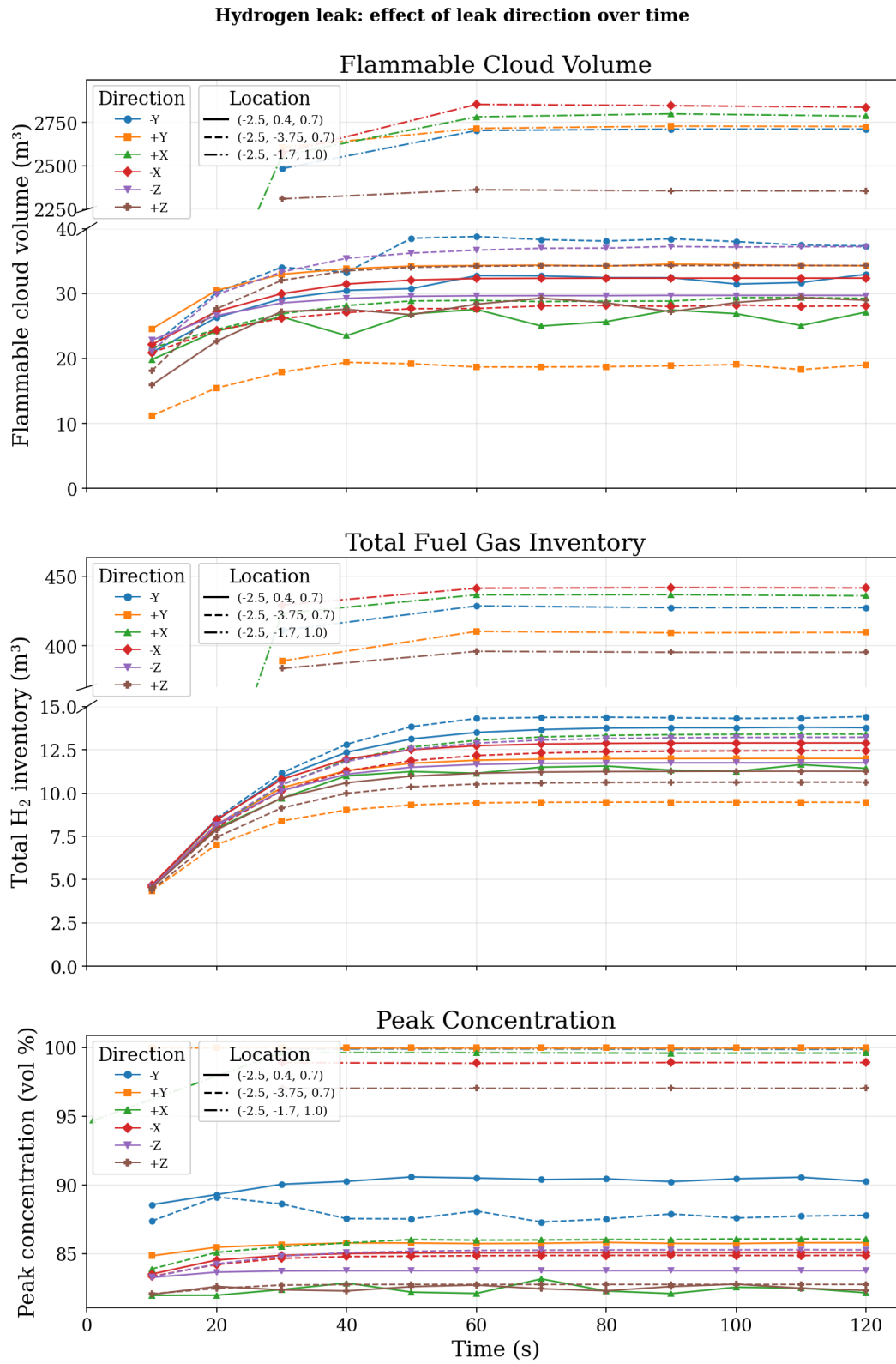


Figure 3.7: Sensitivity study of the different directions for each of the locations.

on the dissipation of the cloud. In fact, the intermediate pressure cases seem to be the least affected by the type of ventilation, but this is most likely due to geometry, as the intermediate pressure location was set in the middle of the container.

Flammable area and maximum distance

In the 430bar case, the results show that the worst case is always in the $-Z$ direction, but there is no clear advantage between natural and forced ventilation (see figure 3.8). However, the $+Z$ direction combined with natural ventilation is consistently the least dangerous of the scenarios considered. In the 630bar case, there is no clear direction that stands out as the most hazardous: in the 1% leak, the worst case is $-Y$ with natural ventilation; in the 10% leak, $+X$ with forced ventilation; in full bore, $-X$ with forced ventilation. In both the 10% and full bore cases, the least dangerous scenario is the $-Y$ direction with natural ventilation. In the 900bar case, for the 1% and 10% cases, $-X$ is the worst direction, but the type of ventilation is not consistent. In the full bore scenario, the worst case is $-Y$ with natural ventilation. In general, the projected area increases over time until it reaches a steady state, approximately after 60s (see figure 3.9 as an example). The maximum distance doesn't appear to be a strong indicator of cloud size.

When comparing all the cases together (figure 3.12), the largest variation is observed with the 1% area hole size, especially in high pressure, but it depends on the direction. Only at 430bar is the maximum distance consistently higher for the forced ventilation cases. At 10% area, all the results look fairly similar between them. The most notable characteristic is a slight reduction of the parameters when looking at the 900bar cases. In the full bore scenarios, the area is consistently larger in the 900bar cases when using natural ventilation, but the maximum distances are similar. There's not much variation in the intermediate pressure cases and there is more variation in the low pressure cases, but it doesn't appear to follow a pattern.

Flammable volume

The second part of the question looks at the size of the cloud in terms of volume. The volume taken into account here only accounts for the fraction of the cloud which has a concentration between LFL and UFL. This typically increases during the transient, and then stabilises. The time to reach steady state is in most cases less than 60s. The results show a tendency of the forced ventilation being beneficial if the leak is small, but disadvantageous when the leak is larger, because at that point the forced ventilation is insufficient—it was not designed for large leaks—the gas rapidly accumulates inside the container, and the smaller openings become a disadvantage. In these cases, the pressure building inside the enclosure is the main factor that pushes the leaked gas outside. In some cases, the direction has a big impact on the size of the cloud. In the 430bar, 1.2mm leak case (1% leak, figure 3.10) the difference in cloud volume between the worst case and the best case is more than twofold: $\sim 26\text{m}^3$ ($-Z$ direction, natural ventilation) vs. $\sim 10\text{m}^3$ ($+Z$ direction, natural ventilation). The 430bar scenarios appear to be the most affected by the direction. In the 630bar case, the difference in volumes tends to be rather small. The observed general tendency is that as the leak flow

increases, the volume is less affected by the direction. There is one particular direction in one case (900bar, 1.0mm hole, figure 3.11) which exhibits an abnormal behaviour: the $-X$ direction seems to stabilise but then dips and stabilises again. This is because a significant fraction of the cloud passes the UFL threshold. As stated, this behaviour is not observed in any other cases in this study.

When comparing all pressures side by side (figure 3.13), in the 1% area cases, the volume is smaller in all the cases when using forced ventilation except for the $+Z$ direction in low pressure. The lowest volume is found in $-X$ in high pressure as this is the scenario where a large portion of the gas overcomes the UFL threshold. At 10% area, there is a lot of variability in low pressure. Intermediate pressure doesn't exhibit a lot of variation, neither does high pressure but its values are slightly lower. In full bore, the values in low and intermediate pressure are approximately the same, except in the $-Z$ direction, low pressure and forced ventilation, which is the highest of all. Natural ventilation is slightly lower in almost all of these cases. In high pressure, the values are overall lower, but they don't vary a lot and natural ventilation is higher in this case.

Flammable cloud footprint | Natural ventilation vs Forced ventilation | 430bar, d=3.8 mm | global ranking by area

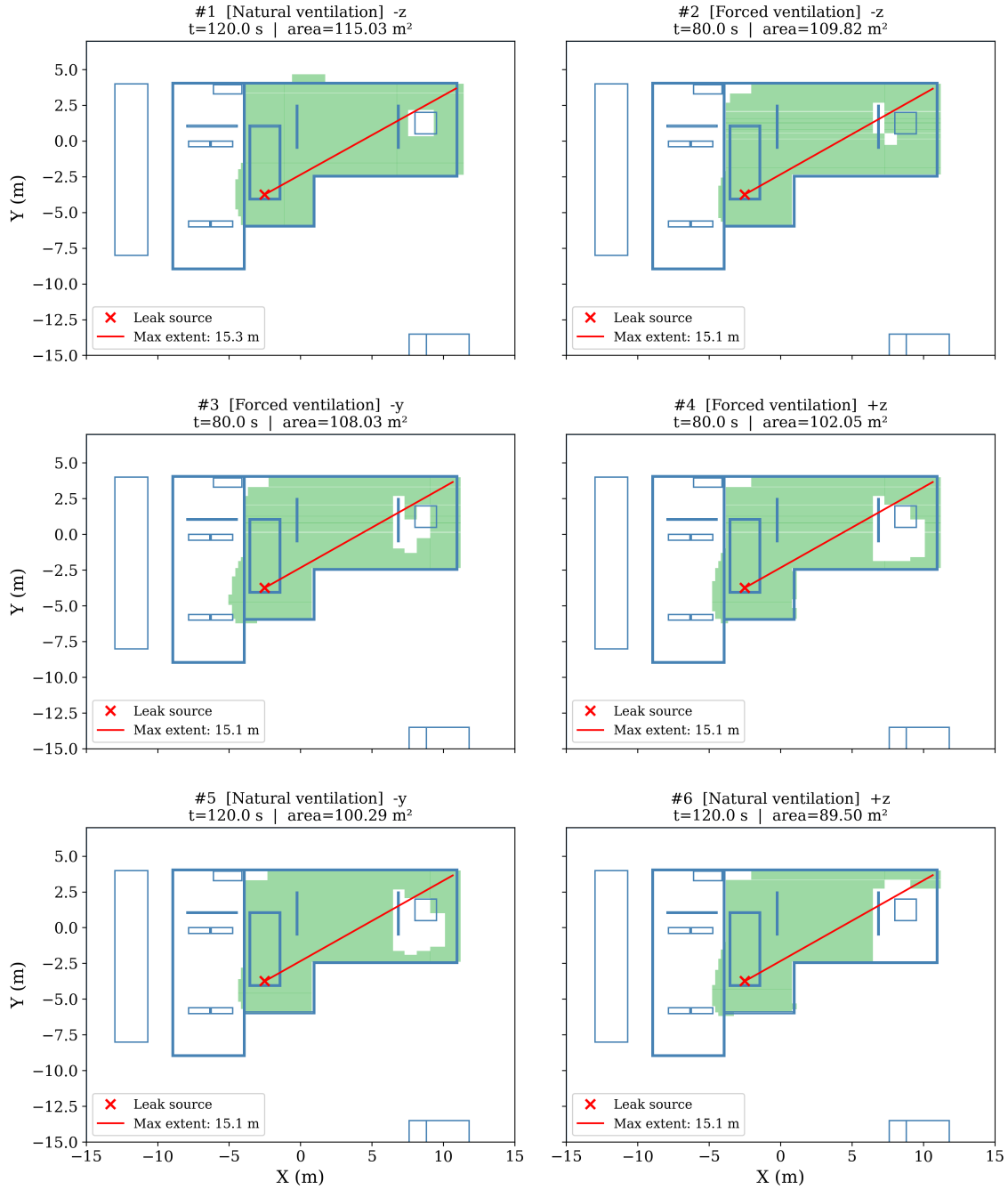


Figure 3.8: Flammable area projection at steady state for 430bar, 10% area cases.

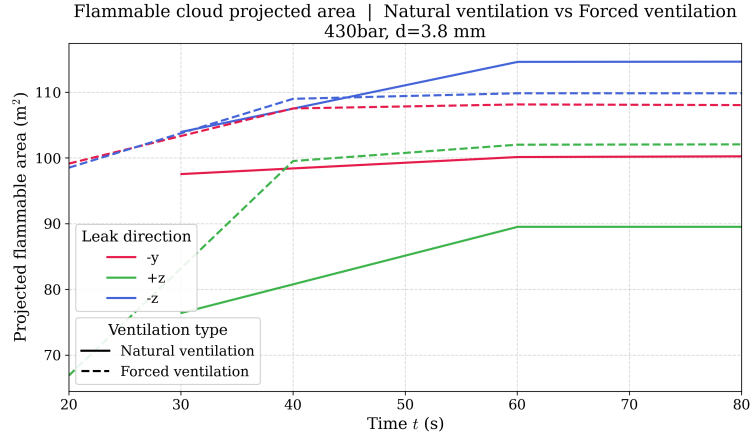


Figure 3.9: Flammable area over time for 430bar, 10% area cases.

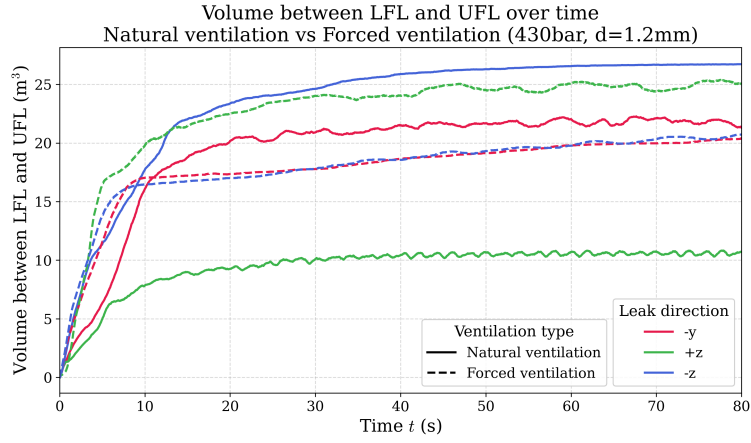


Figure 3.10: Volume between LFL and UFL over time for 430bar, 1% area cases.

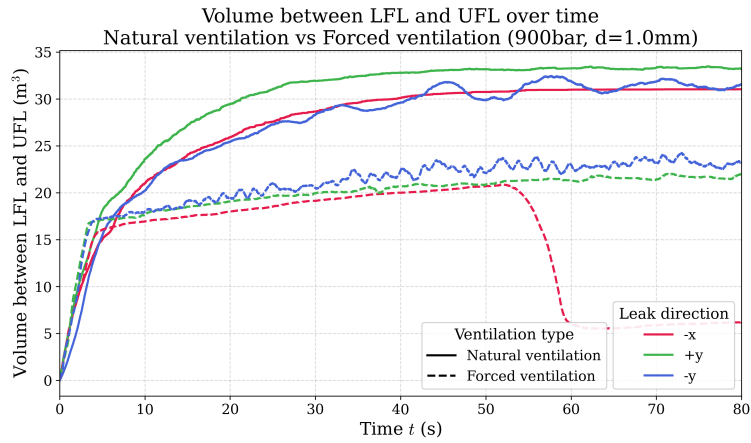


Figure 3.11: Volume between LFL and UFL over time for 900bar, 1% area cases.



Figure 3.12: Summary of the cloud projection of the flammable area and maximum reach at steady state for all the cases organised by a hierarchy of hole size, then pressure, then direction.

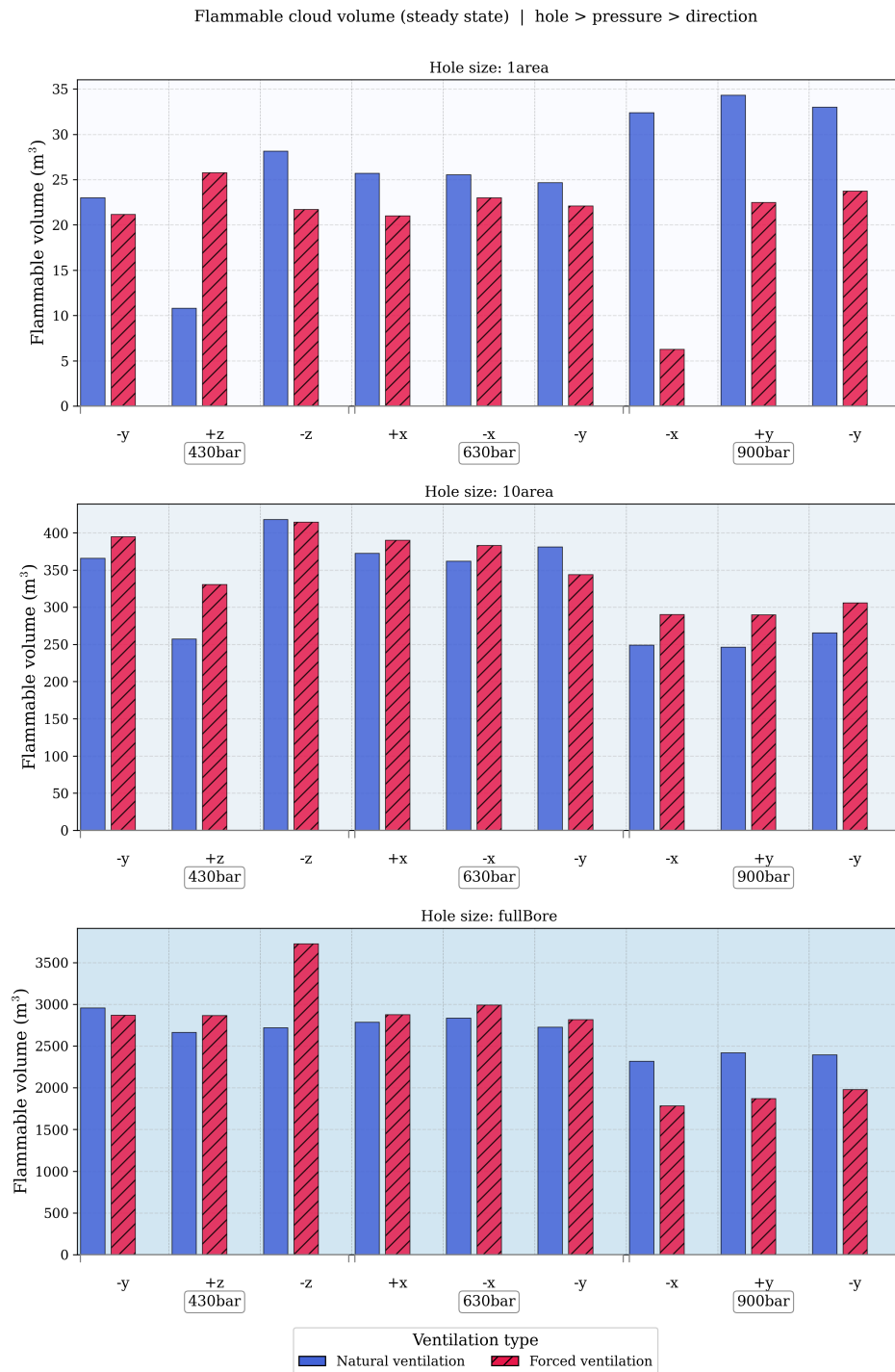


Figure 3.13: Summary of the flammable volume of the cloud at steady state for all the cases organised by a hierarchy of hole size, then pressure, then direction.

3.5 Ignited leaks

3.5.1 Dynamic pressure

The dynamic pressure q [35] is a quantity used in fluid dynamics which can be thought of as the kinetic energy of the fluid per unit volume. It's derived based on the conservation of linear momentum for a fluid and is defined by

$$q = \frac{\rho u^2}{2} \quad (3.1)$$

where ρ is the density of the gas, and u its velocity. The density is calculated assuming that the gas follows the ideal gas law, and using the temperature, concentration and velocity fields calculated by KFX. The ideal gas assumption is a bit lacking in the very near field as the pressure there is very high, but should be quite truthful a bit further, when the gas has expanded to approximately atmospheric pressure. Inside the container, therefore, these results are less reliable. Because of this, the results produced by KFX were inspected and an analytical model for the pressure peaking phenomenon was applied in some cases.

Because of the jet always being impinged and enclosed, the dynamic pressure outside the container, and especially outside the firewalls, is very low, and is not expected to have enough energy to produce any damage in those areas (see figure 3.14), nor being able to pick up stones or similar. It is also important to take into account that the values calculated in this question are on the conservative side because for the calculation of the velocity the module is used, and in practice only a fraction of the directions could potentially produce damage.

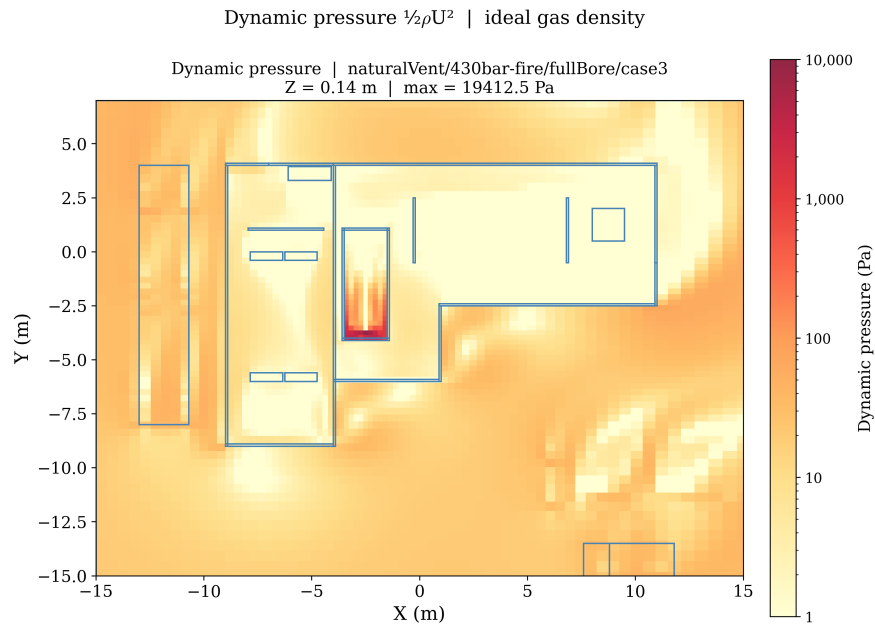


Figure 3.14: Dynamic pressure in steady state for 430bar, full bore, $-Z$, one of the most dangerous scenarios.

Pressure peaking phenomenon

The results obtained in this question show that the dynamic pressure outside the area protected by the passive barriers is very low. However, another important finding is that the pressure inside the compressor enclosure is very high, and there could be instances of an overpressure due to the pressure peaking phenomenon, which could pose serious consequences.

Following the criterion used in [21] where the model to be used was developed, the first step for evaluating the potential for an overpressure caused by pressure peaking phenomenon is to evaluate which of the cases would eventually get to complete displacement of air inside the enclosure. All of the cases were checked, and it was found that for at least some direction of the full bore and large leak scenarios, all the pressure levels would lead to more than 99.9% peak concentration in volume. The results were generally worse for the forced ventilation scenarios.

In particular, of a total of 54 cases, the peak concentration is $>99\%$ in 22 cases (41%), $>99.9\%$ in 15 cases (28%), 100% in 7 cases (13%, full bore in all pressures, in one or more of the directions). These cases are then inspected one by one to check that the inside of the container is homogenous. The conclusion is that the model should be applied to the full bore cases in all pressure levels, because the PPP model doesn't take into account the direction of the leak. The result can be observed in figure 3.15. Taking into account the thresholds in table 3.3, which establishes the expected structural damage due to an explosion as a function of the peak overpressure, the conclusion is that such a scenario, regardless of the pressure at which it happened, would lead to the total destruction of the structure, with possible additional damage to people, assets or the environment in the vicinity because of the pieces that would get expelled as a result of the overpressure. The degree to which the firewalls would be affected is difficult to determine and out of the scope of this project.

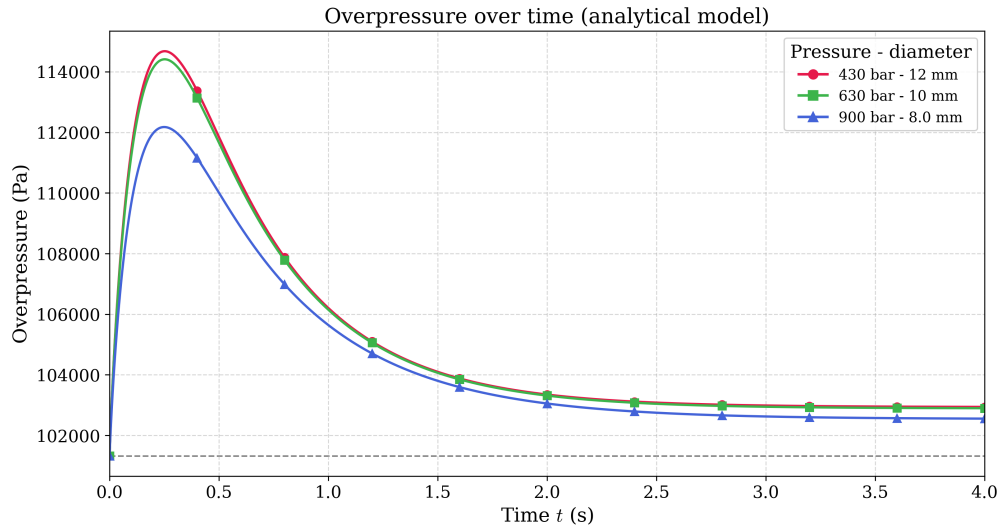


Figure 3.15: Results of the pressure peaking phenomenon model for the full bore scenarios.

Table 3.3: Expected structural damage caused by an explosion depending on peak overpressure, from [36]

Damage	Definition	Threshold
TOTAL DESTRUCTION	Integrity of the structural parties completely compromised.	70kPa
VERY SERIOUS DAMAGE	Collapse of at least 50% of the structure and serious damage to the rest of the building.	35kPa
SERIOUS DAMAGE	Partial demolition of the building but with the possibility of rebuilding the damaged parts.	7-15kPa
MODERATE DAMAGE	Possibility of totally rebuilding the building.	3kPa
NEGLIGIBLE DAMAGE	Broken windows, partial damage to the roof, cracks in the walls, etc.	1-1.5kPa

3.5.2 Radiation

The radiation question evaluates the results of the fire simulations based on the radiation level, and ranks them according to a severity score.

The radiation is evaluated at a height of 1.5m, because it's roughly the height at which a random adult human is likely to have their torso/head. It checks all the cells in the nearest Z plane, and classifies them in five different categories depending on their radiation level. The area of each cell is taken into account for the calculation of the area, because the grid is structured but not uniform. This information is used to calculate a severity score, which uses different weights for each of the categories:

$$\text{Severity score} = \sum_i w_i \times A_i \quad (3.2)$$

where A_i is the area in m^2 where radiation exceeds threshold i , and w_i is its associated weight. Since the weights are dimensionless, the units of the score are technically m^2 . However, the score doesn't represent a physical area, but a severity-weighted exposed area. The thresholds and weights are summarised in table 3.4.

When choosing the weights, it was observed that purely linear weights disregarded high-consequence areas to a certain degree, whilst geometric weights gave too much importance to a small but highly dangerous area, and would give too little weight to a much larger area whose severity was lower but could produce a lot of harm in practice. Therefore, the chosen weights are a deliberate compromise between linear and geometric scaling, and are chosen so they reflect that a higher radiation poses a higher danger level, but don't completely overshadow a technically lower severity area that could, in practice, produce much more harm. Another important detail is that the area inside the compressor enclosure is excluded from

Table 3.4: Radiation thresholds used in the severity score, from [37, 38].

Threshold [W/m ²]	Weight	Description
1 500	1	No damage on long exposure
5 000	3	Second-degree burns if exposure >30s
17 000	8	FAA aircraft escape route
37 500	20	Immediate fatality

the calculation because regular human presence is not expected there.

The results —figures 3.16 and 3.17, the rest (figures A.26 to A.36) can be found in the appendix— show that the affected area is more strongly correlated with the hole size than the pressure. That is, for the medium leak, the results are similar for all the pressure levels, and the same happens for the large leak and full bore scenarios. For each of the leak levels, the severity appears to be proportional to the flow but tends to also be inversely proportional to the pressure. In more than half of the cases, we observe that the fire starts outside of the container. [Here](#) or on <https://youtu.be/dQgMN8o8eqo> one can view one of such cases. The same case seen from a slice of the YZ plane can be found [here](#) or on <https://youtu.be/fWUio4ihvuk>.

The compressor enclosure is good at containing the fire only when the leak is small enough (medium leaks in this study) and the fire has started inside of the enclosure. In the large leaks, we observe that if the fire started inside the container, the area outside of the firewalls is mostly safe at the studied height. In the case of fires that started outside of the container, the area outside of the firewalls is more or less affected depending on the case, but evacuation should nevertheless be possible. In the full bore scenarios, the firewalls fail to contain the fire and its consequences would be devastating in the nearby area. It can also be concluded that the affected area is heavily affected by the wind: this happens because the gas cloud is pushed by it. The reader can watch a video of one of the worst scenarios, 430bar and full bore, [here](#) or on <https://youtu.be/b9JjDklfc-Y>.

The forced ventilation scenarios tend to dominate on severity when looking at full bore leaks, whilst for medium and large leaks, the natural ventilation scenarios tend to be on top. The observed most and least severe directions for each pressure level are detailed in table 3.5. There’s no clear trend, which gives weight to the hypothesis of the geometry being the main driving factor.

Figure 3.17 shows the actual areas instead of a score. It is organised in a hole size, then pressure, then direction hierarchy. In the full bore cases, the forced ventilation scenarios clearly dominate regarding affected area, especially in the intermediate pressure, and the areas under the higher thresholds also tend to be higher. In the 1% area case, the results are varied. Regarding total extension none of the ventilation strategies appear to be clearly superior, but we can appreciate a trend of the forced ventilation scenarios having lower

radiation levels, which could make these scenarios less dangerous. In the 10% area cases, there is a lot of variation and no clear trends are observed.

Table 3.5: Most and least severe observed directions for the different pressure levels and types of ventilation.

Pressure	Ventilation	Most severe	Least severe
430bar	Natural	+Z	-Z
	Forced	-Z	+Z
630bar	Natural	-X	unclear
	Forced	-X	+X
900bar	Natural	unclear	unclear
	Forced	-Y	-X

Radiation comparison | Natural ventilation vs Forced ventilation | 430bar, 3.8 mm | global ranking

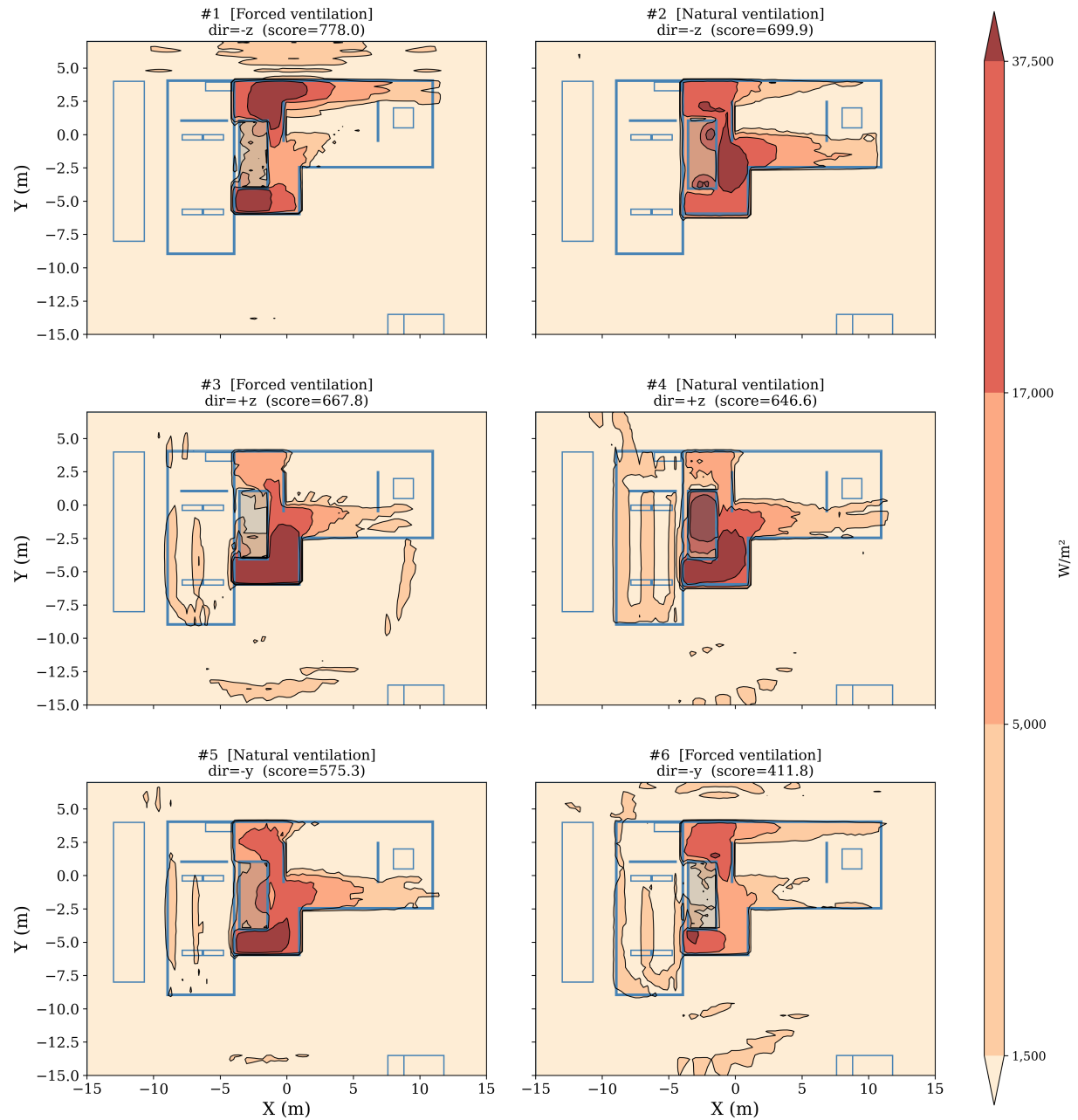


Figure 3.16: Radiation in steady state for 430bar, 10% area cases. The score has units of m^2 , but doesn't directly represent physical area, but a severity-weighted exposed area.

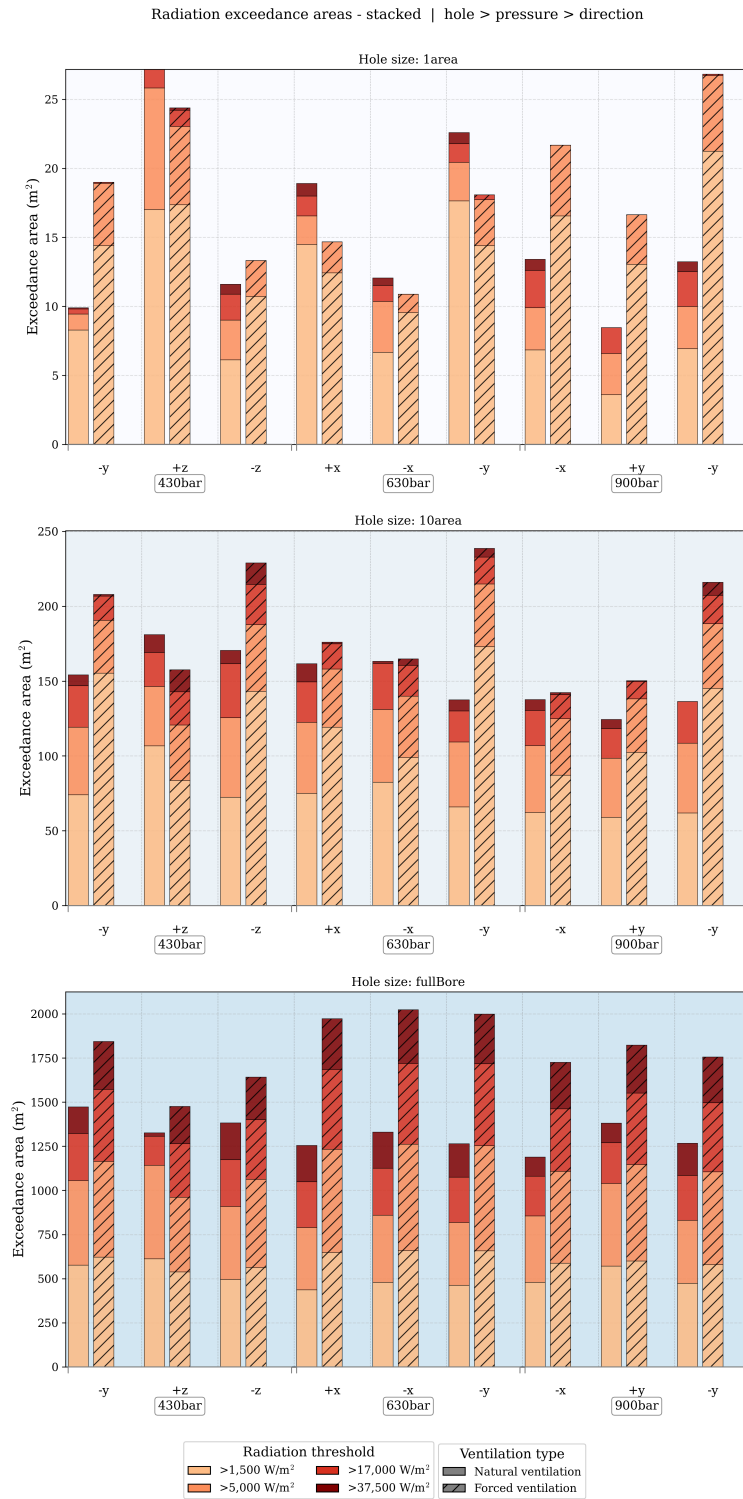


Figure 3.17: Summary of the radiation scores at steady state for all the cases organised by a hierarchy of hole size, then pressure, then direction.

3.6 Safety barriers: effect of the firewall

The height of the firewall was evaluated using the metrics developed in the other questions. Various cases were studied, from 1m-high walls to 8m-high with 1m increases. The original design featured 4m-high walls.

The extension of the cloud in the conditions of the simulations doesn't appear to grow linearly as the wall height increases (figure 3.19). The radiation consequences are a lot worse when the walls are low, since they fail to contain the fire. Their effect as barriers improve when its height is increased, but not linearly (figure 3.20). This is confirmed in figure 3.18, which shows that the area affected by radiation decreases quite linearly in the 2 to 5m region. Increasing the height after this point doesn't provide as much of a significant advantage. Regarding the flammable volume, the results show that it increases as the wall height increases, since more containment also makes it more difficult for the gas to disperse effectively. A relatively flat zone is observed in the 4-6m interval, with the 5m volume actually being slightly lower than the other two, but this might be due to numerical turbulence. In any case, in light of this results, 5m appears to be the optimal wall height, which—for visualisation purposes—is slightly less than the average height of a two-storey building.

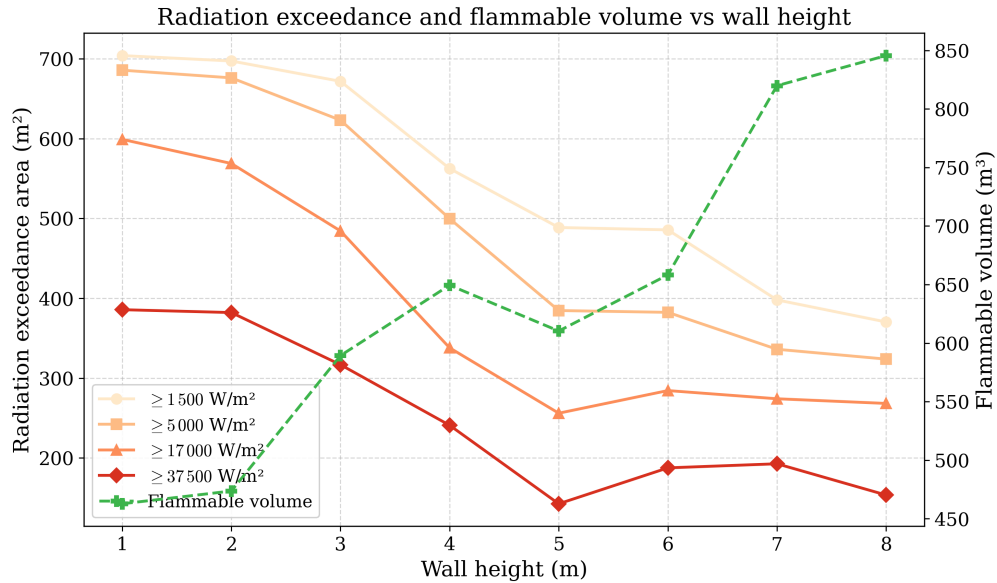


Figure 3.18: Cumulative areas affected by radiation, and volume of the flammable cloud.

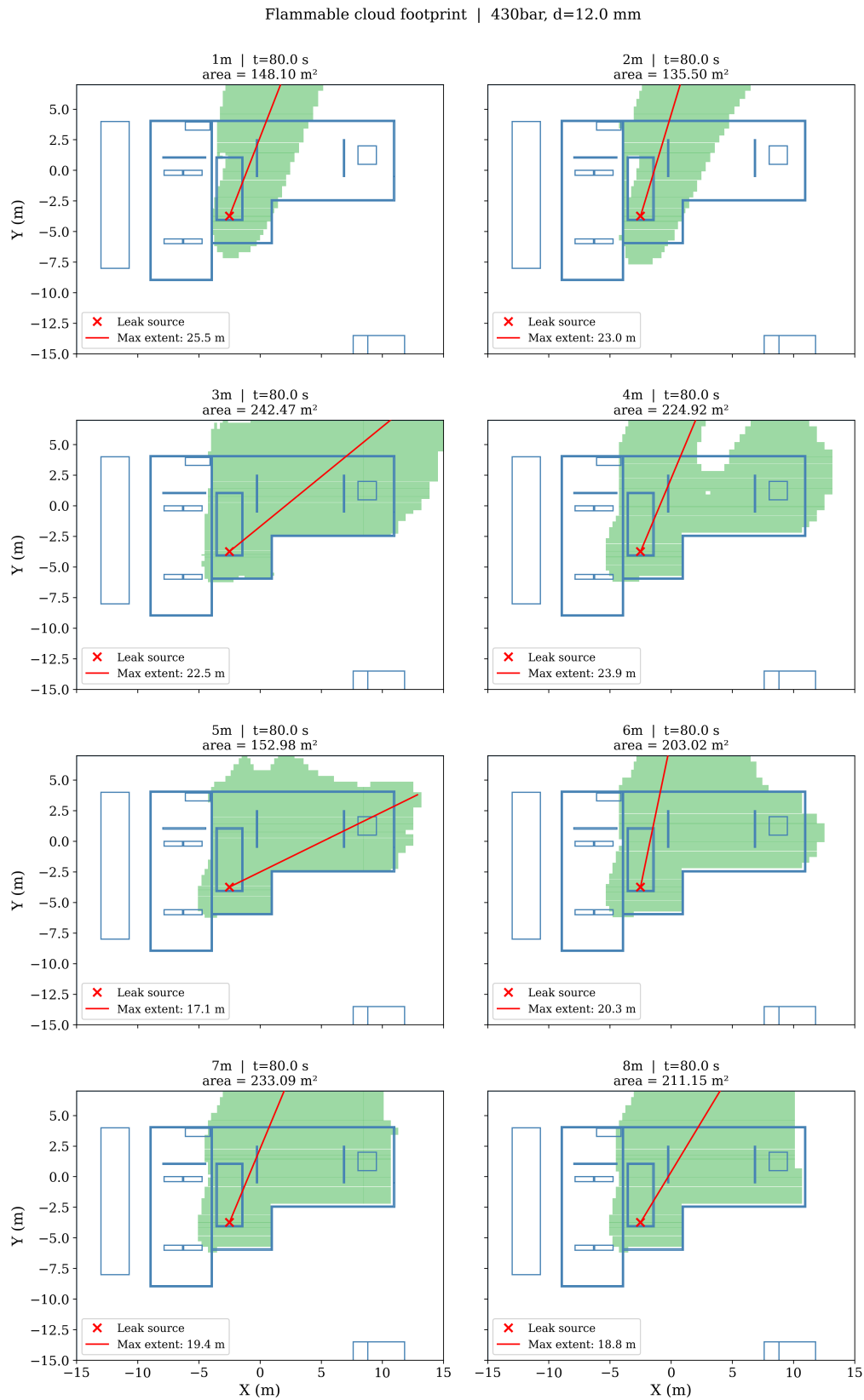


Figure 3.19: Flammable area projection for each of the wall heights, applied to the worst case.

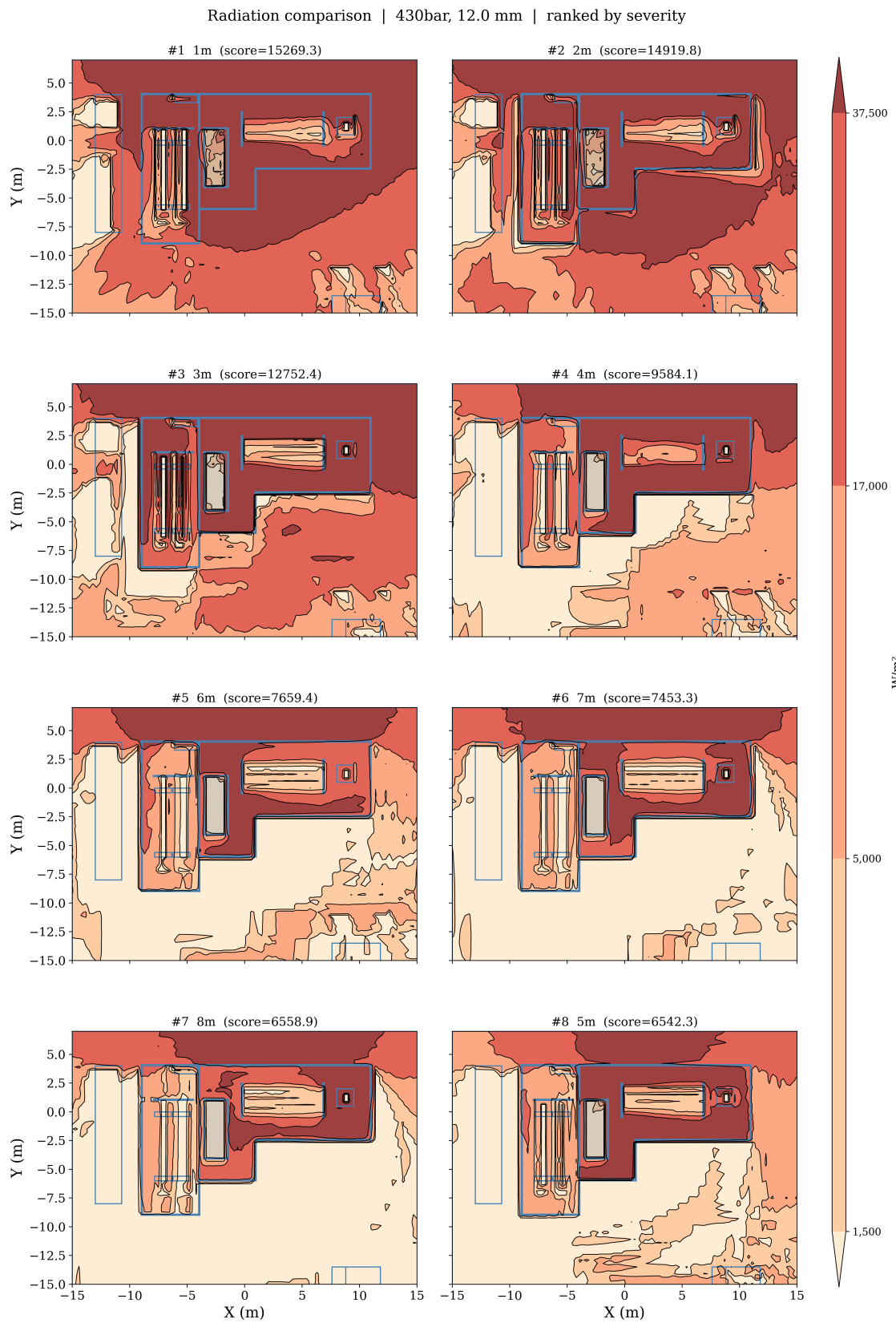


Figure 3.20: Radiation for each of the wall heights, applied to the worst case.

Chapter 4

Discussion

The results produced in the different parts of the analysis are gathered here in order to extract some conclusions. This analysis is made up of several parts: (1) overall severity rankings, (2) main effects analysis, and (3) scatter plot.

4.1 Overall severity rankings

For each metric in each case, a normalised score is calculated. Then, these are averaged in order to get an overall impression of the severity of the case, and be able to rank them. A detailed heatmap of the results can be found in the appendix, figure [A.37](#). Figure [4.1](#) shows a truncated version of figure [A.38](#), in which the cases are ranked according to their overall score. It is clear that there is a huge difference in the impact of the full bore cases in comparison of the rest of the cases. This figure also corroborates that the forced ventilation scenarios tend to be the most dangerous, as they, without a doubt, dominate the ranking and their severity is deemed a lot higher.

4.2 Main effects analysis

The present main effects analysis follows the approach of a one-way ANOVA analysis, but the number of cases is too low to be statistically significant. It can be thought of as a kind of sensitivity analysis. In the plots, the dots show the values of the individual cases at steady state, and the bar shows the average of the group. Therefore, if the dots are spread, the variation within the group is high. If they are close together, it is low. The height of the bar gives information on the variability between groups. Based on the visual combinations one can get from this, four extreme cases can be defined:

1. **Large bar difference, tight points:** little variation within the groups but big difference when we modify this parameter. Strong evidence that this parameter affects the outcome.
2. **Large bar difference, spread out points:** there is a certain variation when we change the parameter, but since the variation within the groups is also very high, the parameter may or may not have an important effect. Other interactions might be

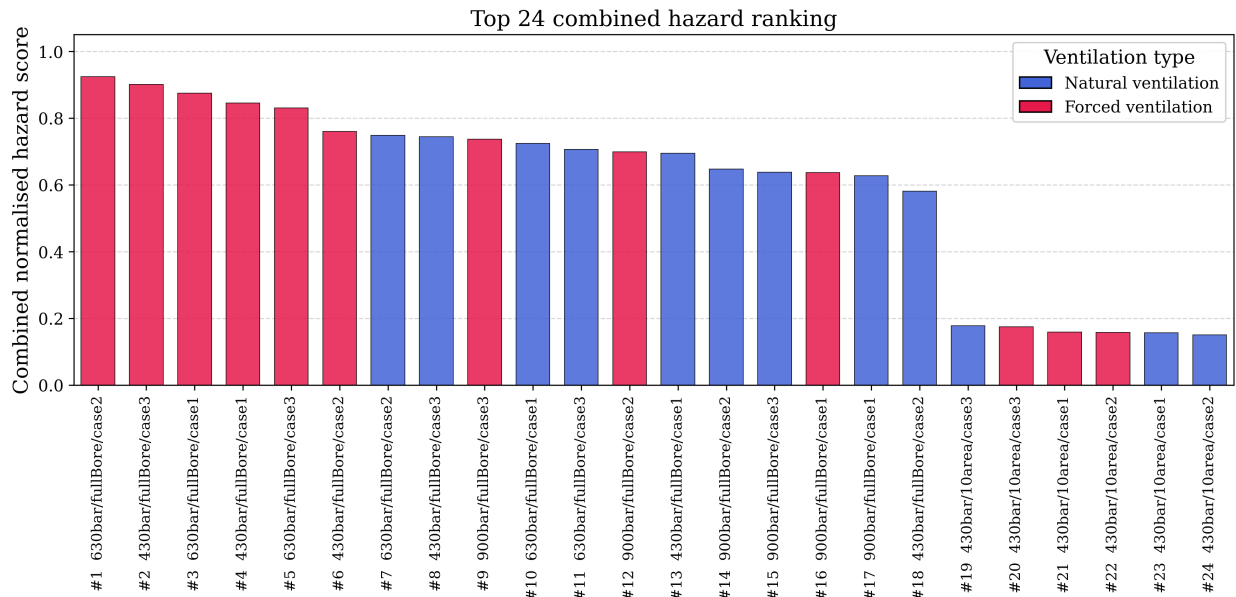


Figure 4.1: Raking on the cases based on their combined score. The ranking is truncated to 24 positions to improve visualisation.

happening.

3. **Small bar difference, tight points:** most likely, the parameter doesn't affect the outcome much.
4. **Small bar difference, spread out points:** there is a lot of variability, so conclusions can't be drawn.

4.2.1 Flammable area

Figure 4.2. No conclusions can be drawn from changing the ventilation type. The pressure may have a small effect, but it's not evident enough. The hole size does appear to have a big importance on the calculated flammable area: there is a big difference in the bar height, especially in the full bore case. The points are close together in the 1% and 10% area cases, but quite spread out in the full bore case.

4.2.2 Flammable volume

Figure 4.3. The conclusions are the same as in the flammable area.

4.2.3 Dynamic pressure

Figure 4.4. The cases in ventilation type and pressure are less spread out than in the flammable area and volume cases, but the bar height is roughly the same. The difference is more notable in the hole size case, but not clear.

4.2.4 Radiation

Figure 4.5. More variation when changing the ventilation type than in the other cases, but the difference is small and the variability within groups high, so it's inconclusive. No conclusions can be drawn for the pressure either. As for the hole size, there is a big variation between groups, especially in the full bore case, but the variation within the group is also greater than in the smaller leaks.

4.2.5 General conclusions

From this analysis, the only variable that appears to clearly be affecting the outcome is the hole size. This conclusion sides with the previous section, where the full bore scenarios were ranked the most severe. However, the forced ventilation conclusion cannot be backed by the main effects analysis.

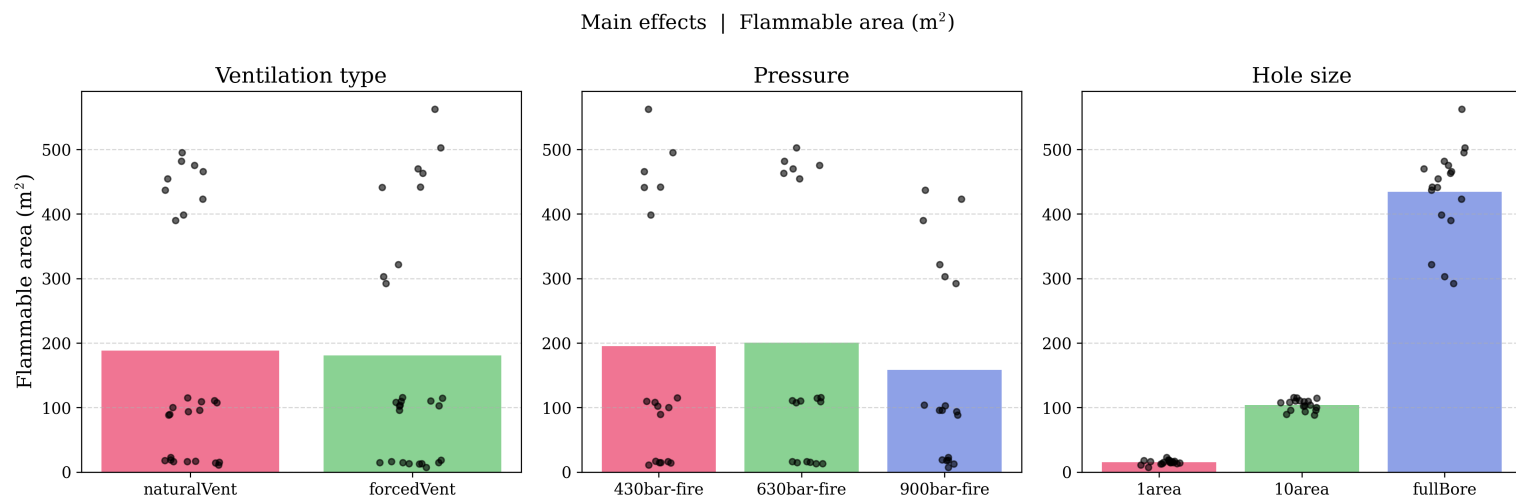


Figure 4.2: Main effects of the different variables on the flammable area.

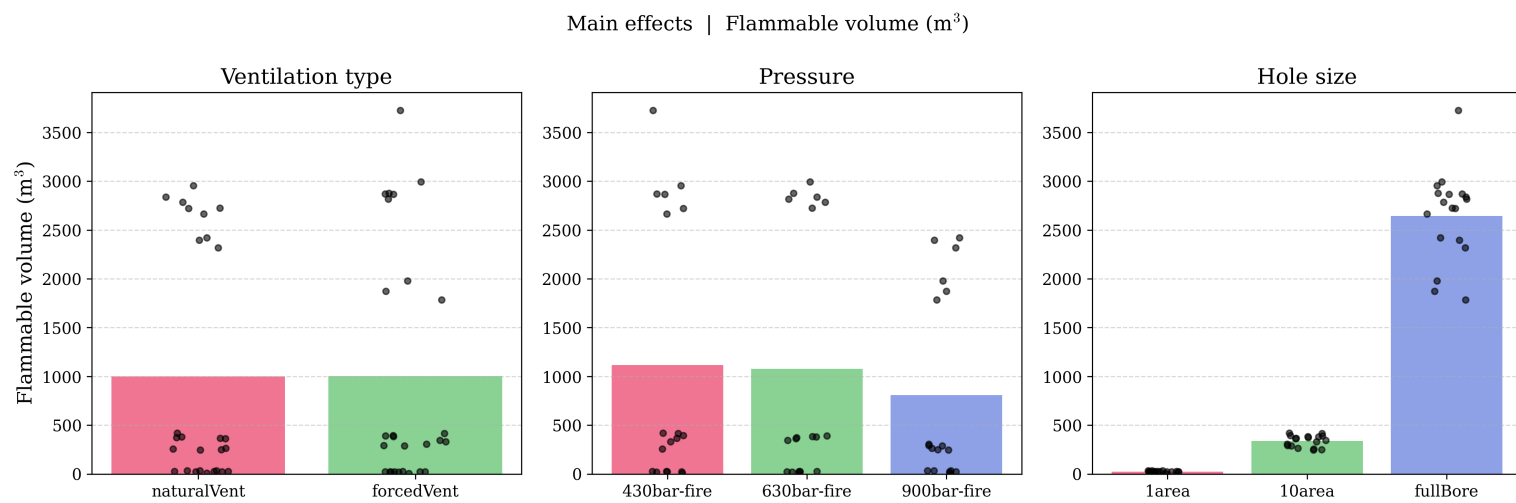


Figure 4.3: Main effects of the different variables on the flammable volume.

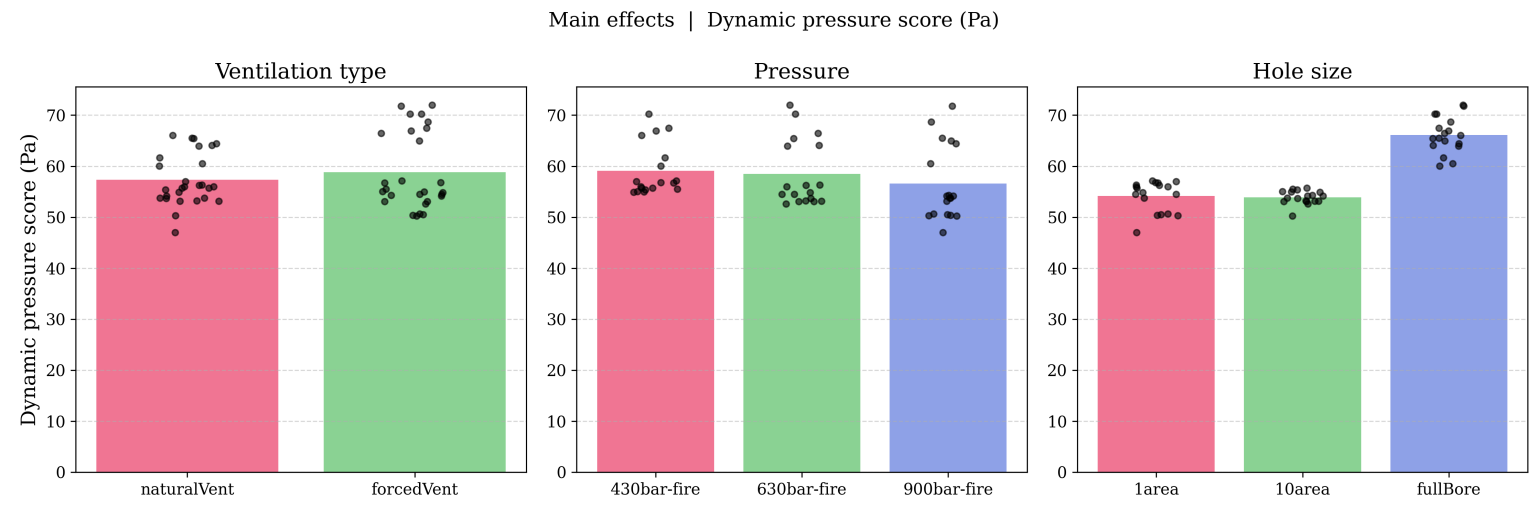


Figure 4.4: Main effects of the different variables on the dynamic pressure.

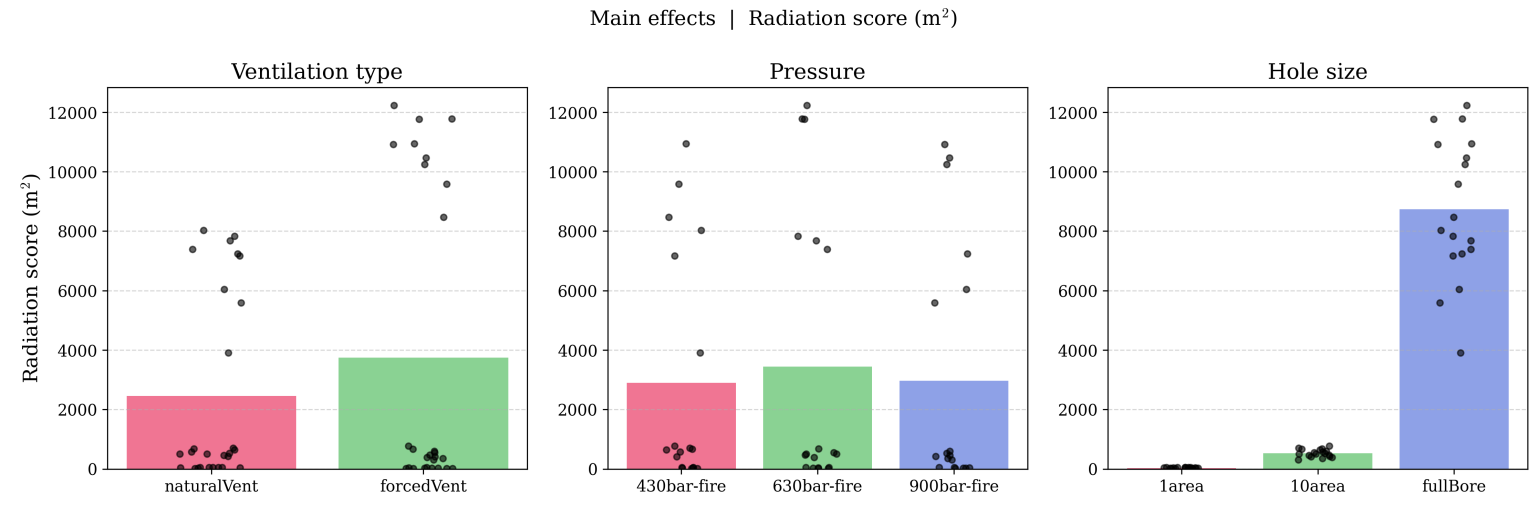


Figure 4.5: Main effects of the different variables on the radiation score.

4.3 Scatter plot

The values for flammable area and volume, radiation score and dynamic pressure score are plotted against each other in figure 4.6. There's no clear correlations between the metrics, other than flammable area and volume, which is to be expected since the calculated area is the projection in the XY plane of the cloud volume. There's nevertheless some variation due to the cloud being more or less tall.

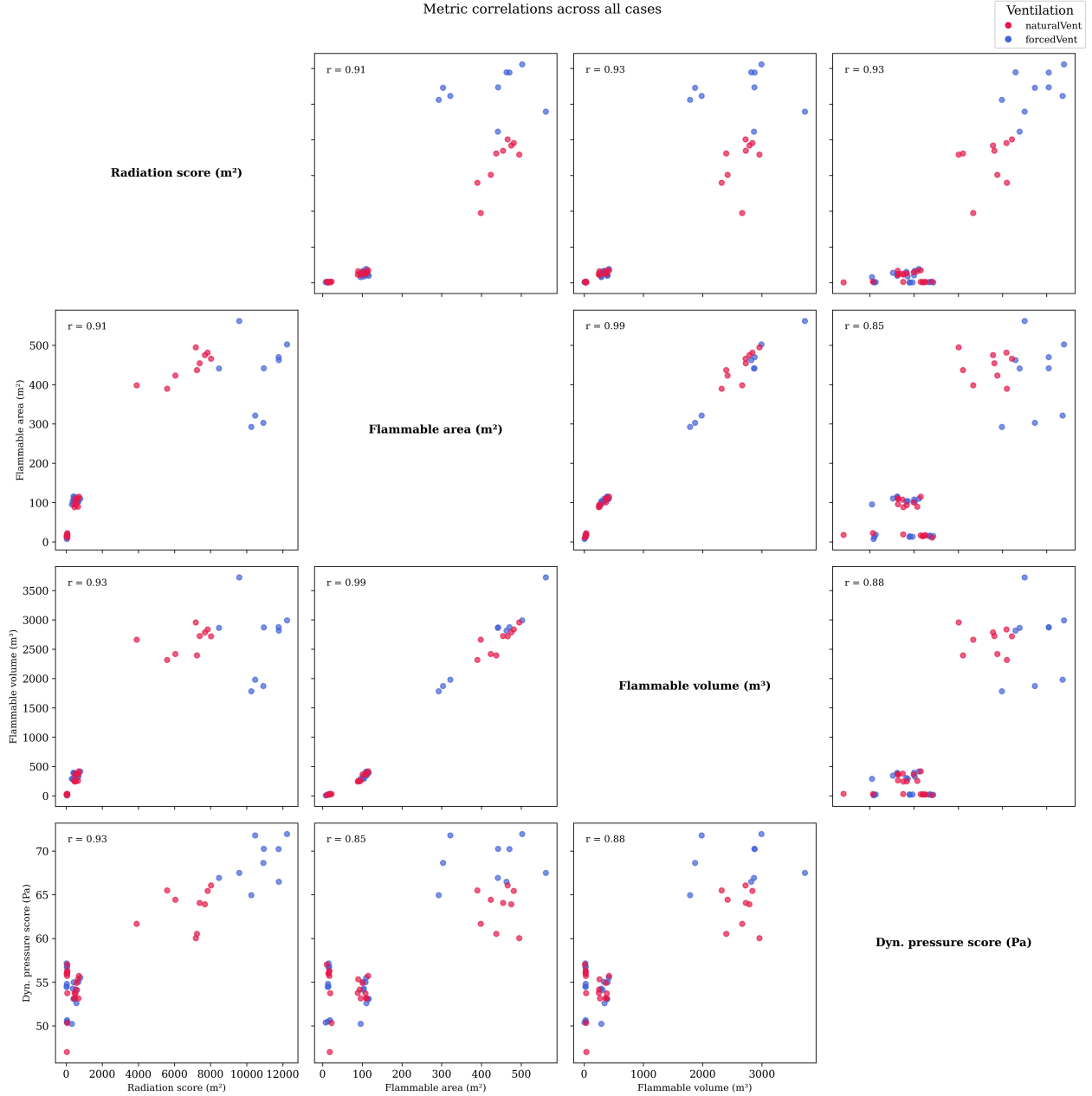


Figure 4.6: Scatter confronting flammable area and volume, radiation score and dynamic pressure score.

Conclusion and further work

This work explored the consequences of hydrogen leaks happening in the compressor enclosure of a hydrogen refuelling station. It is a first step on consequence analysis using CFD of hydrogen leaks in specifically one of the components with the highest failure rates, the compressor, comparing types of ventilation, different pressures, and hole sizes. The findings were many, but in many cases not conclusive because of the lack of statistical significance due to the low number of simulations, as well as their being highly susceptible to geometry and wind conditions.

The flammable area shows a strong correlation to the flammable volume, but it appears to be very much affected by the geometry and may not be a very reliable metric. The same thing can be said about the maximum distance from the leak point. Because of the leak being impinged in all cases and the presence of two physical barriers between it and the open area where customers and passersby would be, the results show that the risk of stones being picked up and creating further problems in the nearby area is very low. However, due to the small dimensions of the compressor enclosure, it was found that in many cases a complete displacement of air was to be expected, which could lead to significant overpressures driven by the pressure peaking phenomenon, which may have concerning consequences. Radiation was successfully contained by the firewalls in the vast majority of the 1% and 10% area scenarios, but not in many of the full bore scenarios, in which the nearby area would be heavily affected. The wind played a strong role in these cases, the results showing that the side of the station the wind was favouring would be safe, but the radiation on the other side would be very high. This is an important finding to include in future HRS design, and is very much dependent on location. The forced ventilation showed a tendency to be helpful when the leaks were small, but detrimental when they were large. In general, the only variable that appears to strongly affect the results is the hole size. The other variables display weaker effects, and should be investigated further. In particular, even though a higher pressure is expected to have more devastating consequences, this is not supported by the findings here. The firewall height was found to be key in containing the fire and protecting the surroundings, but at the same time a higher wall hinders the dissipation of the cloud. A 5m wall was found to be optimal.

The present study can be thought of as a step forward in consequence analysis of hydrogen leaks in HRSs, but further work should be done: (1) further investigation of the effects of geometry and optimal layout; (2) analysis of the effect of the geometry based on the locations of the leaks; (3) study of the scenarios that were found to be likely to experience structural

damage because of overpressure and analysis of their consequences to establish their severity; (4) frequency analysis of these harshest scenarios, to therefore combine this metric with their severity and be able to establish their risk; (5) experimental validation of the results; (6) consequence analysis of leaks originated in the dispenser, which is the other most frequent location and in addition directly in contact with people; (7) analysis of the other safety barriers, to optimise their design.

Bibliography

- [1] Y. S. H. Najjar, “Hydrogen safety: The road toward green technology,” *International Journal of Hydrogen Energy*, vol. 38, no. 25, pp. 10 716–10 728, Aug. 21, 2013, ISSN: 0360-3199. DOI: [10.1016/j.ijhydene.2013.05.126](https://doi.org/10.1016/j.ijhydene.2013.05.126). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S036031991301358X> (visited on 05/25/2026).
- [2] R. A. Mensah, A. Correa, S. Asante-Okyere, C. Wallmark, and M. Försth, “A comprehensive review of risks and mitigation strategies for safe hydrogen infrastructure deployment,” *Frontiers in Chemical Engineering*, vol. 8, Jan. 27, 2026, ISSN: 2673-2718. DOI: [10.3389/fceng.2026.1721648](https://doi.org/10.3389/fceng.2026.1721648). [Online]. Available: <https://www.frontiersin.org/journals/chemical-engineering/articles/10.3389/fceng.2026.1721648/full> (visited on 06/11/2026).
- [3] C. Dong, Q. Zhou, X. Zhang, Q. Zhao, T. Xu, and S. Hui, “Experimental study on the laminar flame speed of hydrogen/natural gas/air mixtures,” *Frontiers of Chemical Engineering in China*, vol. 4, no. 4, pp. 417–422, Dec. 1, 2010, ISSN: 1673-7474. DOI: [10.1007/s11705-010-0515-8](https://doi.org/10.1007/s11705-010-0515-8). [Online]. Available: <https://doi.org/10.1007/s11705-010-0515-8> (visited on 06/25/2026).
- [4] A. Raj *et al.*, “Evaluating hydrogen gas transport in pipelines: Current state of numerical and experimental methodologies,” *International Journal of Hydrogen Energy*, vol. 67, pp. 136–149, May 20, 2024, ISSN: 0360-3199. DOI: [10.1016/j.ijhydene.2024.04.140](https://doi.org/10.1016/j.ijhydene.2024.04.140). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360319924014137> (visited on 06/25/2026).
- [5] Y. Tian *et al.*, “Methodology for optimally designing firewalls in hydrogen refueling stations,” *International Journal of Hydrogen Energy*, vol. 49, pp. 1196–1209, Jan. 2, 2024, ISSN: 0360-3199. DOI: [10.1016/j.ijhydene.2023.08.051](https://doi.org/10.1016/j.ijhydene.2023.08.051). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360319923040053> (visited on 06/11/2026).
- [6] L. Kumar and A. K. Sleiti, “A comprehensive review of hydrogen safety through a metadata analysis framework,” *Renewable and Sustainable Energy Reviews*, vol. 214, p. 115 509, May 1, 2025, ISSN: 1364-0321. DOI: [10.1016/j.rser.2025.115509](https://doi.org/10.1016/j.rser.2025.115509). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1364032125001820> (visited on 05/25/2026).

- [7] Y. Xing, J. Wu, Y. Bai, J. Cai, and X. Zhu, “All-process risk modelling of typical accidents in urban hydrogen refueling stations,” *Process Safety and Environmental Protection*, vol. 166, pp. 414–429, Oct. 2022, ISSN: 09575820. DOI: [10.1016/j.psep.2022.08.047](https://doi.org/10.1016/j.psep.2022.08.047). [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0957582022007340> (visited on 01/27/2026).
- [8] Y. Li, L. Yu, and Q. Jing, “Dynamic risk assessment method for urban hydrogen refueling stations: A novel dynamic Bayesian network incorporating multiple equipment states and accident cascade effects,” *International Journal of Hydrogen Energy*, vol. 54, pp. 1367–1385, Feb. 2024, ISSN: 03603199. DOI: [10.1016/j.ijhydene.2023.11.324](https://doi.org/10.1016/j.ijhydene.2023.11.324). [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0360319923061633> (visited on 01/27/2026).
- [9] J. Shi, Y. Chang, F. Khan, Y. Zhu, and G. Chen, “Methodological improvements in the risk analysis of an urban hydrogen fueling station,” *Journal of Cleaner Production*, vol. 257, p. 120545, Jun. 2020, ISSN: 09596526. DOI: [10.1016/j.jclepro.2020.120545](https://doi.org/10.1016/j.jclepro.2020.120545). [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0959652620305928> (visited on 01/27/2026).
- [10] G. Yang, D. Kong, X. He, X. Yu, and K. Jiang, “Prediction of hydrogen leakage location and intensity in hydrogen refueling stations based on deep learning,” *International Journal of Hydrogen Energy*, vol. 68, pp. 209–220, May 2024, ISSN: 03603199. DOI: [10.1016/j.ijhydene.2024.04.234](https://doi.org/10.1016/j.ijhydene.2024.04.234). [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0360319924015325> (visited on 01/27/2026).
- [11] X. Wang and W. Gao, “Hydrogen leakage risk assessment for hydrogen refueling stations,” *International Journal of Hydrogen Energy*, vol. 48, no. 91, pp. 35795–35808, Nov. 2023, ISSN: 03603199. DOI: [10.1016/j.ijhydene.2023.05.340](https://doi.org/10.1016/j.ijhydene.2023.05.340). [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0360319923027581> (visited on 01/23/2026).
- [12] N. R. Mirza, S. Degenkolbe, and W. Witt, “Analysis of hydrogen incidents to support risk assessment,” *International Journal of Hydrogen Energy*, vol. 36, no. 18, pp. 12068–12077, Sep. 2011, ISSN: 03603199. DOI: [10.1016/j.ijhydene.2011.06.080](https://doi.org/10.1016/j.ijhydene.2011.06.080). [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0360319911015527> (visited on 01/27/2026).
- [13] F. Yang *et al.*, “Review on hydrogen safety issues: Incident statistics, hydrogen diffusion, and detonation process,” *International Journal of Hydrogen Energy*, vol. 46, no. 61, pp. 31467–31488, Sep. 3, 2021, ISSN: 0360-3199. DOI: [10.1016/j.ijhydene.2021.07.005](https://doi.org/10.1016/j.ijhydene.2021.07.005). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360319921025520> (visited on 01/29/2026).
- [14] J. X. Wen *et al.*, “Statistics, lessons learned and recommendations from analysis of HIAD 2.0 database,” *International Journal of Hydrogen Energy*, vol. 47, no. 38, pp. 17082–17096, May 2022, ISSN: 03603199. DOI: [10.1016/j.ijhydene.2022.03.170](https://doi.org/10.1016/j.ijhydene.2022.03.170). [Online].

- Available: <https://linkinghub.elsevier.com/retrieve/pii/S0360319922012976> (visited on 01/29/2026).
- [15] H. Ahmed, “Loss of Containment (LoC) Frequency Model for Hydrogen Refueling Stations using SAFEN Approach,” M.S. thesis, NTNU, Jun. 2026.
- [16] E. Kim, J. Park, J. H. Cho, and I. Moon, “Simulation of hydrogen leak and explosion for the safety design of hydrogen fueling station in Korea,” *International Journal of Hydrogen Energy*, vol. 38, no. 3, pp. 1737–1743, Feb. 2013, ISSN: 03603199. DOI: [10.1016/j.ijhydene.2012.08.079](https://doi.org/10.1016/j.ijhydene.2012.08.079). [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0360319912018812> (visited on 01/23/2026).
- [17] S. H. Han, D. Chang, and J. S. Kim, “Release characteristics of highly pressurized hydrogen through a small hole,” *International Journal of Hydrogen Energy*, vol. 38, no. 8, pp. 3503–3512, Mar. 19, 2013, ISSN: 0360-3199. DOI: [10.1016/j.ijhydene.2012.11.071](https://doi.org/10.1016/j.ijhydene.2012.11.071). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360319912025608> (visited on 03/10/2026).
- [18] S. H. Han, D. Chang, and J. S. Kim, “Experimental investigation of highly pressurized hydrogen release through a small hole,” *International Journal of Hydrogen Energy*, vol. 39, no. 17, pp. 9552–9561, Jun. 2014, ISSN: 03603199. DOI: [10.1016/j.ijhydene.2014.03.044](https://doi.org/10.1016/j.ijhydene.2014.03.044). [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0360319914006983> (visited on 04/07/2026).
- [19] M. S. Butler, C. W. Moran, P. B. Sunderland, and R. L. Axelbaum, “Limits for hydrogen leaks that can support stable flames,” *International Journal of Hydrogen Energy*, UGHEK 2008, vol. 34, no. 12, pp. 5174–5182, Jun. 1, 2009, ISSN: 0360-3199. DOI: [10.1016/j.ijhydene.2009.04.012](https://doi.org/10.1016/j.ijhydene.2009.04.012). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360319909005011> (visited on 03/10/2026).
- [20] K. M. Groth and E. S. Hecht, “HyRAM: A methodology and toolkit for quantitative risk assessment of hydrogen systems,” *International Journal of Hydrogen Energy*, vol. 42, no. 11, pp. 7485–7493, Mar. 2017, ISSN: 03603199. DOI: [10.1016/j.ijhydene.2016.07.002](https://doi.org/10.1016/j.ijhydene.2016.07.002). [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0360319916303640> (visited on 04/07/2026).
- [21] S. Brennan and V. Molkov, “Pressure peaking phenomenon for indoor hydrogen releases,” *International Journal of Hydrogen Energy*, vol. 43, no. 39, pp. 18 530–18 541, Sep. 27, 2018, ISSN: 0360-3199. DOI: [10.1016/j.ijhydene.2018.08.096](https://doi.org/10.1016/j.ijhydene.2018.08.096). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360319918326375> (visited on 05/25/2026).
- [22] M. Genovese, D. Blekhman, and P. Fragiaco, “An Exploration of Safety Measures in Hydrogen Refueling Stations: Delving into Hydrogen Equipment and Technical Performance,” *Hydrogen*, vol. 5, no. 1, pp. 102–122, Feb. 17, 2024, ISSN: 2673-4141. DOI: [10.3390/hydrogen5010007](https://doi.org/10.3390/hydrogen5010007). [Online]. Available: <https://www.mdpi.com/2673-4141/5/1/7> (visited on 02/06/2026).

- [23] K. Tsunemi, T. Kihara, E. Kato, A. Kawamoto, and T. Saburi, “Quantitative risk assessment of the interior of a hydrogen refueling station considering safety barrier systems,” *International Journal of Hydrogen Energy*, vol. 44, no. 41, pp. 23 522–23 531, Aug. 30, 2019, ISSN: 0360-3199. DOI: [10.1016/j.ijhydene.2019.07.027](https://doi.org/10.1016/j.ijhydene.2019.07.027). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360319919325753> (visited on 06/11/2026).
- [24] W. G. Houf, G. H. Evans, R. W. Schefer, E. Merilo, and M. Groethe, “A study of barrier walls for mitigation of unintended releases of hydrogen,” *International Journal of Hydrogen Energy*, The Third Annual International Conference on Hydrogen Safety, vol. 36, no. 3, pp. 2520–2529, Feb. 1, 2011, ISSN: 0360-3199. DOI: [10.1016/j.ijhydene.2010.04.003](https://doi.org/10.1016/j.ijhydene.2010.04.003). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360319910006658> (visited on 06/11/2026).
- [25] “KFX: CFD simulation software for dispersions and fires,” DNV. (), [Online]. Available: <https://www.dnv.com/services/kfx/> (visited on 05/20/2026).
- [26] R. Ghimire, “Dispersion and ignition probability modelling of accidental hydrogen release: A CFD-based approach,” M.S. thesis, NTNU, 2025.
- [27] James E. Francfort, “Arizona Public Service - Alternative Fuel (Hydrogen) Pilot Plant Design Report,” INEEL/EXT-03-00976, 910735, Dec. 1, 2003, INEEL/EXT-03-00 976, 910 735. DOI: [10.2172/910735](https://doi.org/10.2172/910735). [Online]. Available: <http://www.osti.gov/servlets/purl/910735-rcyNDm/> (visited on 03/31/2026).
- [28] “Guidance Note on Quantitative Risk Assessment Study for High Pressure Town Gas Installations In Hong Kong,”
- [29] “IEC 60079-10-1:2020.” (), [Online]. Available: <https://webstore.iec.ch/en/publication/63327> (visited on 04/08/2026).
- [30] “Ventilation | H2tools | Hydrogen Tools.” (), [Online]. Available: <https://h2tools.org/bestpractices/hydrogen-properties-and-leak-detection-considerations/ventilation> (visited on 04/21/2026).
- [31] “Vindrose med frekvensfordeling - Seklima.” (), [Online]. Available: https://seklima.met.no/windrose/?timeresolution=last_30_years&locationid=SN68860 (visited on 05/20/2026).
- [32] F. |. Dotpeople. “ZerAx AZN axial flow fans – industrial and smoke exhaust fans | NOVENCO.” (), [Online]. Available: <https://www.novenco-building.com/products/axial-flow-fans/zerax-fans/azn/> (visited on 04/21/2026).
- [33] F. |. Dotpeople. “AirBox™ fan configuration and selection software for NOVENCO axial and centrifugal fans.” (), [Online]. Available: <https://www.novenco-building.com/services/software-airbox/> (visited on 04/21/2026).

- [34] “UBC ATSC 113 - Standard Atmosphere-Pressure.” (), [Online]. Available: https://www.eoas.ubc.ca/courses/atsc113/flying/met_concepts/02-met_concepts/02a-std_atmos-P/index.html (visited on 04/21/2026).
- [35] “Dynamic Pressure.” (), [Online]. Available: <https://www.grc.nasa.gov/www/BGH/dynpress.html> (visited on 06/08/2026).
- [36] A. Carpignano, *RIsk Analysis*, Lecture notes, Politecnico di Torino, Dipartimento di Energetica, Jan. 9, 9.
- [37] P. K. Raj, “A review of the criteria for people exposure to radiant heat flux from fires,” *Journal of Hazardous Materials*, vol. 159, no. 1, pp. 61–71, Nov. 2008, issn: 03043894. DOI: [10.1016/j.jhazmat.2007.09.120](https://doi.org/10.1016/j.jhazmat.2007.09.120). [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0304389407014197> (visited on 06/16/2026).
- [38] T. N. O. Green-Book, “Method for the determination of possible damage to people and objects resulting from releases of hazardous materials,” *CPR 16E, TNO–The Netherlands Organisation of Applied Scientific Research*,, 1992.

Appendix A

Additional figures

A.1 Exposure to flammable gas

Flammable cloud footprint | Natural ventilation vs Forced ventilation | 430bar, d=1.2 mm | global ranking by area

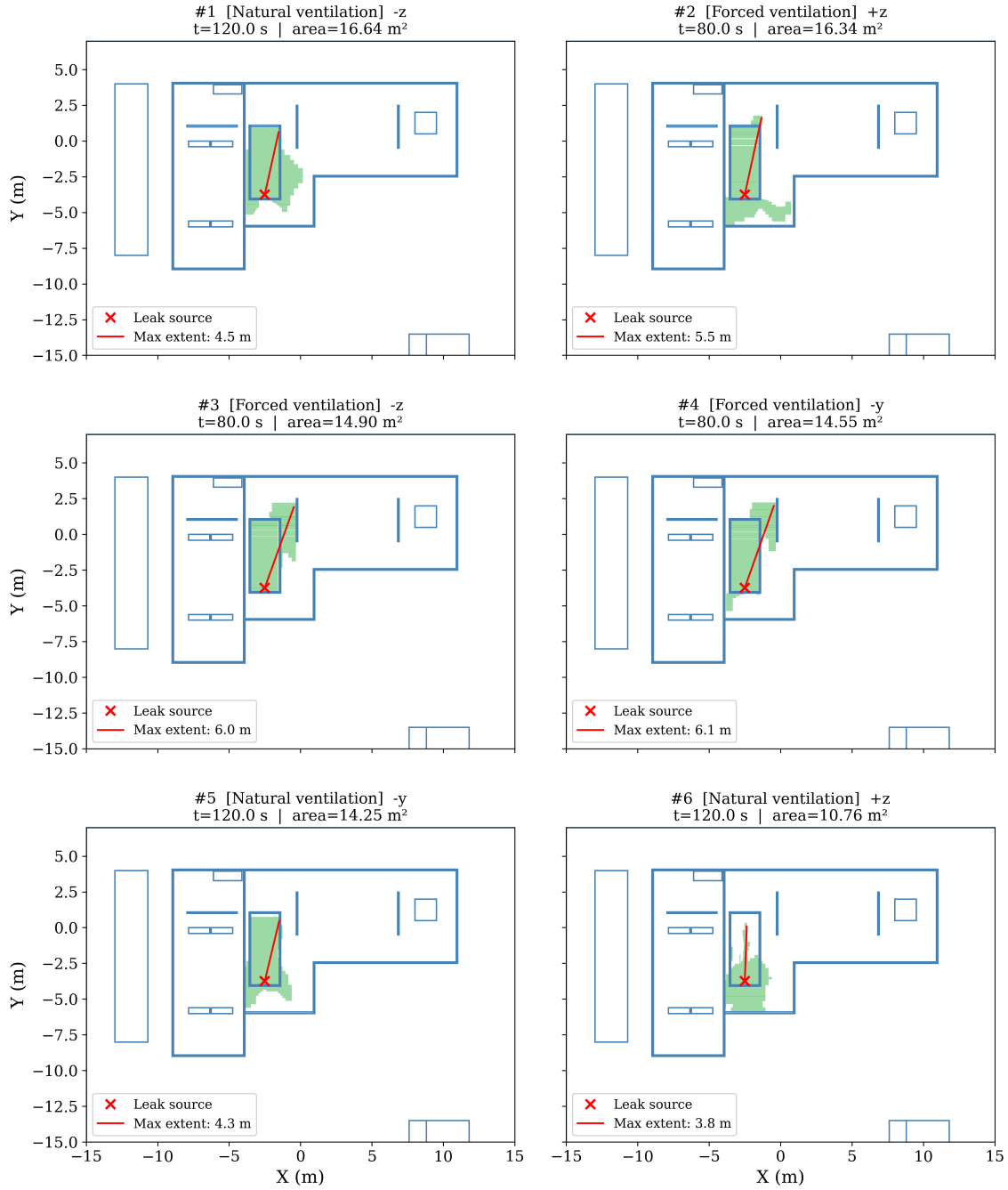


Figure A.1: Flammable area projection at steady state for 430bar, 1% area cases.

Flammable cloud footprint | Natural ventilation vs Forced ventilation | 430bar, d=12.0 mm | global ranking by area

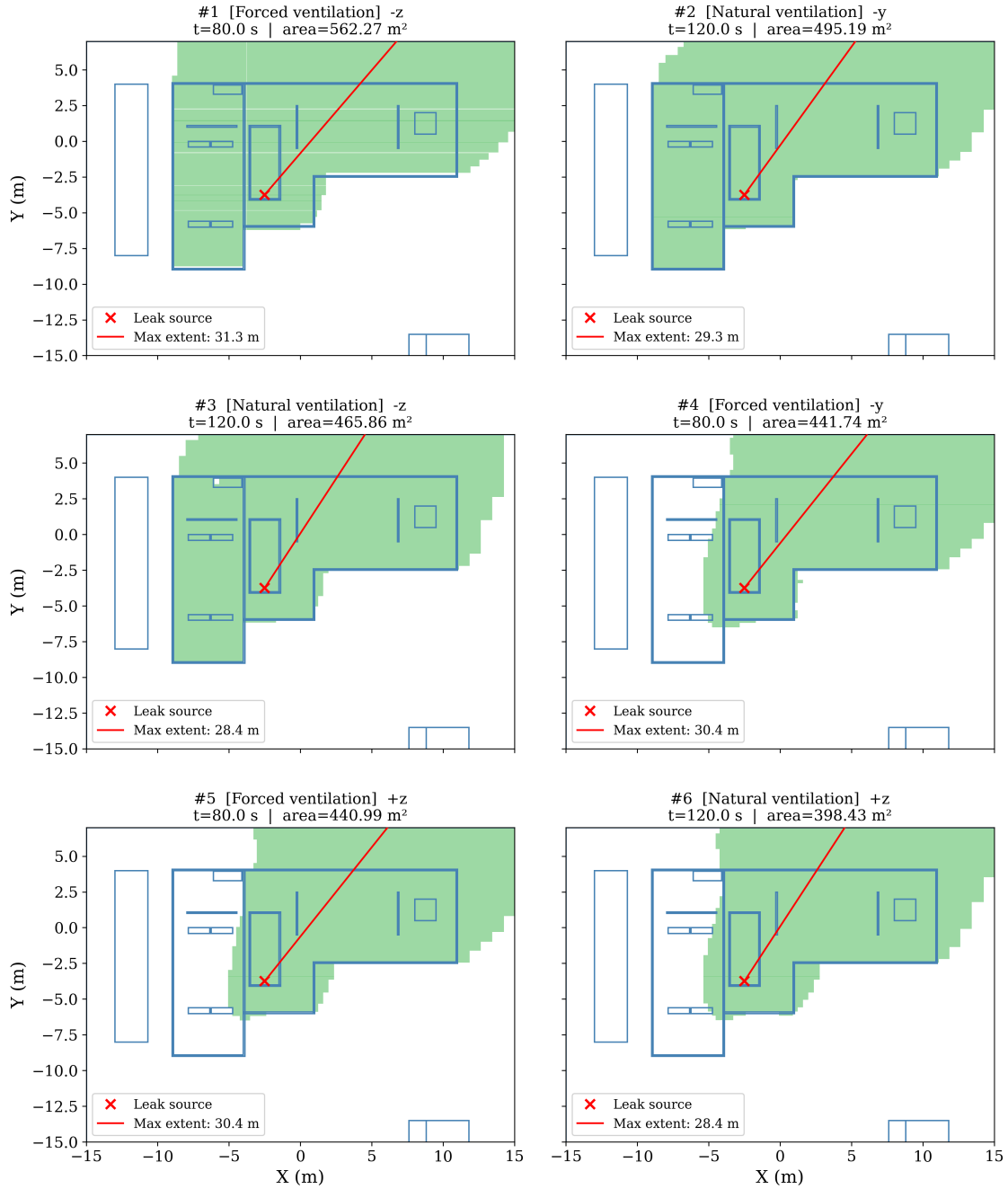


Figure A.2: Flammable area projection at steady state for 430bar, full bore cases.

Flammable cloud footprint | Natural ventilation vs Forced ventilation | 630bar, d=1.0 mm | global ranking by area

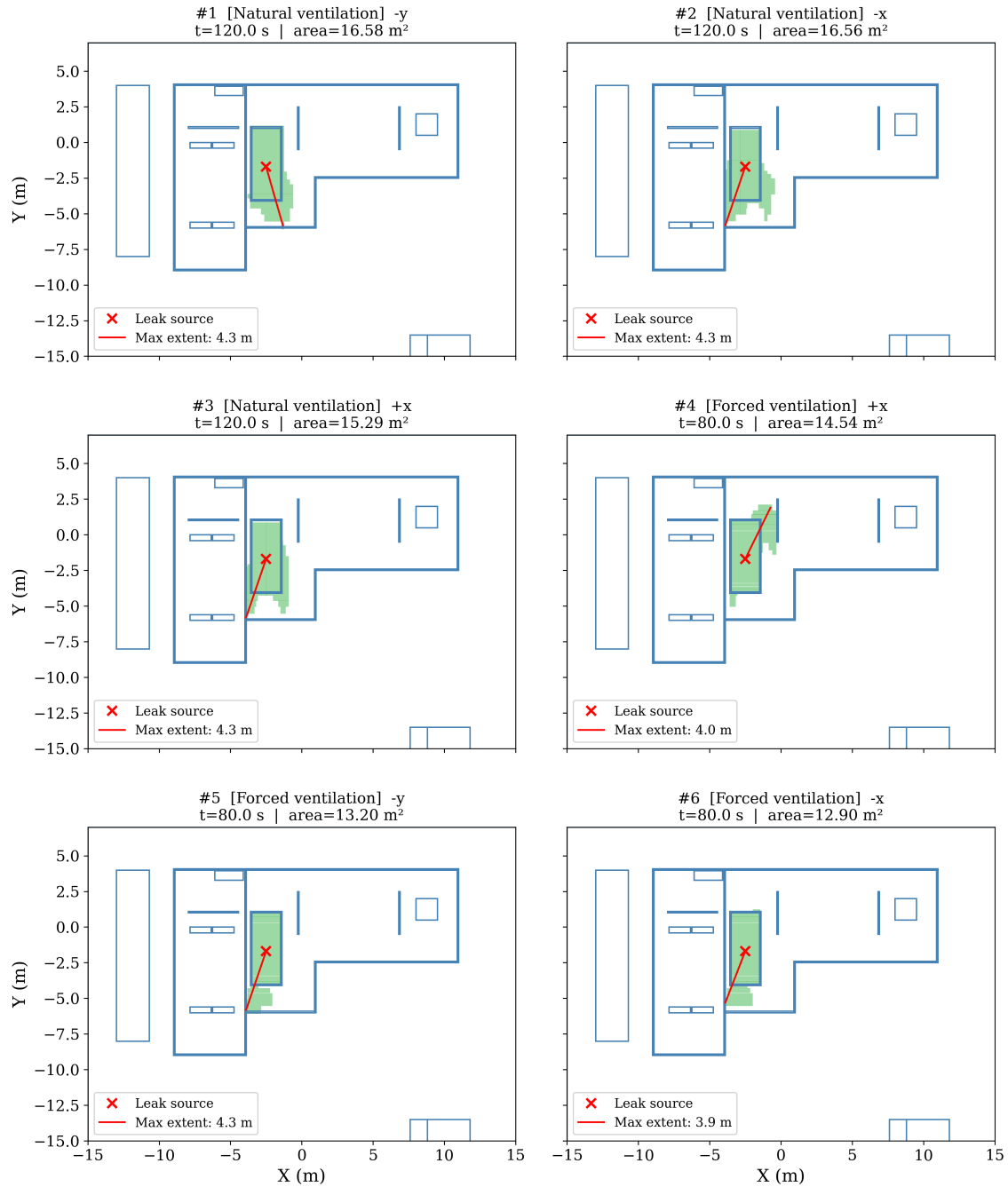


Figure A.3: Flammable area projection at steady state for 630bar, 1% area cases.

Flammable cloud footprint | Natural ventilation vs Forced ventilation | 630bar, d=3.2 mm | global ranking by area

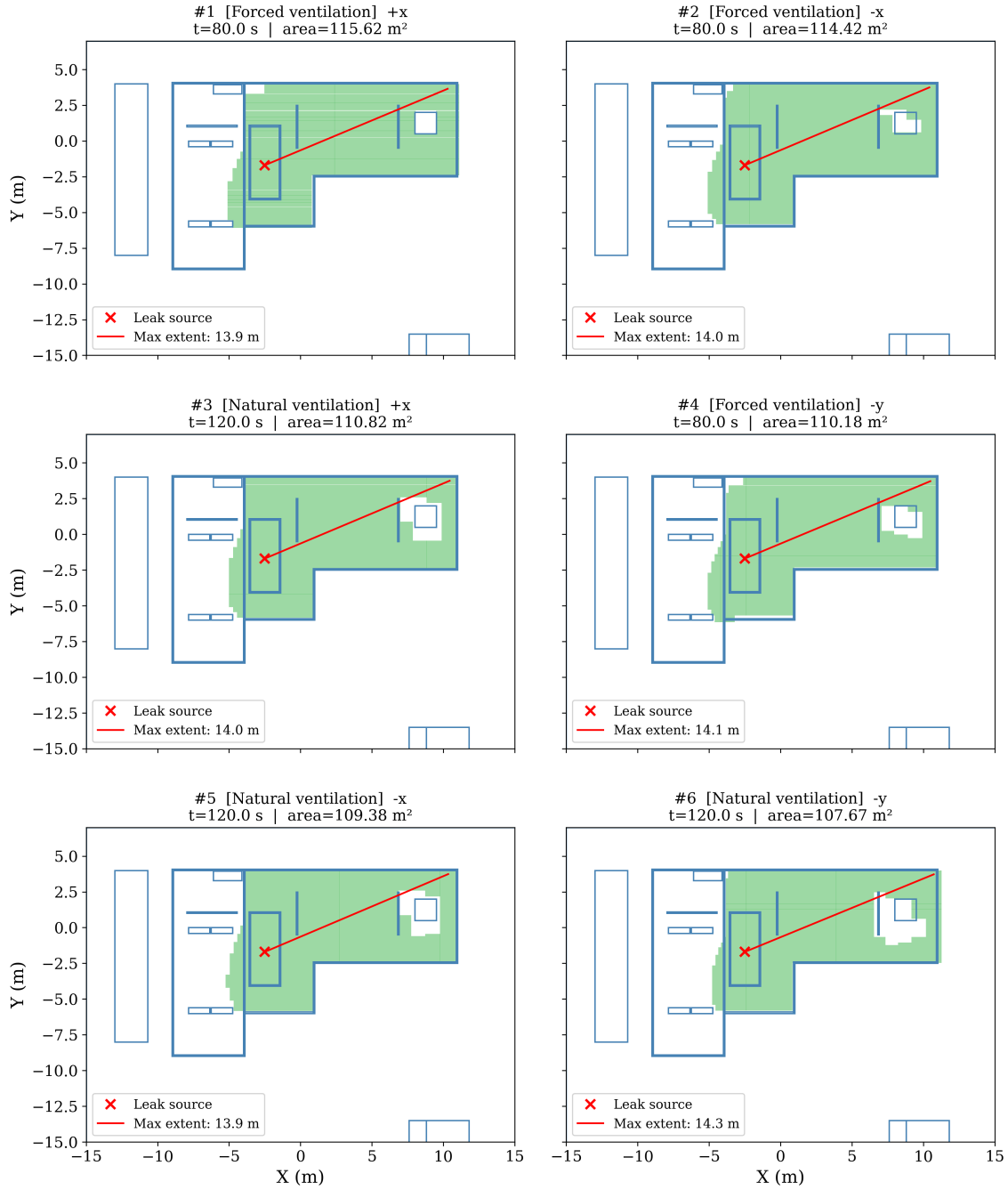


Figure A.4: Flammable area projection at steady state for 630bar, 10% area cases.

Flammable cloud footprint | Natural ventilation vs Forced ventilation | 630bar, d=10.0 mm | global ranking by area

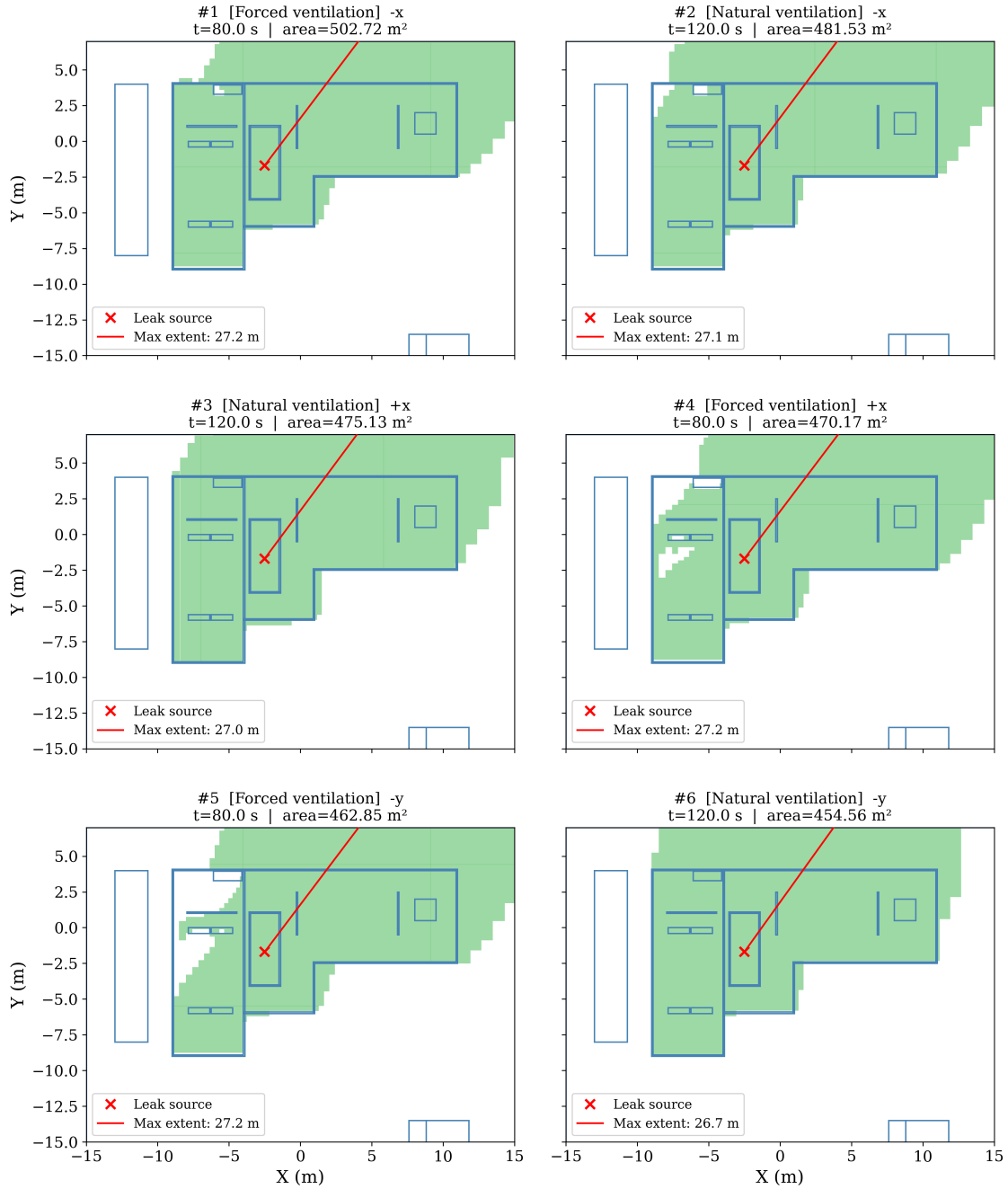


Figure A.5: Flammable area projection at steady state for 630bar, full bore cases.

Flammable cloud footprint | Natural ventilation vs Forced ventilation | 900bar, d=1.0 mm | global ranking by area

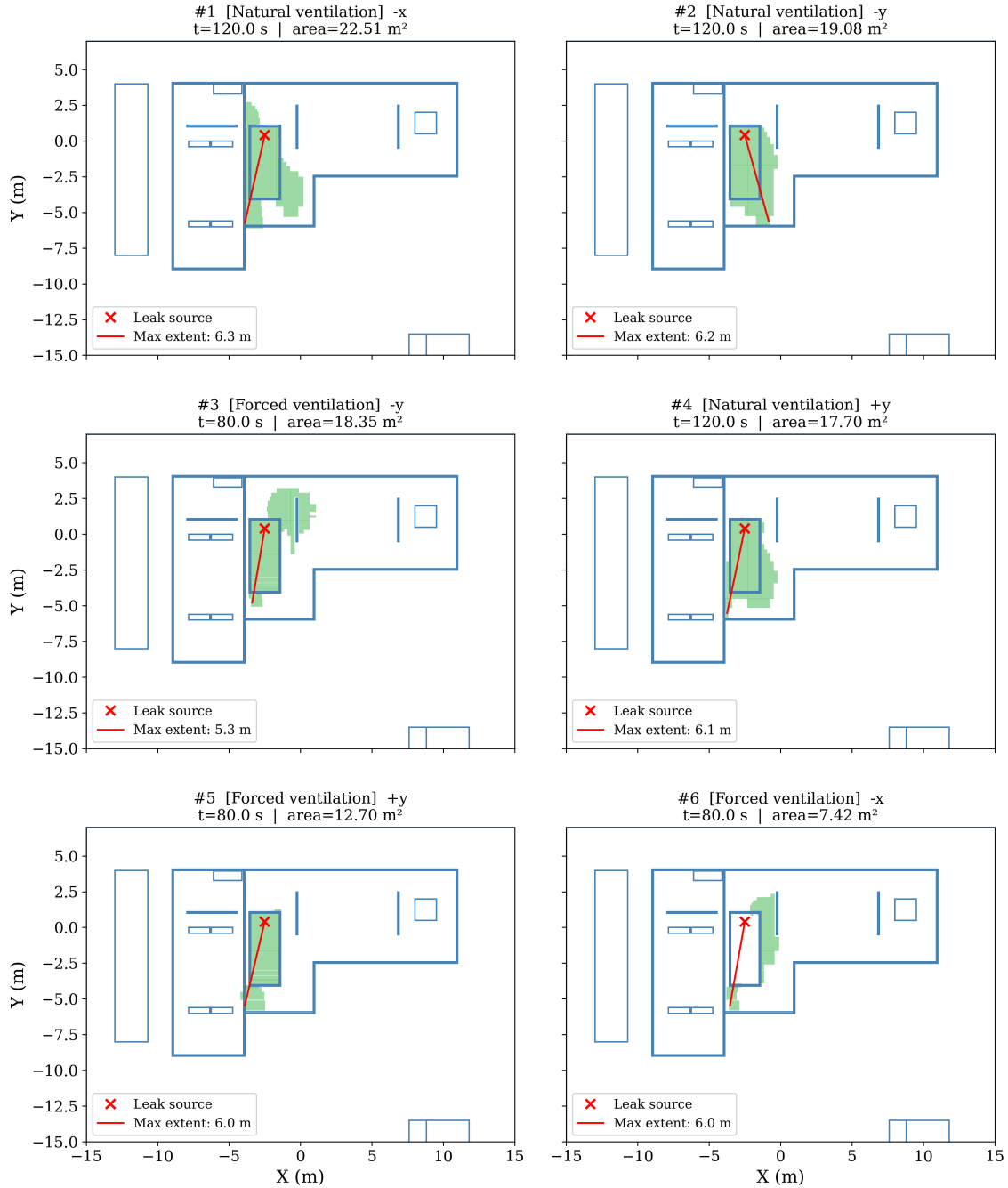


Figure A.6: Flammable area projection at steady state for 900bar, 1% area cases.

Flammable cloud footprint | Natural ventilation vs Forced ventilation | 900bar, d=2.5 mm | global ranking by area

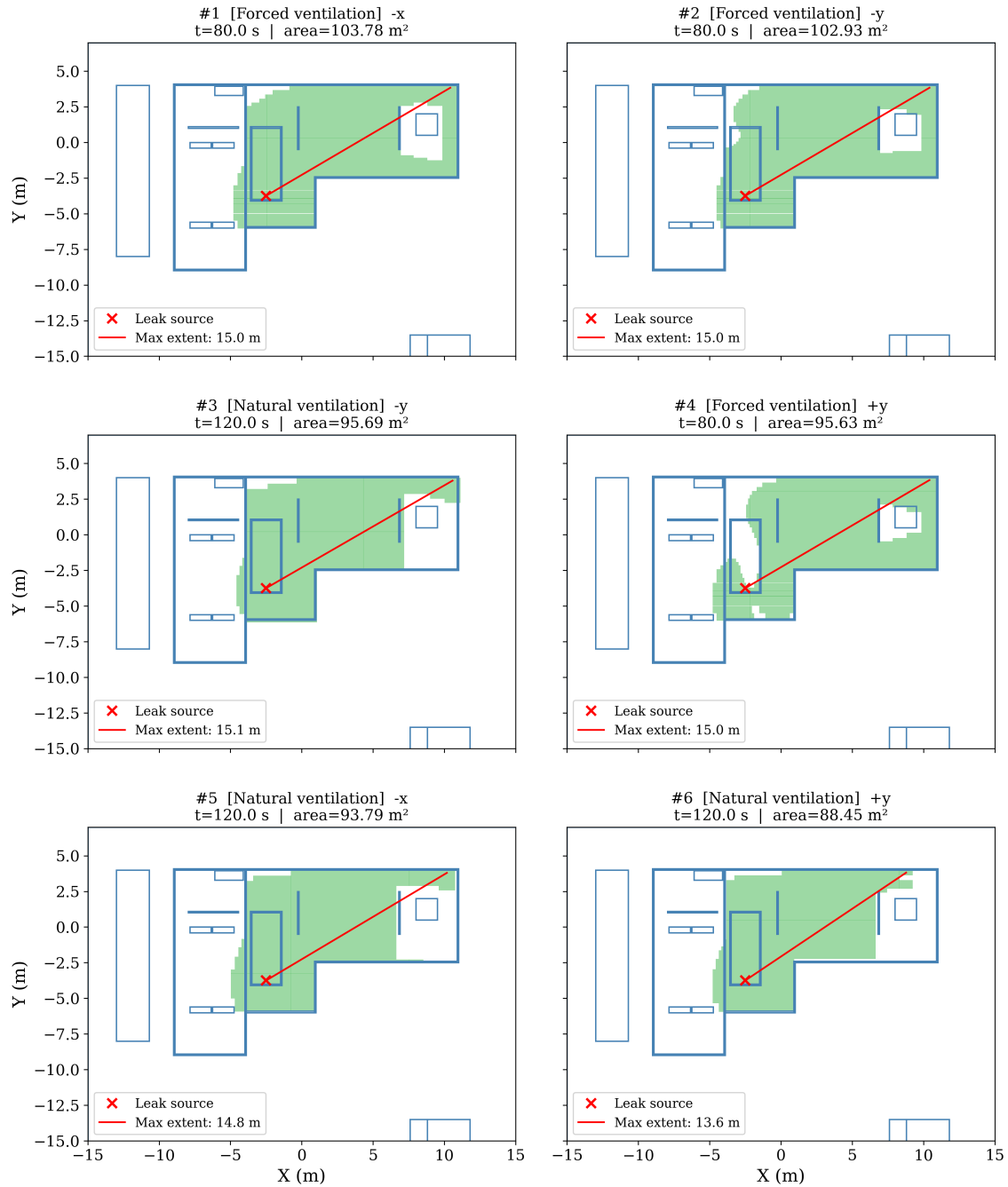


Figure A.7: Flammable area projection at steady state for 900bar, 10% area cases.

Flammable cloud footprint | Natural ventilation vs Forced ventilation | 900bar, d=8.0 mm | global ranking by area

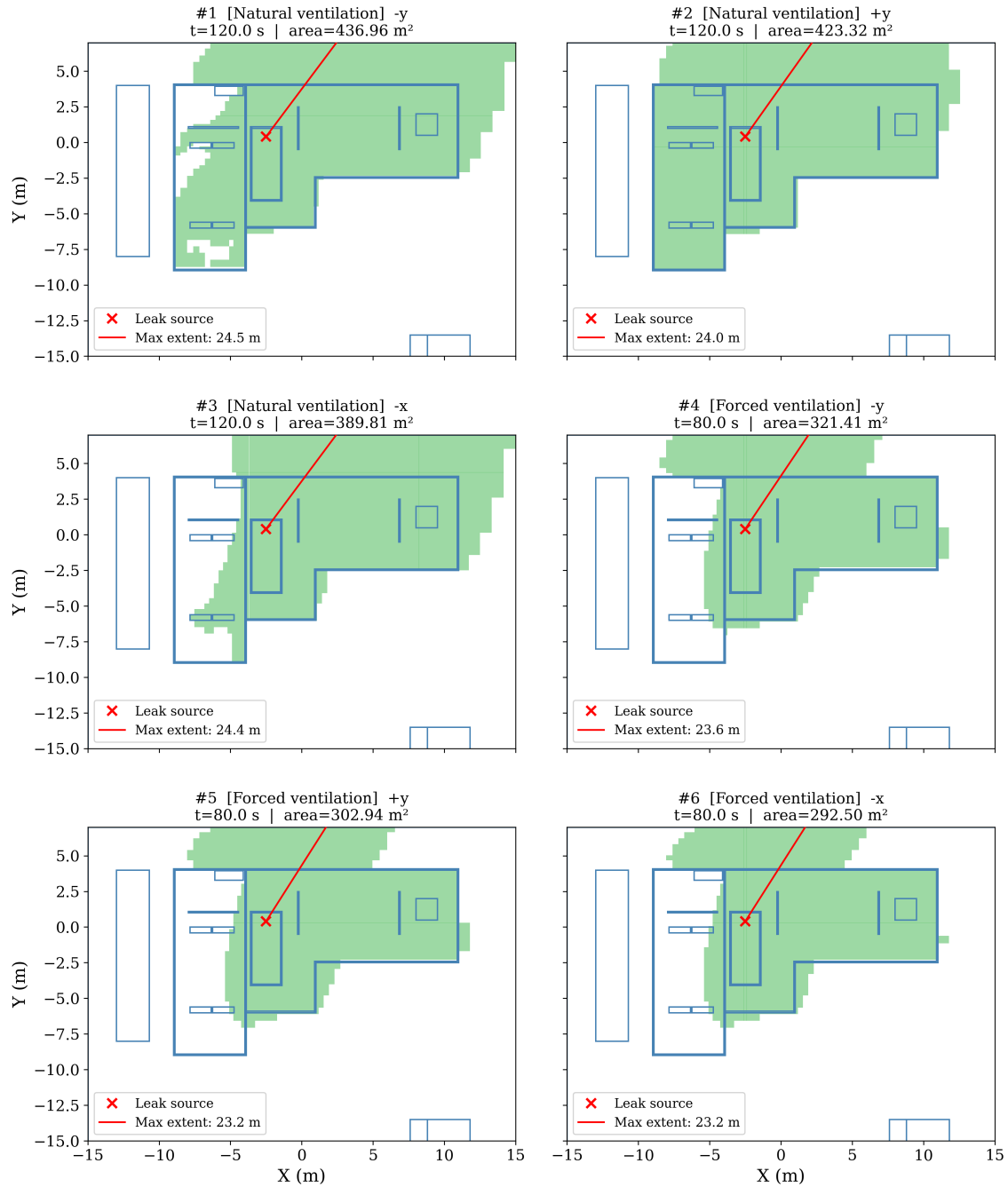


Figure A.8: Flammable area projection at steady state for 900bar, full bore cases.

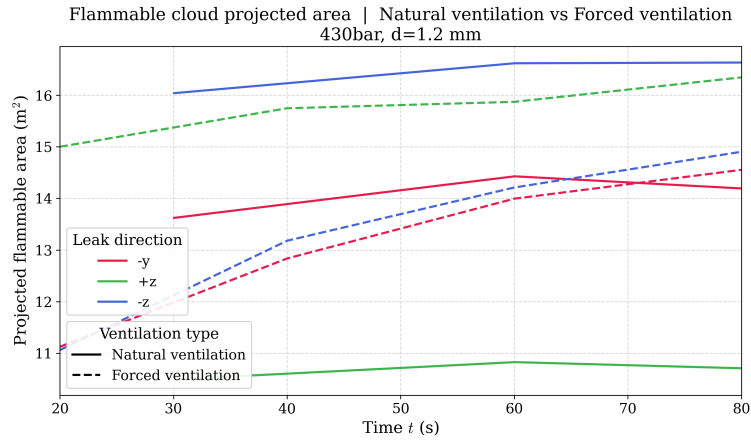


Figure A.9: Flammable area over time for 430bar, 1% area cases.

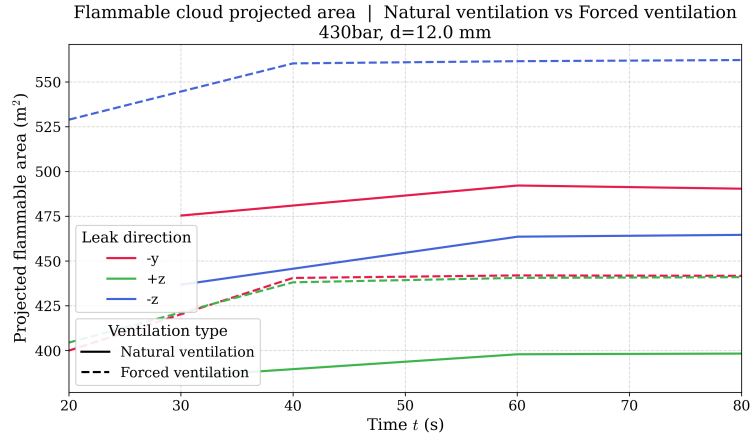


Figure A.10: Flammable area over time for 430bar, full bore cases.

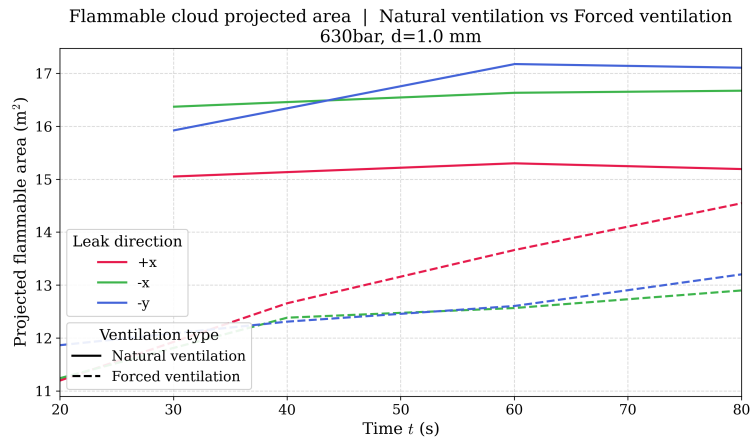


Figure A.11: Flammable area over time for 630bar, 1% area cases.

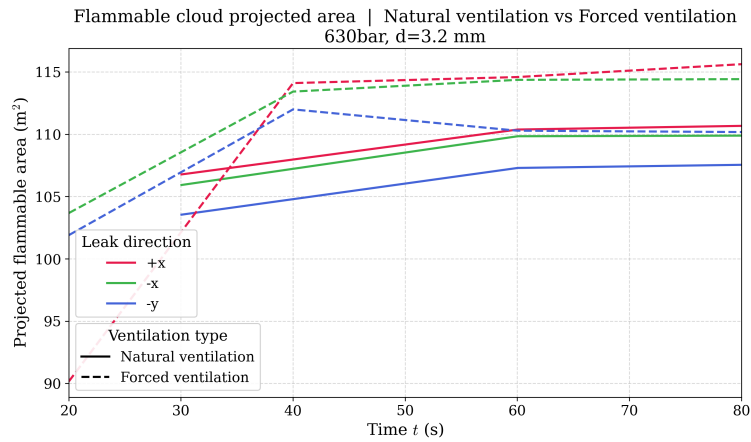


Figure A.12: Flammable area over time for 630bar, 10% area cases.

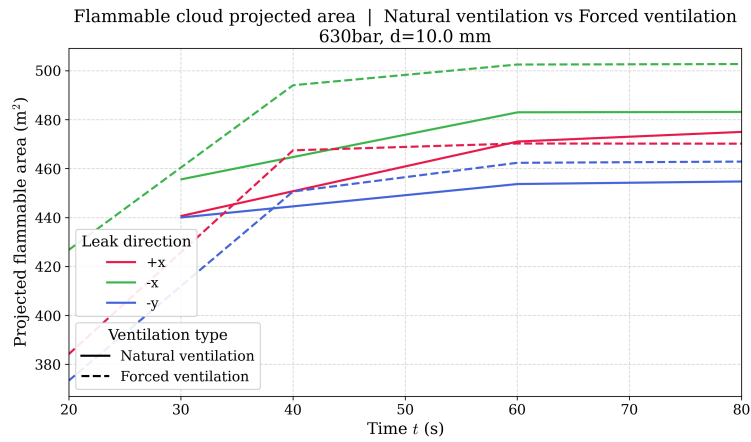


Figure A.13: Flammable area over time for 630bar, full bore cases.

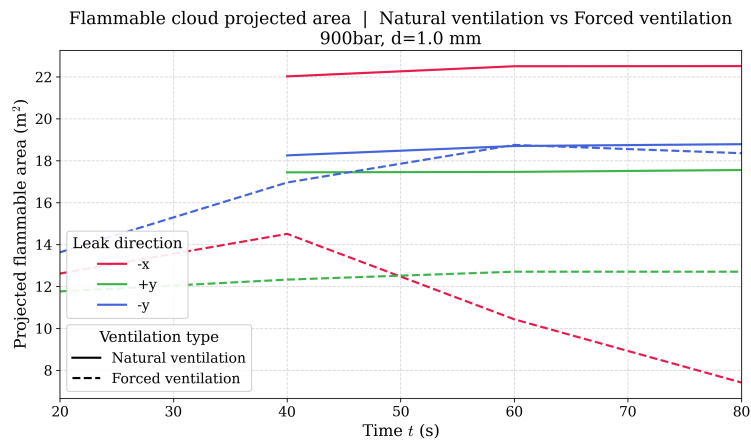


Figure A.14: Flammable area over time for 900bar, 1% area cases.

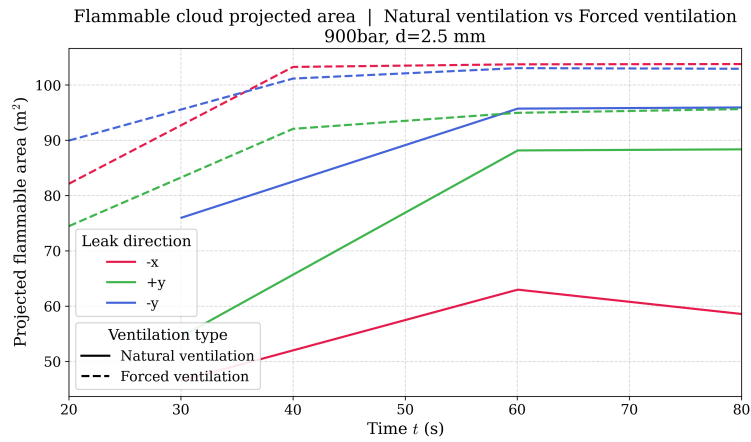


Figure A.15: Flammable area over time for 900bar, 10% area cases.

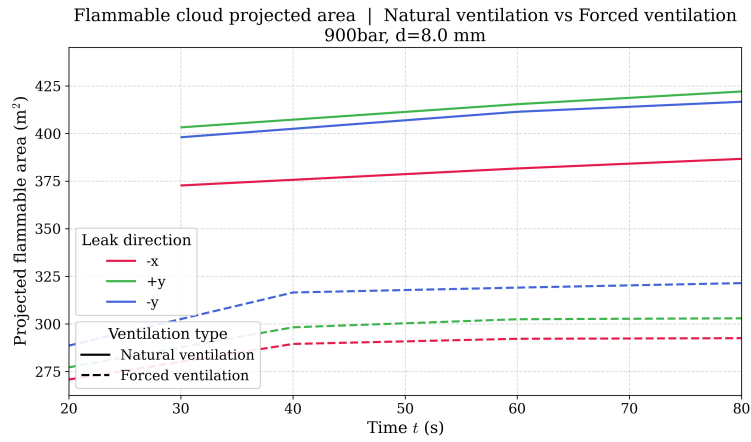


Figure A.16: Flammable area over time for 900bar, full bore cases.

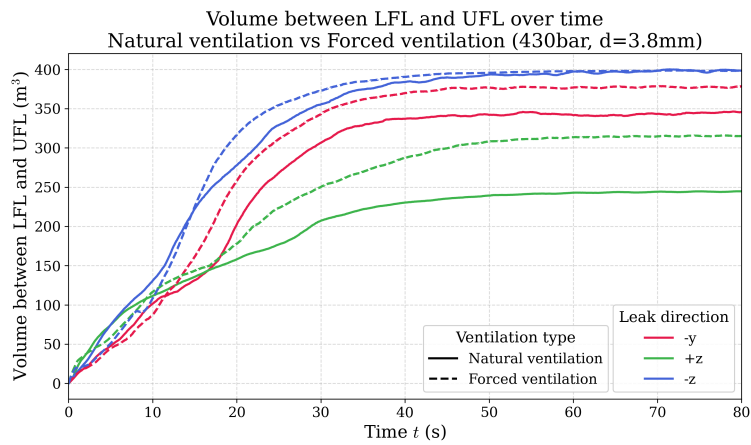


Figure A.17: Volume between LFL and UFL over time for 430bar, 10% area cases.

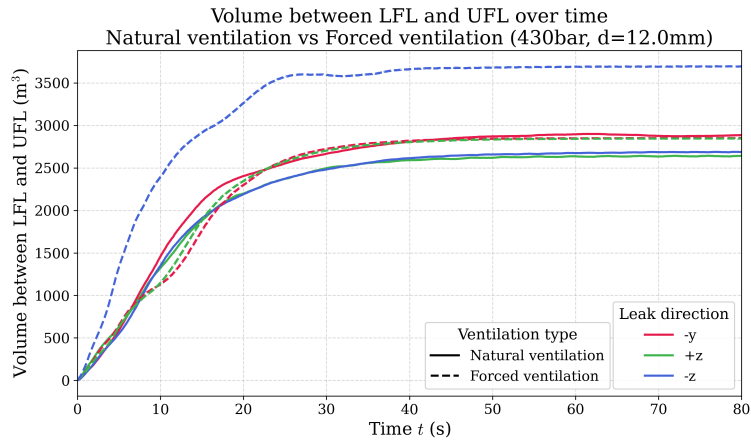


Figure A.18: Volume between LFL and UFL over time for 430bar, full bore cases.

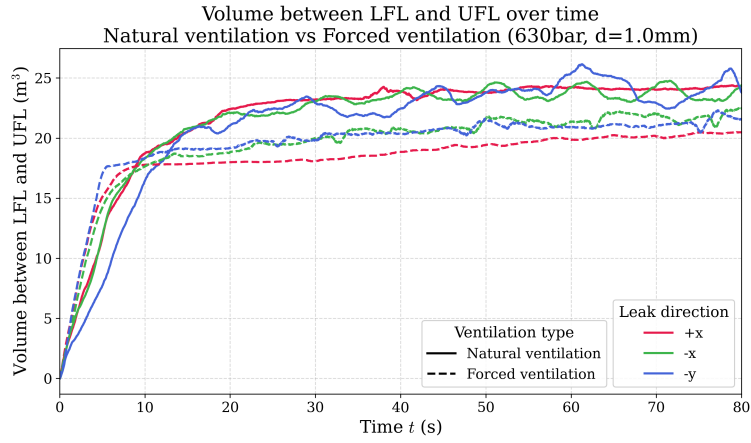


Figure A.19: Volume between LFL and UFL over time for 630bar, 1% area cases.

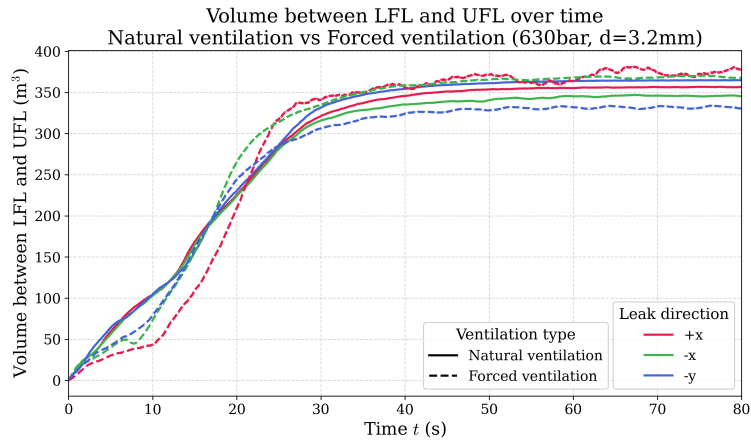


Figure A.20: Volume between LFL and UFL over time for 630bar, 10% area cases.

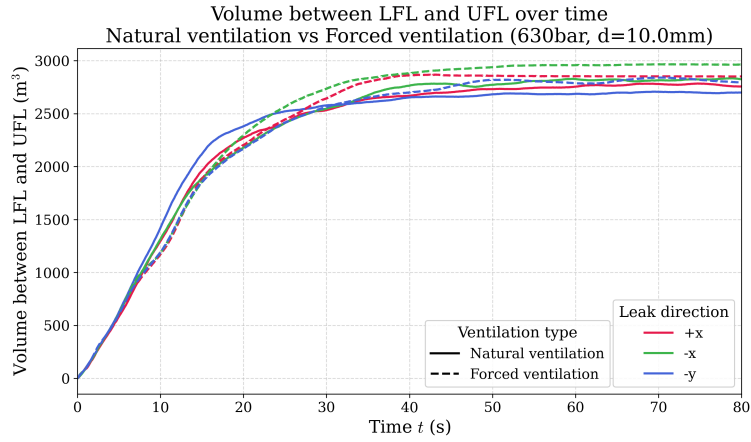


Figure A.21: Volume between LFL and UFL over time for 630bar, full bore cases.

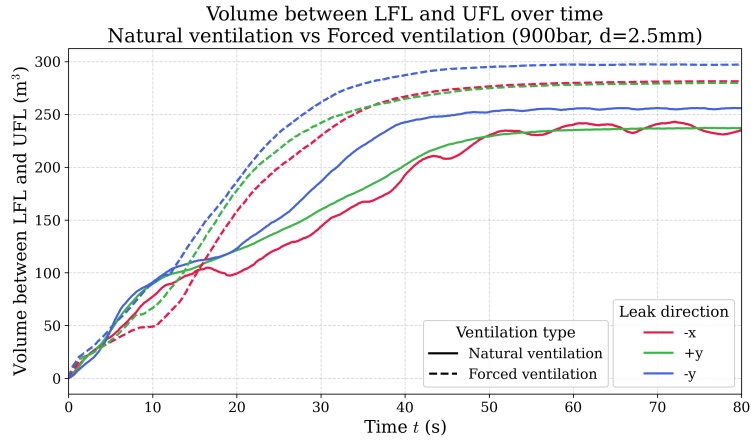


Figure A.22: Volume between LFL and UFL over time for 900bar, 10% area cases.

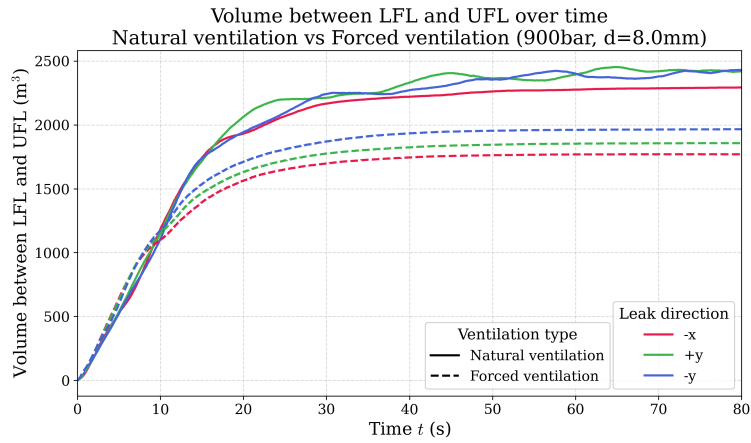


Figure A.23: Volume between LFL and UFL over time for 900bar, full bore cases.

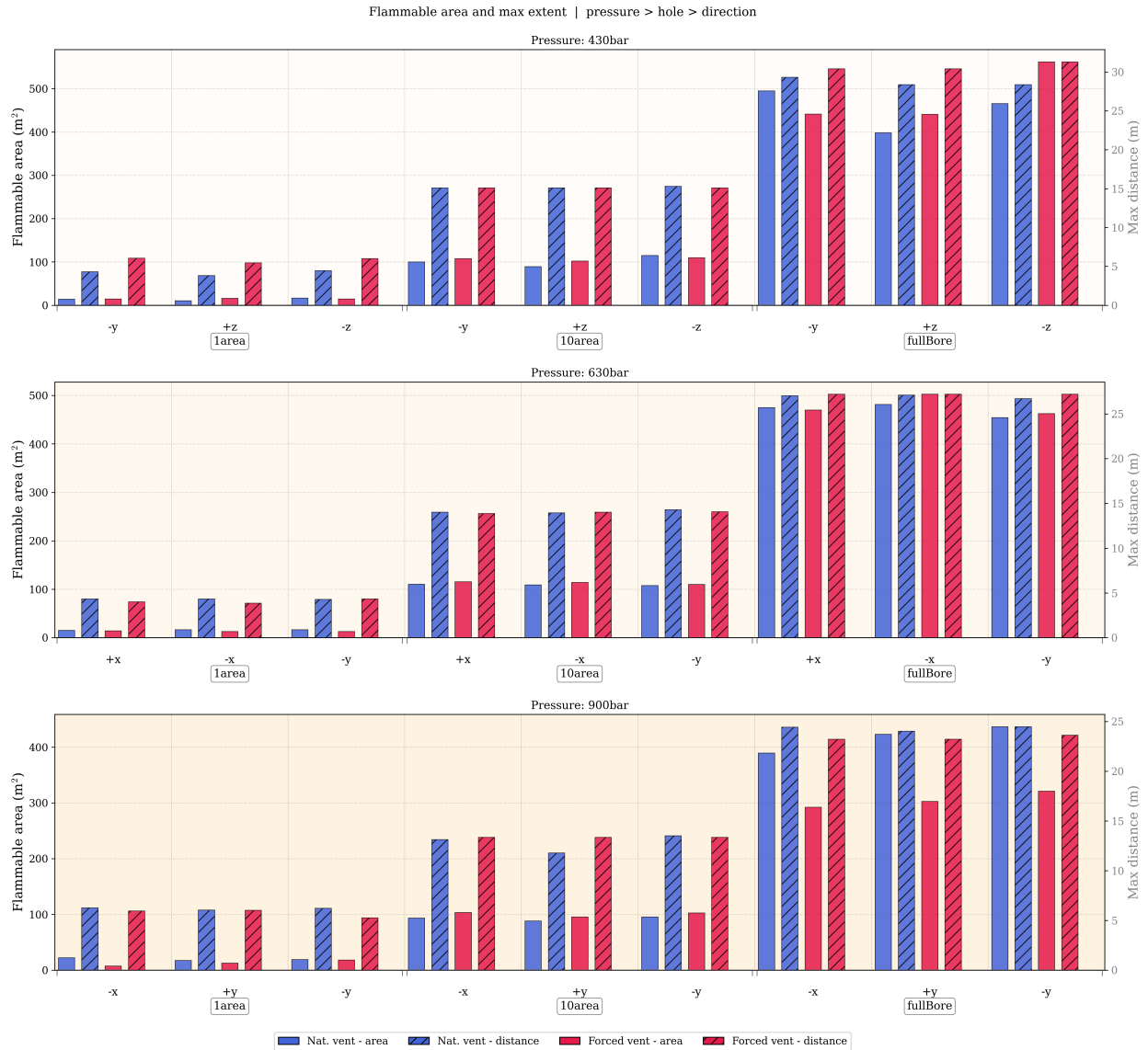


Figure A.24: Summary of the cloud projection of the flammable area and maximum reach at steady state for all the cases organised by a hierarchy of pressure, then hole size, then direction.

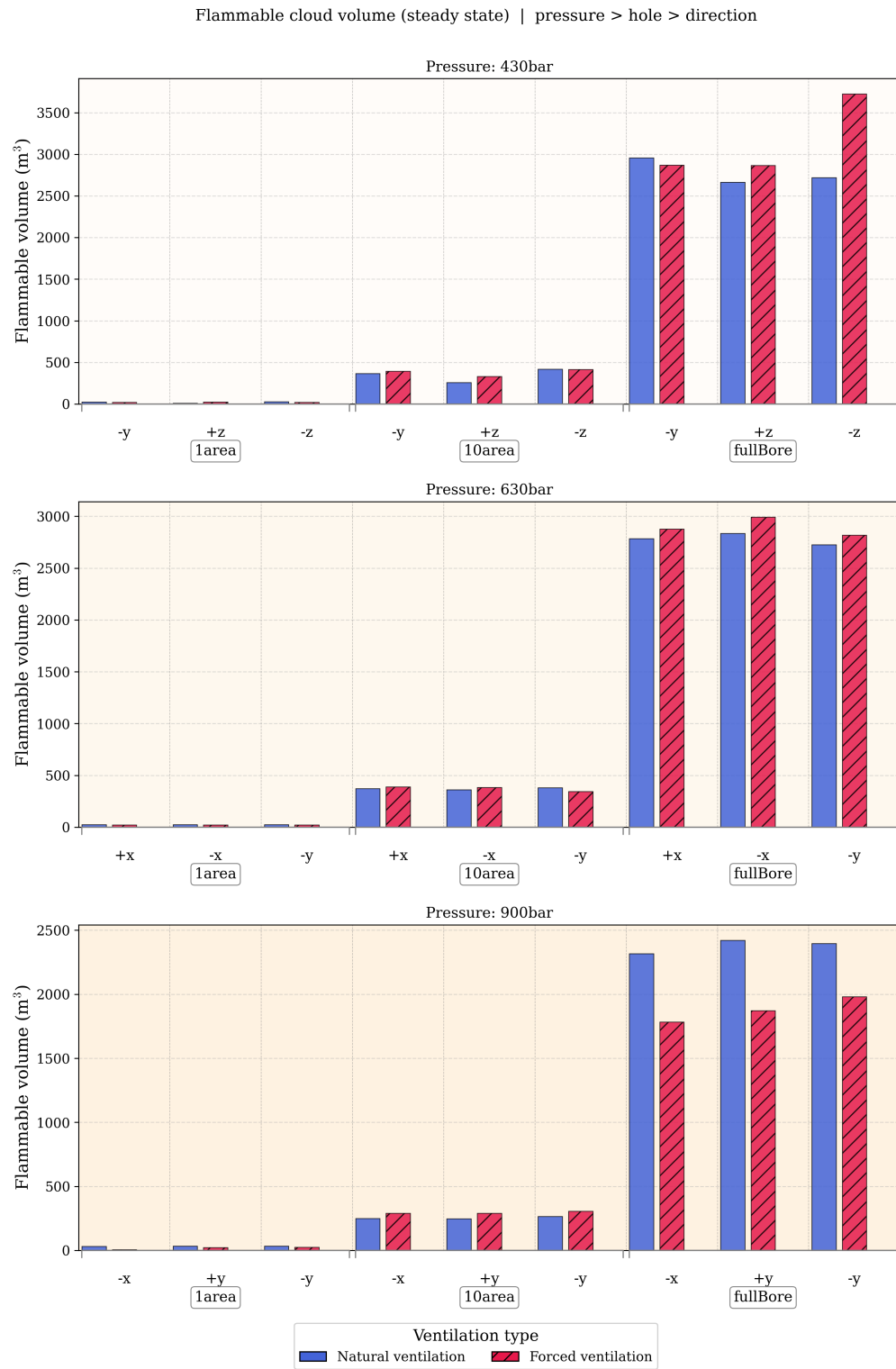


Figure A.25: Summary of the flammable volume of the cloud at steady state for all the cases organised by a hierarchy of pressure, then hole size, then direction.

A.2 Radiation

Radiation comparison | Natural ventilation vs Forced ventilation | 430bar, 1.2 mm | global ranking

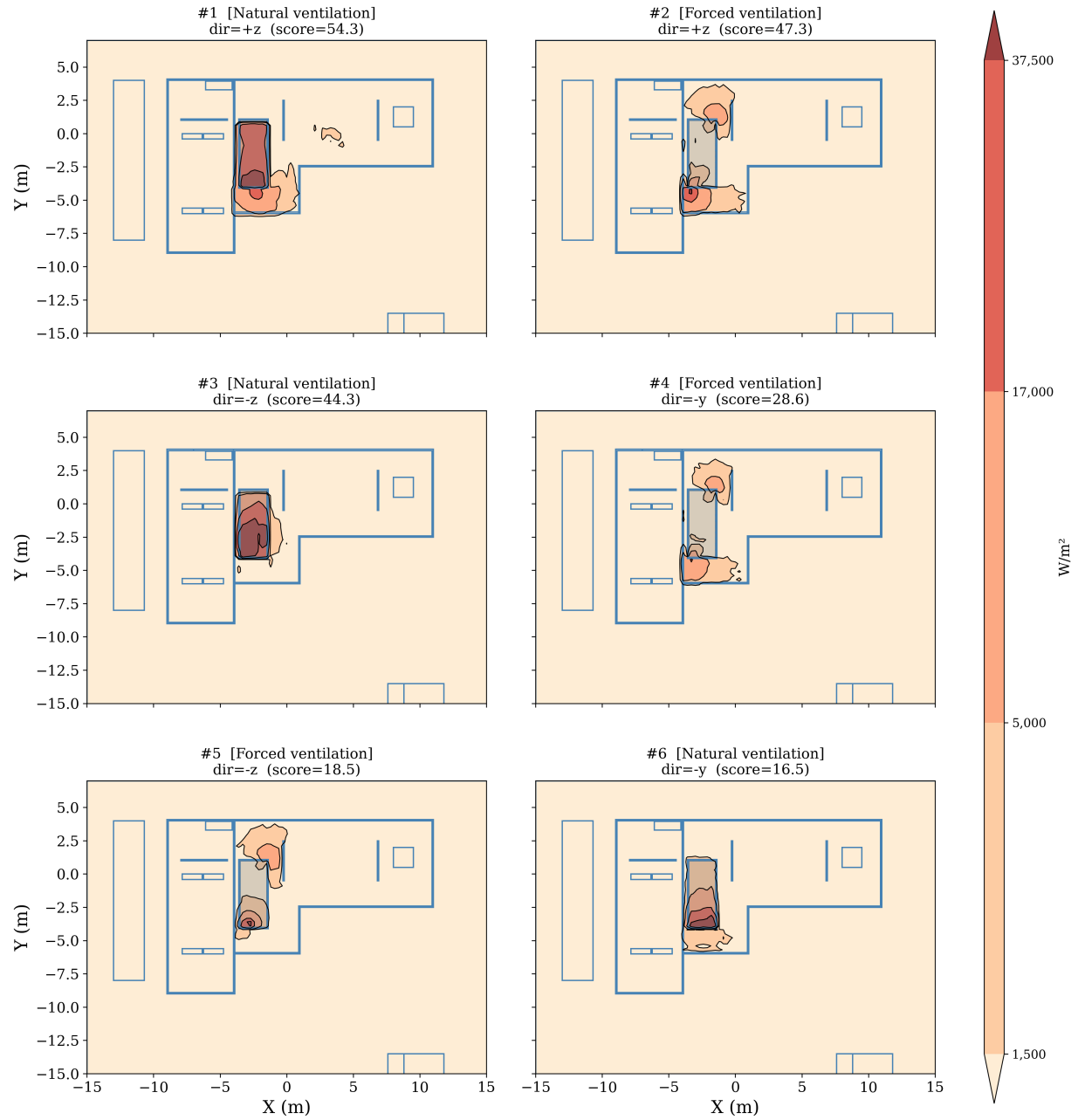


Figure A.26: Radiation in steady state for 430bar, 1% area cases.

Radiation comparison | Natural ventilation vs Forced ventilation | 430bar, 12.0 mm | global ranking

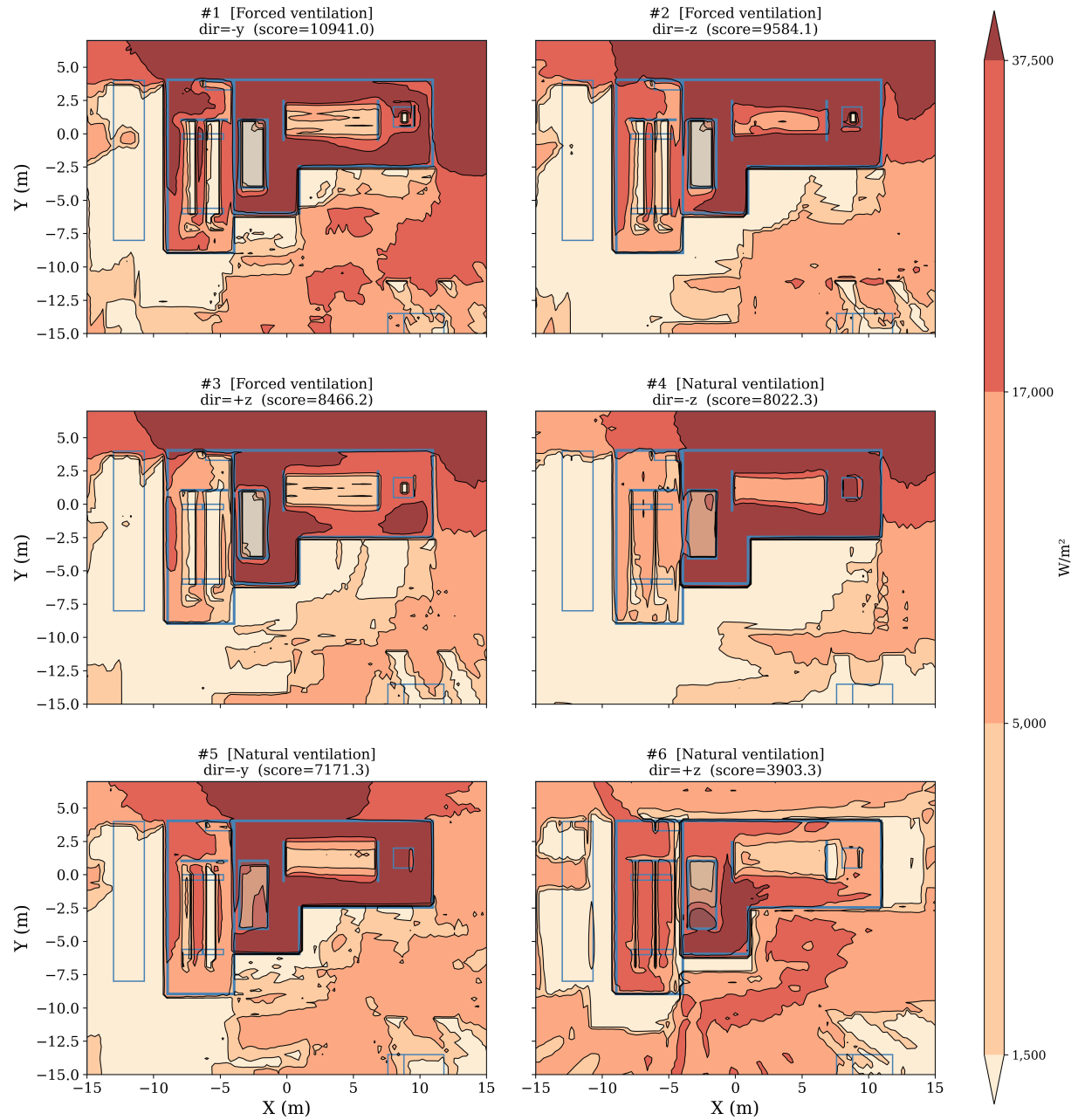


Figure A.27: Radiation in steady state for 430bar, full bore cases.

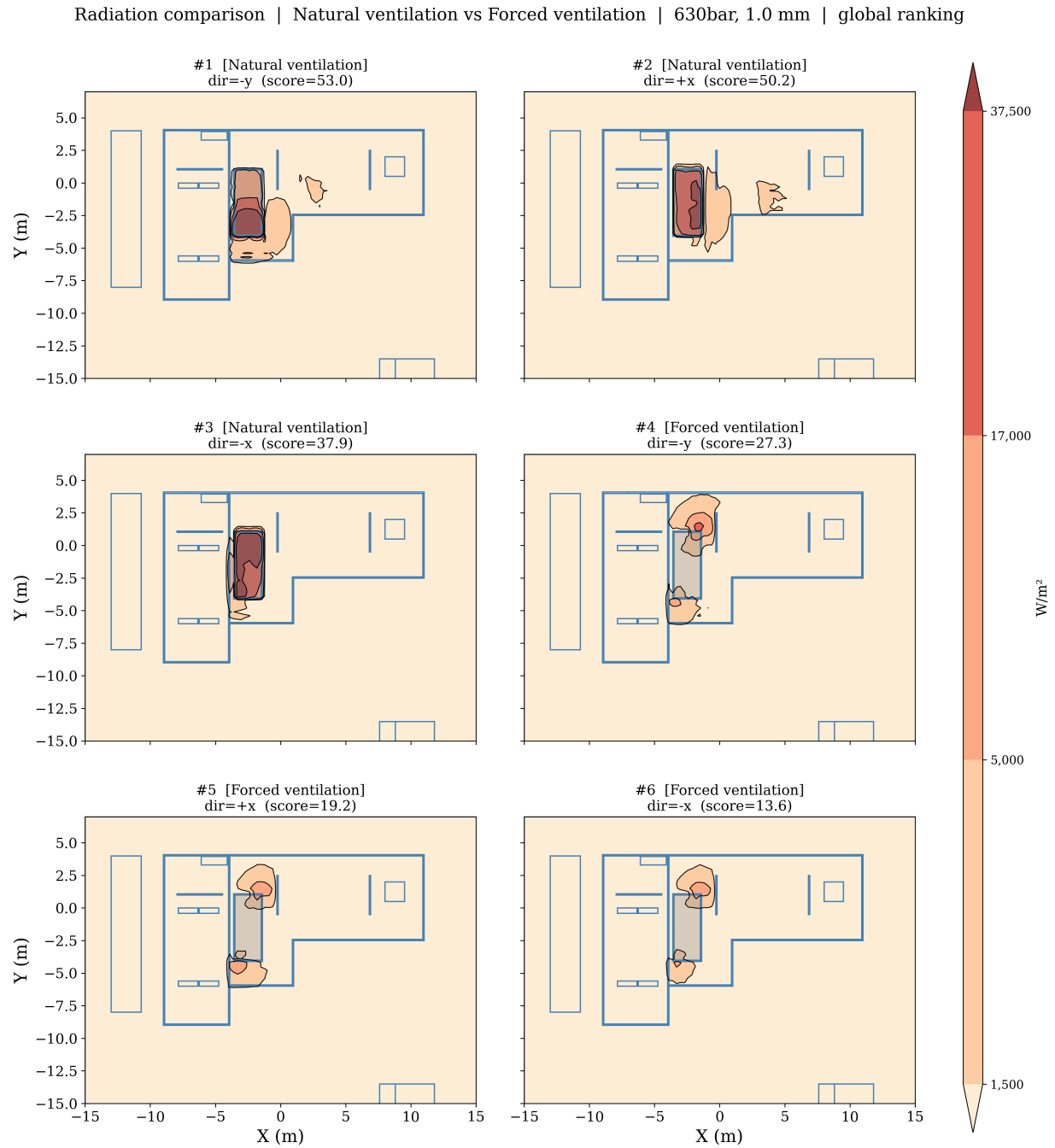
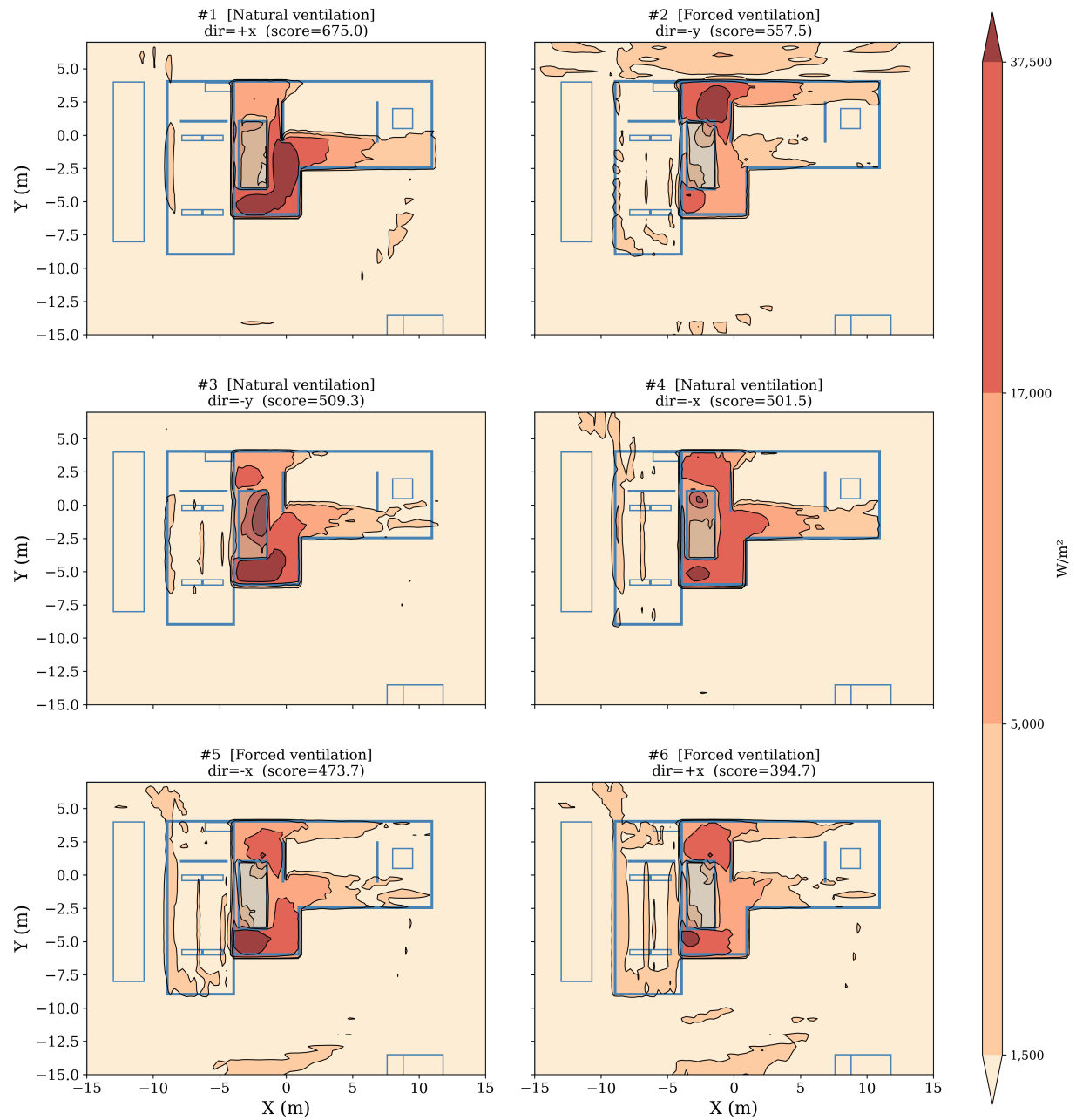
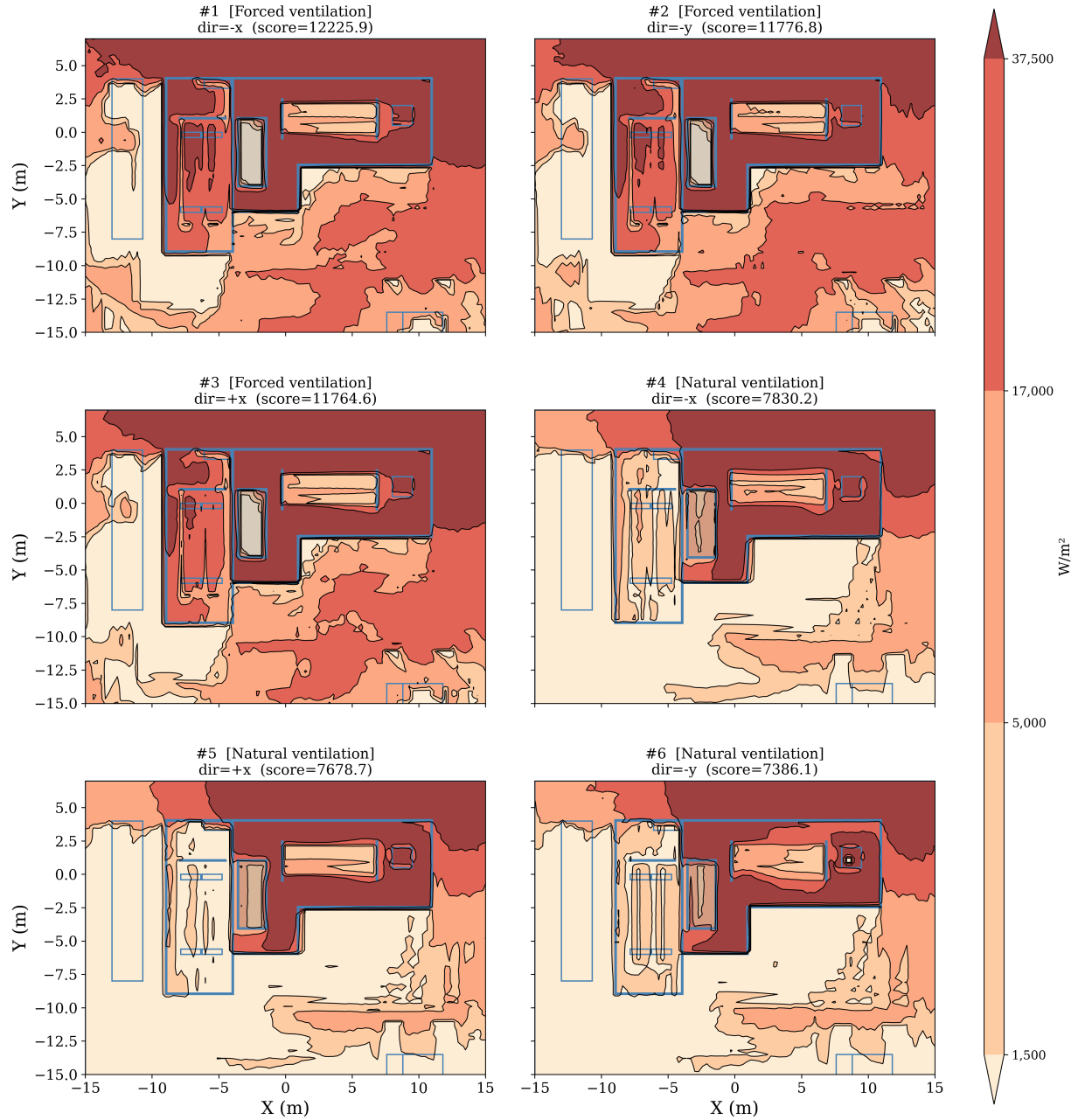


Figure A.28: Radiation in steady state for 630bar, 1% area cases.

Radiation comparison | Natural ventilation vs Forced ventilation | 630bar, 3.2 mm | global ranking

**Figure A.29:** Radiation in steady state for 630bar, 10% area cases.

Radiation comparison | Natural ventilation vs Forced ventilation | 630bar, 10.0 mm | global ranking

**Figure A.30:** Radiation in steady state for 630bar, full bore cases.

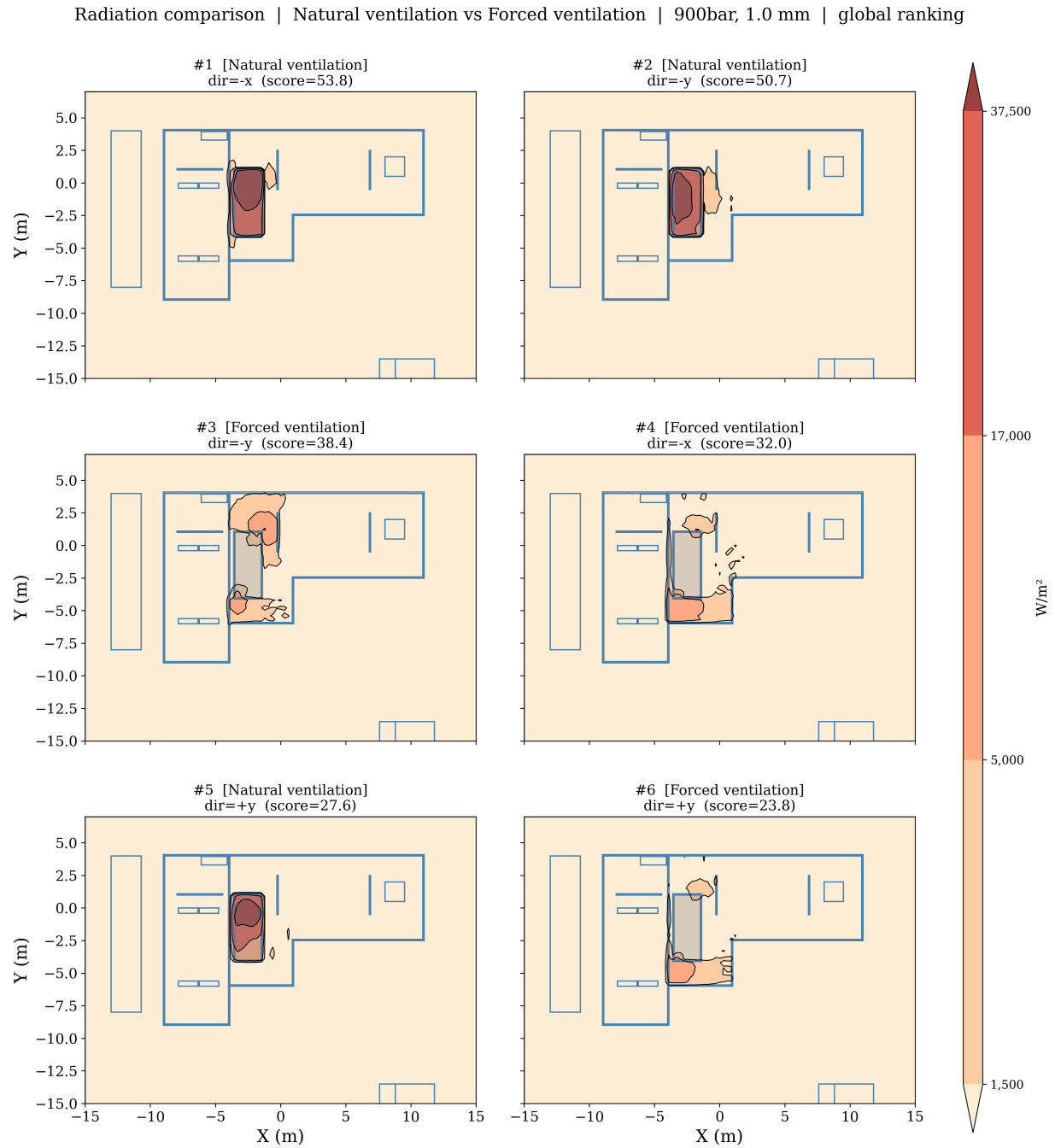


Figure A.31: Radiation in steady state for 900bar, 1% area cases.

Radiation comparison | Natural ventilation vs Forced ventilation | 900bar, 2.5 mm | global ranking

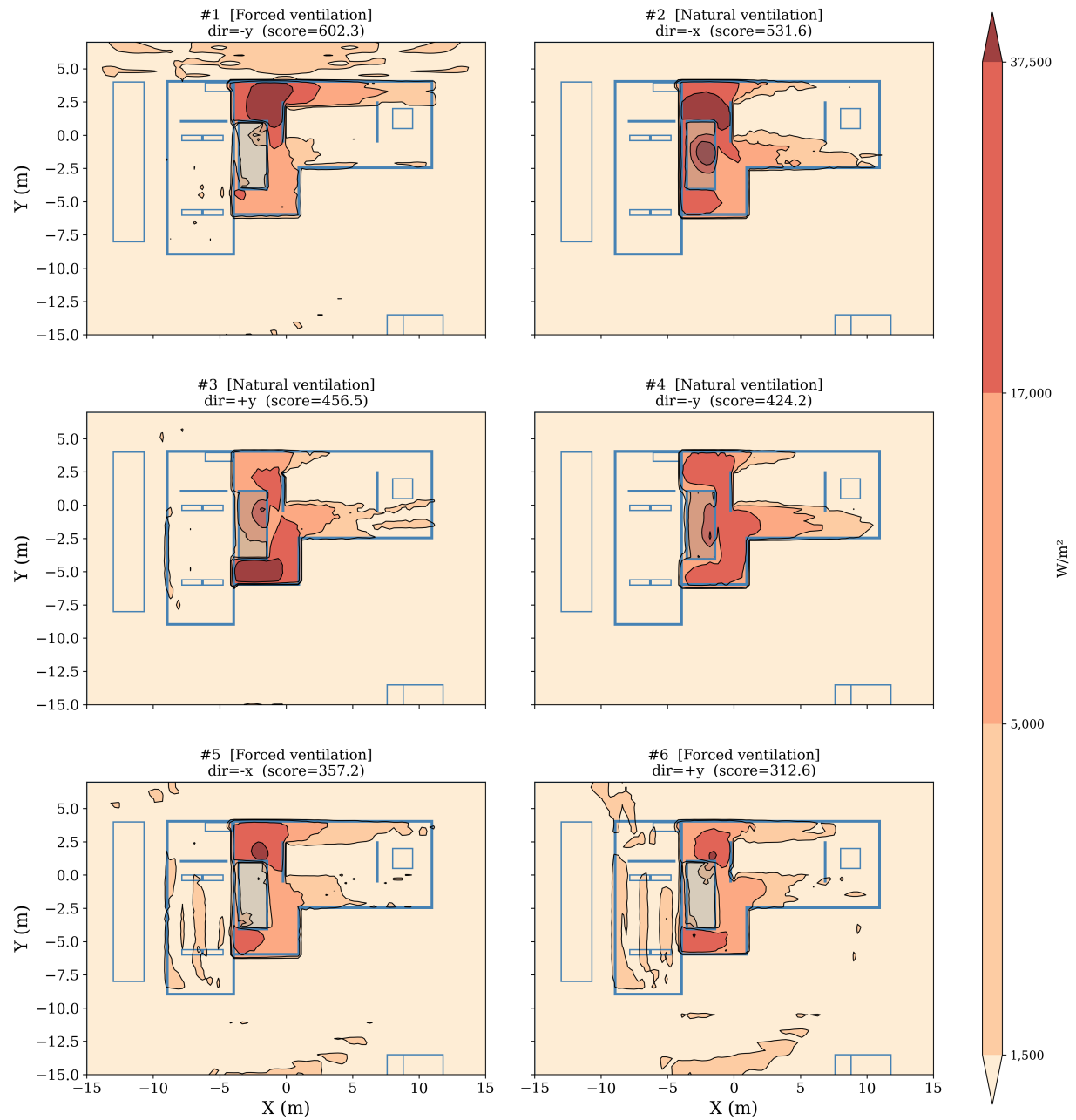


Figure A.32: Radiation in steady state for 900bar, 10% area cases.

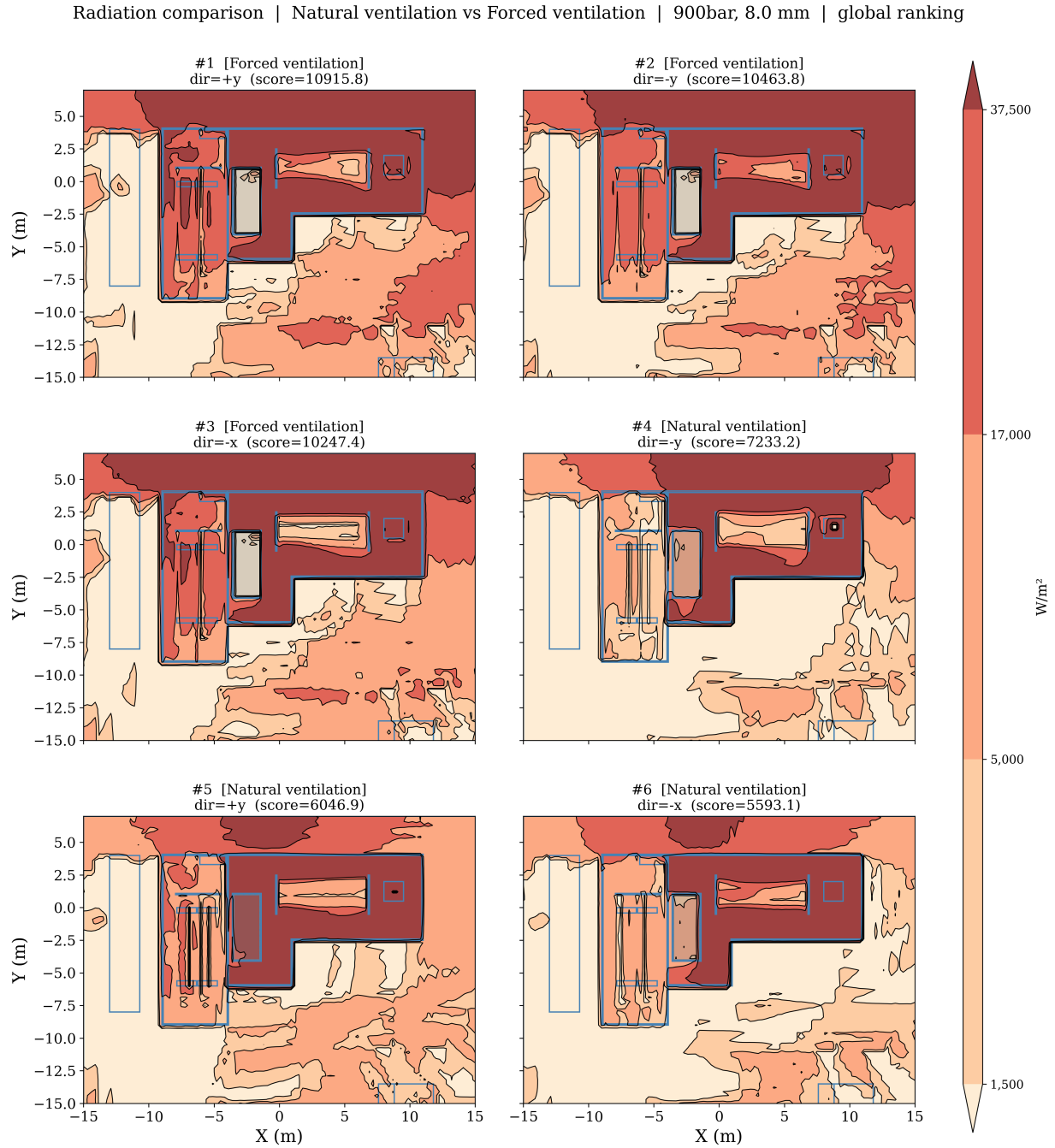


Figure A.33: Radiation in steady state for 900bar, full bore cases.

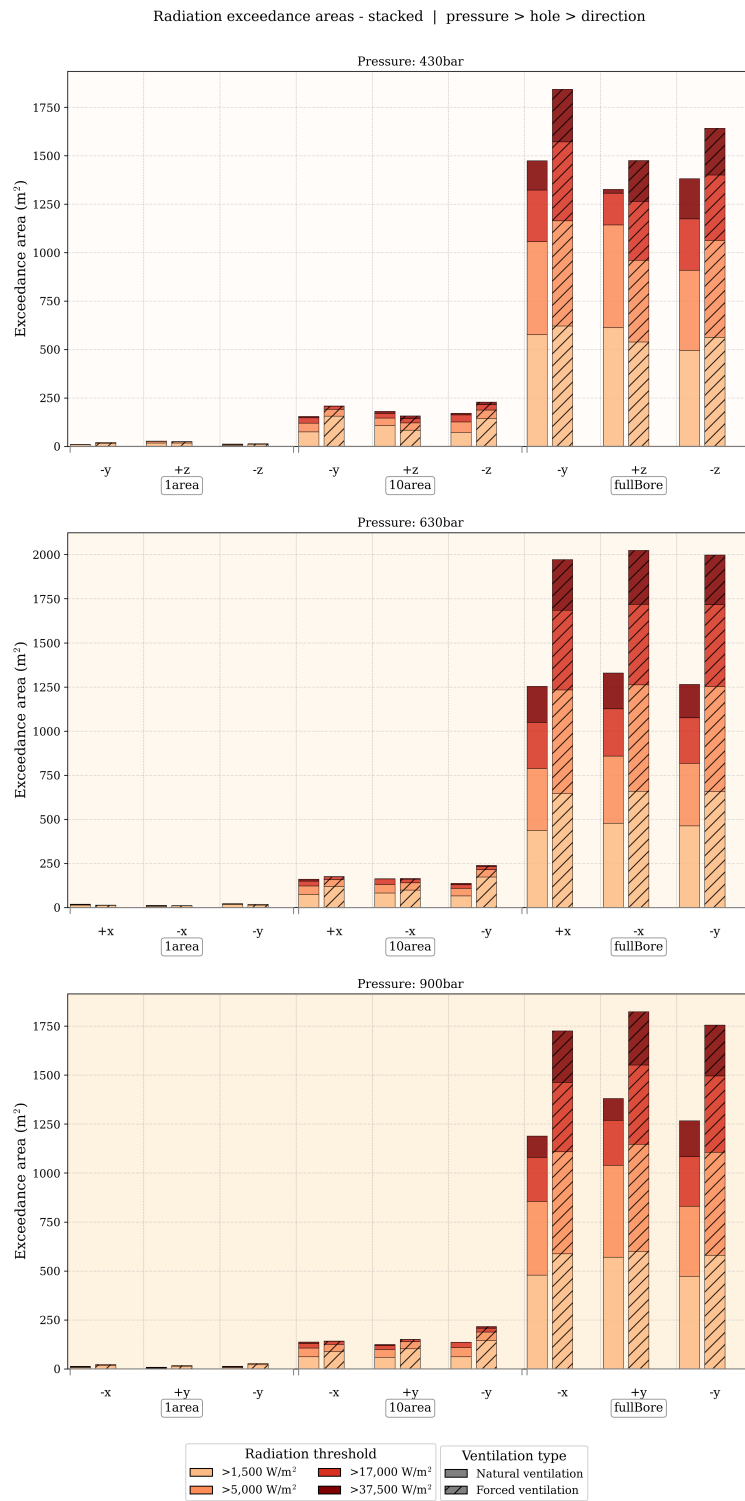


Figure A.34: Summary of the radiation scores at steady state for all the cases organised by a hierarchy of pressure, then hole size, then direction.

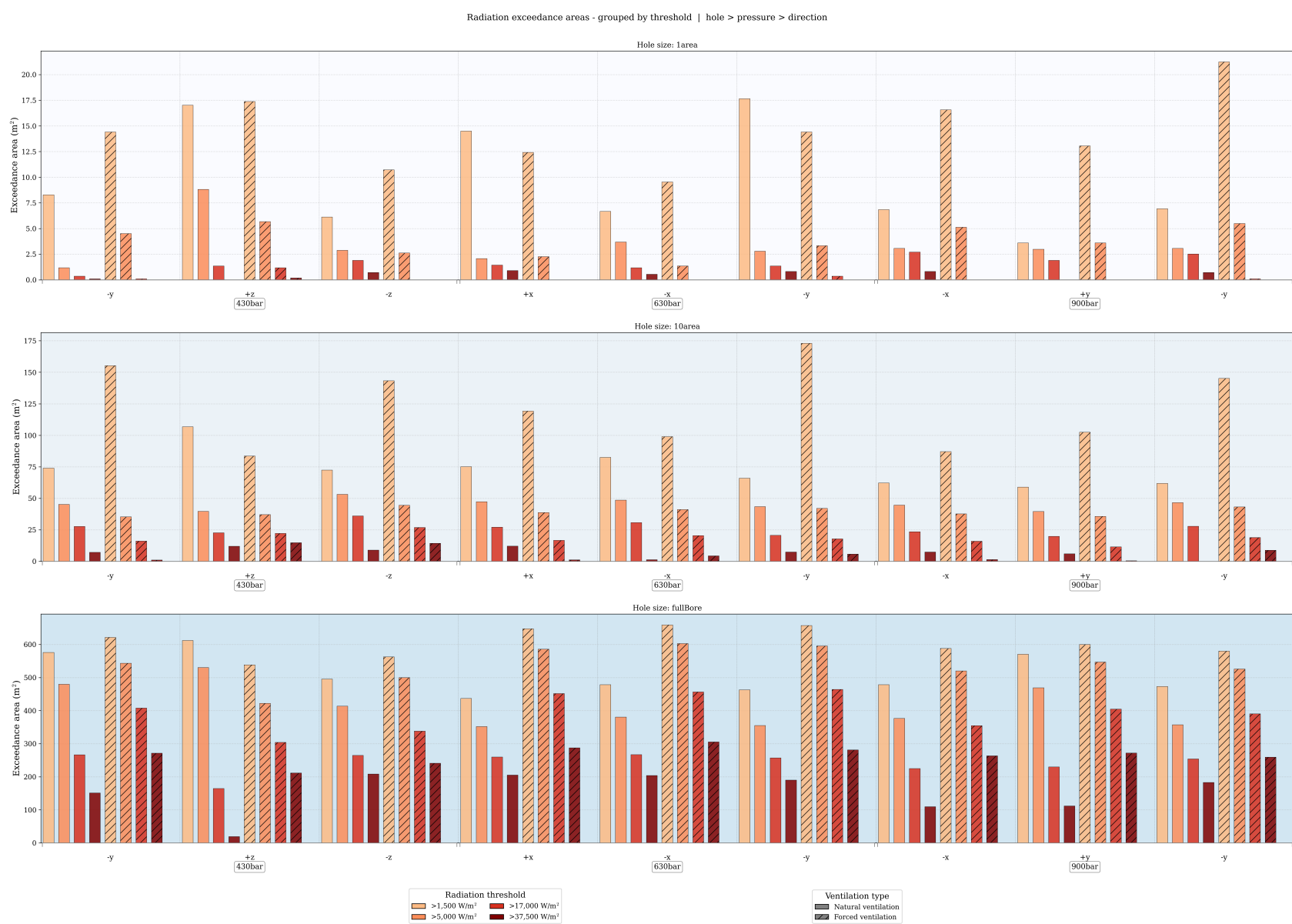


Figure A.35: Summary of the radiation scores at steady state for all the cases organised by a hierarchy of hole size, then pressure, then direction.



Figure A.36: Summary of the radiation scores at steady state for all the cases organised by a hierarchy of pressure, then hole size, then direction.

A.3 Consolidated analysis

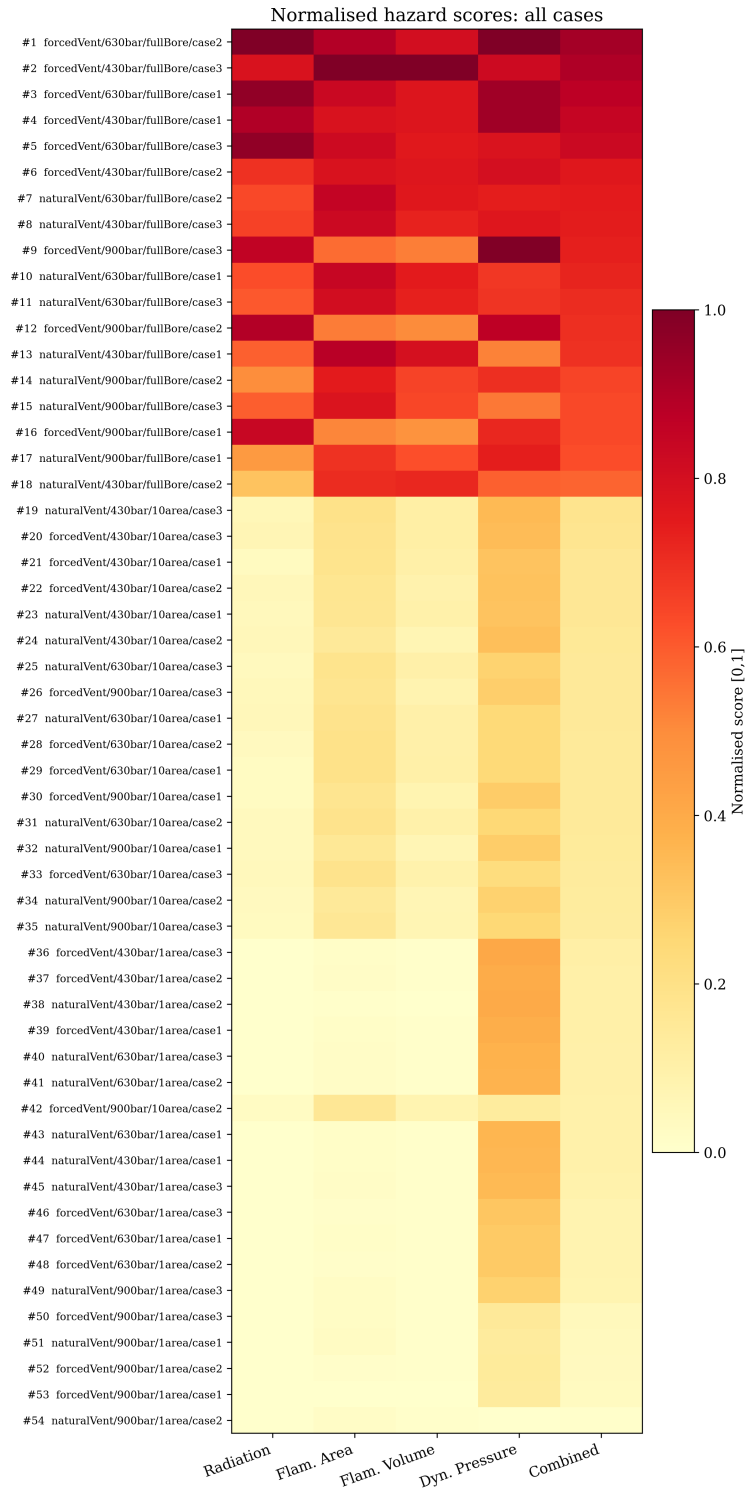


Figure A.37: Heatmap with all the cases of the study.

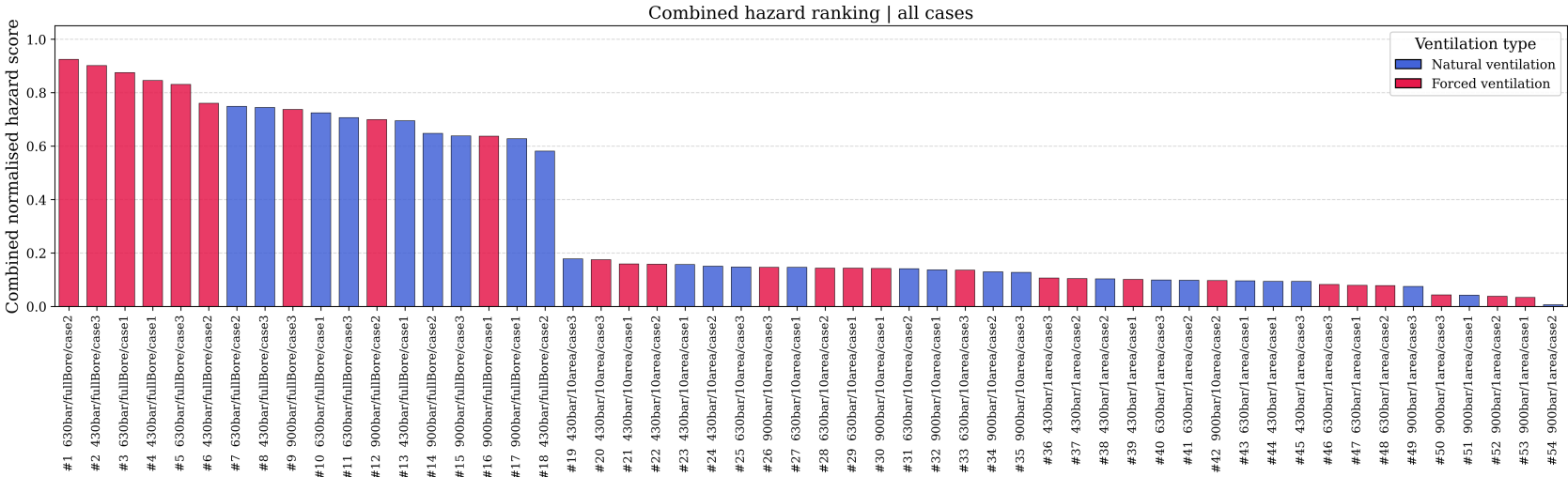


Figure A.38: Raking on the cases based on their combined score.

Appendix B

Python scripts

B.1 Sensitivity studies

B.1.1 Sensitivity analysis

```
"""
sensAn-variousCAses_v2.py    H2 leak sensitivity: location and direction
=====
Compares flammable cloud volume, total H2 inventory, and peak
    ↪ concentration
across multiple leak locations and directions over time.

Each study path corresponds to one leak location (e.g. loc A, B, C).
Within each study, caseN subfolders correspond to different leak
    ↪ directions.

Features
-----
- Configurable number of studies and cases per study
- Per-study skip lists for missing cases
- Log scale or broken y-axis for metrics where one location dominates
  (controlled by USE_LOG_SCALE boolean)
- Three-level legend: colour = direction, line style = location
"""

from pathlib import Path
import glob

import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
```



```

from kfxutils import r3d

# =====
# CONFIGURATION
# =====

LFL = 4.0    # lower flammability limit (vol %)
UFL = 75.0   # upper flammability limit (vol %)

# --- Study definitions
# Each entry: (path, label, line_style)
# Add or comment out entries as needed.
STUDIES = [
    (
        "/proj/P10475_RA/STUDENTS/2026/Nur/simulations/naturalVent/sensAn/d"
        ↪ "irSens-locA",
        "(-2.5, 0.4, 0.7)",
        "-",
    ),
    (
        "/proj/P10475_RA/STUDENTS/2026/Nur/simulations/naturalVent/sensAn/d"
        ↪ "irSens-locB",
        "(-2.5, -3.75, 0.7)",
        "--",
    ),
    (
        "/proj/P10475_RA/STUDENTS/2026/Nur/simulations/naturalVent/630bar/f"
        ↪ "ullBore/compBetweenLocs",
        "(-2.5, -1.7, 1.0)",
        "-.",
    ),
]

# Maximum number of cases across all studies
N_CASES = 6

# Per-study skip lists - format: {study_path: [case_numbers_to_skip]}
# Case numbers are 1-based (matching caseN folder names).
SKIP_CASES = {

```

```

    "/proj/P10475_RA/STUDENTS/2026/Nur/simulations/naturalVent/630bar/fullB_
    ↪ ore/compBetweenLocs":
    ↪ [5],
    # add more studies here if needed:
    # "/proj/.../heightSens-locA": [3],
}

# Case labels - one per case number (same across all studies)
#CASE_LABELS = ["z = 0.2m", "z = 0.7m", "z = 1.2m", "z = 1.7m", "z =
    ↪ 2.2m"]
CASE_LABELS = ['-Y', '+Y', '+X', '-X', '-Z', '+Z']

# --- Y-axis mode for volume/inventory plots
    ↪ -----
# True ? logarithmic y-axis (handles large dynamic range honestly)
# False ? broken y-axis (visually cleaner, shows detail at low values)
USE_LOG_SCALE = 0

# Broken axis split point - only used when USE_LOG_SCALE = False.
# Values below BREAK_LOWER are shown in the bottom panel,
# values above BREAK_UPPER are shown in the top panel.
BREAK_LOWER_VOL = 40.0      # m³ - upper limit of bottom (detail) panel
BREAK_UPPER_VOL = 2250.0    # m³ - lower limit of top (overview) panel
BREAK_LOWER_INVENT = 15.0
BREAK_UPPER_INVENT = 370.0

# --- Plot settings
    ↪ -----
PLOT_TITLE = "Hydrogen leak: effect of leak direction over time"
OUTPUT_FILE = ("/proj/P10475_RA/STUDENTS/2026/Nur/img/plots/dirSens/"
               "leak_dir_comparison_3locs.png")

savePlot = 1
showPlot = 0

# =====
    ↪ =====
# STYLING
# =====
    ↪ =====

# One colour per case (direction), consistent across all studies
COLORS = [f"C{i}" for i in range(N_CASES)]
MARKERS = ["o", "s", "^", "D", "v", "P"]

```

```

mpl.rcParams.update({
    "text.usetex":      False,
    "font.family":      "serif",
    "mathtext.fontset": "cm",
    "font.size":        14,
    "axes.titlesize":   18,
    "axes.labelsize":   16,
    "xtick.labelsize":  13,
    "ytick.labelsize":  13,
    "legend.fontsize":  11,
    "figure.dpi":       300,
    "savefig.dpi":      300,
    "lines.linewidth":  1.0,
})

# =====
# DATA EXTRACTION
# =====

def extract_metrics(folder):
    """
    Load all case_std_0*.r3d snapshots in folder and compute per-snapshot:
    - flamm_vol : volume of flammable gas (LFL = conc = UFL) in m3
    - inventory : total H2 volume in domain (m3)
    - peak      : max H2 concentration (vol %)
    Returns dict of numpy arrays, or None if no files found.
    """
    files = sorted(glob.glob(f"{folder}/case_std_0*.r3d"))
    if not files:
        return None

    times, flamm_vols, inventories, peaks = [], [], [], []
    for fpath in files:
        r = r3d.read(fpath)
        r.to_type2()

        fg_idx = r.fieldnames.index("Fuel gas (vol %)")
        fg      = r.fielfddata[fg_idx]
        cv      = r.compute_cell_volume()

        times.append(r.time)

```

```

    flamm_vols.append((((fg >= LFL) & (fg <= UFL)) * cv).sum())
    inventories.append((fg / 100.0 * cv).sum())
    peaks.append(float(fg.max()))

return {
    "time":      np.array(times),
    "flamm_vol": np.array(flamm_vols),
    "inventory": np.array(inventories),
    "peak":      np.array(peaks),
}

def load_all_results():
    """
    Load metrics for all studies and cases, applying per-study skip lists.
    Returns nested dict: results[study_path][case_n] = metrics_dict or
    ↪ None.
    """
    results = {}
    for study_path, study_label, _ in STUDIES:
        results[study_path] = {}
        skip = SKIP_CASES.get(study_path, []) # cases to skip for this
        ↪ study

        for case_n in range(1, N_CASES + 1):
            if case_n in skip:
                print(f" SKIPPED: {study_path}/case{case_n} (in skip
                ↪ list)")
                results[study_path][case_n] = None
                continue

            folder = Path(study_path) / f"case{case_n}"
            if not folder.exists():
                print(f" SKIPPED: {folder} (not found)")
                results[study_path][case_n] = None
                continue

            print(f" Loading: {folder}")
            data = extract_metrics(str(folder))
            if data is None:
                print(f" No r3d files found in {folder}")
            results[study_path][case_n] = data

    return results

```

```

# =====
↳ =====
# LEGEND BUILDERS
# =====
↳ =====

def _case_legend_handles(n_cases):
    """One coloured handle per case (direction/height)."""
    return [
        Line2D([0], [0],
               color=COLORS[i],
               marker=MARKERS[i % len(MARKERS)],
               linestyle="-", markersize=5,
               label=CASE_LABELS[i] if i < len(CASE_LABELS) else
               ↳ f"case{i+1}"),
        for i in range(n_cases)    # always N_CASES entries
    ]

def _study_legend_handles():
    """One line-style handle per study (location)."""
    return [
        Line2D([0], [0], color="k", linestyle=ls, linewidth=1.5,
               ↳ label=label)
        for _, label, ls in STUDIES
    ]

# =====
↳ =====
# PLOT HELPERS
# =====
↳ =====

def _plot_lines_on_ax(ax, results, metric_key):
    """
    Draw all time series for one metric on a single axes.
    Skipped/missing cases are silently omitted.
    """
    for i in range(N_CASES):
        case_n = i + 1
        for study_path, _, linestyle in STUDIES:
            data = results[study_path].get(case_n)
            if data is None:
                continue

```

```

        ax.plot(data["time"], data[metric_key],
                color=COLORS[i],
                linestyle=linestyle,
                marker=MARKERS[i % len(MARKERS)],
                markersize=4,
                markevery=max(1, len(data["time"]) // 8),
                linewidth=1.2)

def _add_legends(ax):
    case_handles = _case_legend_handles(N_CASES)
    style_handles = _study_legend_handles()
    leg1 = ax.legend(handles=case_handles, loc="upper left",
                    bbox_to_anchor=(0, 1), fontsize=9,
                    title="Direction", framealpha=0.9)
    ax.add_artist(leg1)
    ax.legend(handles=style_handles, loc="upper left",
            bbox_to_anchor=(0.13, 1), fontsize=9,
            title="Location", framealpha=0.9)

# =====
# BROKEN Y-AXIS HELPER
# =====

def _make_broken_axes(fig, gridspec_slot):
    """
    Create two vertically stacked axes within gridspec_slot that simulate
    a broken y-axis. Returns (ax_top, ax_bottom).
    The caller is responsible for setting ylims and hiding the shared
    spine.
    """
    gs_inner = gridspec_slot.subgridspec(2, 1, hspace=0.05,
                                         height_ratios=[1, 2])
    ax_top = fig.add_subplot(gs_inner[0])
    ax_bot = fig.add_subplot(gs_inner[1], sharex=ax_top)
    return ax_top, ax_bot

def _style_broken_axes(ax_top, ax_bot, ylabel):
    """
    Apply the standard broken-axis styling:
    - hide inner spines

```

```

- add diagonal tick marks at the break
- y-label centred between the two panels
"""
ax_top.spines["bottom"].set_visible(False)
ax_bot.spines["top"].set_visible(False)
ax_top.tick_params(labelbottom=False, bottom=False)

# diagonal break markers
d = 0.008
kwargs = dict(transform=ax_top.transAxes, color="k",
               clip_on=False, linewidth=0.8)
ax_top.plot((-d, +d), (-d, +d), **kwargs)
ax_top.plot((1-d, 1+d), (-d, +d), **kwargs)
kwargs.update(transform=ax_bot.transAxes)
ax_bot.plot((-d, +d), (1-d, 1+d), **kwargs)
ax_bot.plot((1-d, 1+d), (1-d, 1+d), **kwargs)

# shared y-label placed between panels
ax_bot.set_ylabel(ylabel, labelpad=10)

# =====
↳ =====
# MAIN PLOT
# =====
↳ =====

def make_plot(results):
    """
    Build the 3-subplot figure.
    Subplots 1 & 2 (flamm_vol, inventory): log or broken axis depending on
    USE_LOG_SCALE. Subplot 3 (peak conc): standard linear axis always.
    """
    metrics = [
        ("flamm_vol", "Flammable cloud volume (m³)", "Flammable Cloud
        ↳ Volume"),
        ("inventory", r"Total H$2$ inventory (m³)", "Total Fuel Gas
        ↳ Inventory"),
        ("peak", "Peak concentration (vol %)", "Peak Concentration"),
    ]

    if USE_LOG_SCALE:
        fig, axes = plt.subplots(3, 1, figsize=(10, 13), sharex=True)
        fig.subplots_adjust(hspace=0.15, top=0.92, bottom=0.07,
                           left=0.12, right=0.97)

```

```

for ax, (key, ylabel, title) in zip(axes, metrics):
    _plot_lines_on_ax(ax, results, key)
    _add_legends(ax)
    ax.set_ylabel(ylabel)
    ax.set_title(title)
    ax.grid(True, alpha=0.3, which="both")
    if key in ("flamm_vol", "inventory"):
        ax.set_yscale("log")
        ax.set_ylabel(ylabel + " [log scale]")

axes[0].set_xlim(left=0)
axes[-1].set_xlabel("Time (s)")

else:
    from matplotlib.gridspec import GridSpec, GridSpecFromSubplotSpec

    # --- per-metric broken axis limits ---
    # set independently for each volume metric
    BREAK_CONFIG = {
        "flamm_vol": {
            "bot_ylim": (0, BREAK_LOWER_VOL),          # detail
            ↪ panel
            "top_ylim": (BREAK_UPPER_VOL, None),        # overview
            ↪ panel (None = auto)
            "top_ratio": 1,    # relative height of overview panel
            "bot_ratio": 2,    # relative height of detail panel
        },
        "inventory": {
            "bot_ylim": (0, BREAK_LOWER_INVENT),
            "top_ylim": (BREAK_UPPER_INVENT, None),
            "top_ratio": 1,
            "bot_ratio": 2,
        },
    }

    # outer GridSpec: 3 rows - each row height proportional to content
    # volume rows are taller because they're split into two panels
    fig = plt.figure(figsize=(10, 15))
    gs = GridSpec(3, 1, figure=fig,
                  height_ratios=[3, 3, 2],
                  hspace=0.19,
                  top=0.92, bottom=0.07,
                  left=0.12, right=0.97)

```



```

all_axes = []    # collect bottom-most ax per row for sharex
first_ax = None

for row, (key, ylabel, title) in enumerate(metrics):
    if key in BREAK_CONFIG:
        cfg = BREAK_CONFIG[key]

        # split this row into two panels (top=overview,
        ↪ bot=detail)
        gs_inner = GridSpecFromSubplotSpec(
            2, 1,
            subplot_spec=gs[row],
            hspace=0.1,
            height_ratios=[cfg["top_ratio"], cfg["bot_ratio"]],
        )
        ax_top = fig.add_subplot(gs_inner[0],
                                sharex=first_ax)
        ax_bot = fig.add_subplot(gs_inner[1],
                                sharex=ax_top)

        if first_ax is None:
            first_ax = ax_top

        # draw on both panels
        _plot_lines_on_ax(ax_top, results, key)
        _plot_lines_on_ax(ax_bot, results, key)

        # set y limits
        ax_top.set_ylim(cfg["top_ylim"])
        ax_bot.set_ylim(cfg["bot_ylim"])

        # broken axis spine styling
        ax_top.spines["bottom"].set_visible(False)
        ax_bot.spines["top"].set_visible(False)
        ax_top.tick_params(labelbottom=False, bottom=False)
        # also suppress on ax_bot of each broken pair:
        ax_bot.tick_params(labelbottom=False)

        # replace the 4 diagonal markers with just the left-side
        ↪ ones:
        d = 0.012
        kw = dict(color="k", clip_on=False, linewidth=0.8,
                  transform=ax_top.transAxes)

```

```

ax_top.plot((-d, +d), (-d*2, +d*2), **kw)    # left side
    ↪ only
kw["transform"] = ax_bot.transAxes
ax_bot.plot((-d, +d), (1-d*2, 1+d*2), **kw)  # left side
    ↪ only

# y-label: centred across both panels using a shared
    ↪ invisible
# axis that spans the full row height
ax_span = fig.add_subplot(gs[row], frameon=False)
ax_span.tick_params(labelleft=False, left=False,
                    labelbottom=False, bottom=False)
ax_span.set_ylabel(ylabel, labelpad=40)
ax_span.patch.set_visible(False)
for sp in ax_span.spines.values():
    sp.set_visible(False)

# grids and titles
ax_top.set_title(title)
ax_top.grid(True, alpha=0.3)
ax_bot.grid(True, alpha=0.3)
ax_bot.set_xlim(left=0)

# legend on the overview panel
_add_legends(ax_top)
all_axes.append(ax_bot)

else:
    # peak concentration - plain linear axis
    ax = fig.add_subplot(gs[row], sharex=first_ax)
    _plot_lines_on_ax(ax, results, key)
    _add_legends(ax)
    ax.set_ylabel(ylabel)
    ax.set_title(title)
    ax.set_xlabel("Time (s)")
    ax.set_xlim(left=0)
    ax.spines["right"].set_visible(False)
    ax.grid(True, alpha=0.3)
    all_axes.append(ax)

fig.suptitle(PLOT_TITLE, fontsize=13, fontweight="bold", y=0.97)

if savePlot:
    Path(OUTPUT_FILE).parent.mkdir(parents=True, exist_ok=True)

```

```
plt.savefig(OUTPUT_FILE, dpi=150, bbox_inches="tight")
print(f"Saved ? {OUTPUT_FILE}")
if showPlot:
    plt.show()
plt.close(fig)

# =====
↳ =====
# MAIN
# =====
↳ =====

def main():
    print("\n" + "="*55)
    print("  Loading simulation results")
    print("="*55)
    results = load_all_results()

    print("\n" + "="*55)
    print("  Generating plot")
    print("="*55)
    make_plot(results)

if __name__ == "__main__":
    main()
```

B.1.2 Ignition probability

```
import matplotlib.pyplot as plt

#x_column = time
#y_column = Prob

for i in range(1, 8):
    file = f"/proj/P10475_RA/STUDENTS/2026/Nur/test/case{i}-ignition/case_i_
        ↪ gnprob.log"

    with open(file, "r") as f:
        header_line = f.readline()
        data_lines = f.readlines()

    headers = [h.strip().strip('"') for h in header_line.split(",")]
    x_index = 0
    y_index = 4

    last_row = data_lines[-1].split()

    #print(header_line) # print column names
    print(f"case{i} ->", last_row[y_index])

    # Lists to store data
    x_values = []
    y_values = []

    # Loop over all data rows
    for line in data_lines:
        parts = [p.strip() for p in line.split(',')] # split by
            ↪ whitespace. if other character, ','
        x_values.append(float(parts[x_index]))
        y_values.append(float(parts[y_index]))

    # Plot this case
    plt.plot(x_values, y_values, label=f"case{i}")

# Add labels and legend
plt.xlabel(x_column)
plt.ylabel(y_column)
plt.title(f"{y_column.strip()} vs {x_column}")
plt.legend()
plt.show()
```

B.2 Flammable area and volume

B.2.1 Area

```

"""
compare_cases_flammable_area.py - KFX flammable cloud area comparison
=====
Computes the XY-projected area exposed to flammable gas (LFL-UFL) for
a set of KFX simulation cases and produces:

    1. A time-series plot of projected flammable area over time
       (Option B only - requires multiple r3d snapshots per case)
    2. A shaded-area floorplan overlay showing the flammable footprint
       at a chosen snapshot (both Option A and Option B)
    3. A console table of projected areas per case / snapshot

Folder modes
-----
Option A - explicit list of case folders (single snapshot only):
    CASE_FOLDERS = ["/path/to/case1", "/path/to/case2", ...]
    PARENT_FOLDERS_A / PARENT_FOLDERS_B = []

Option B - parent folder(s) with case1/case2/.../caseN subfolders:
    PARENT_FOLDERS_A = ["/path/to/parent"]
    CASE_FOLDERS = []

Scenario comparison
-----
COMPARE_TWO_SCENARIOS = True ? overlay Scenario A (solid) and B (dashed)
                                on the time-series plot; floorplan
                                ↪ subplots
                                are ranked globally as in radiation
                                ↪ script.
COMPARE_TWO_SCENARIOS = False ? single-scenario mode.

Plot booleans
-----
PLOT_TIMESERIES - time-series line plot (Option B only)
PLOT_FLOORPLAN  - shaded floorplan overlay subplots
"""

import os
from pathlib import Path

```

```

import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import matplotlib.colors as mcolors
import numpy as np
from matplotlib.gridspec import GridSpec
from matplotlib.lines import Line2D

from kfxutils import r3d

# =====
# CONFIGURATION
# =====

LFL = 4.0          # lower flammability limit (vol %)
UFL = 75.0         # upper flammability limit (vol %)
LEAK_POS = (-2.5, -3.75) # (x, y) of leak source in metres - no z
    ↪ needed

# --- Comparison mode
    ↪ -----
COMPARE_TWO_SCENARIOS = 1 # True ? overlay two ventilation scenarios

# --- Plot toggles
    ↪ -----
PLOT_TIMESERIES = 1      # time-series plot (Option B only)
PLOT_FLOORPLAN = 1      # shaded floorplan subplots

# --- Case info
    ↪ -----
pressure = ["430bar", "630bar", "900bar"]
press_i = 2
holeCat = ["1area", "10area", "fullBore"]
hole_i = 1

num_cases = 3 # number of caseN subfolders for Scenario A
num_cases_B = 3 # number of caseN subfolders for Scenario B

# --- Axis / display limits
    ↪ -----
x_lim = (20, 80) # time axis limits for the time-series plot
    ↪ (s)

```

```

floorplan_xlim = (-15,15)    # in general, (-15,15)
floorplan_ylim = (-15,7)     # in general, (-15,7)
# if these limits are increased too much, the subplots don't fit and
  ↳ require further tempering

SNAPSHOT_INDEX = -1          # index into r3d_files_* to use for floorplan
  ↳ (-1 = last)

# --- Floor plan overlay
  ↳ -----
FLOORPLAN_JSON = "floorplan.json" # path to floorplan JSON; set to None
  ↳ to skip

# --- Plot saving
  ↳ -----
savePlot = 1
showPlot = 0

# =====
  ↳ =====
# OPTION A - explicit folder list (single snapshot, no time series)
# only used if CASE_FOLDERS is non empty!
# =====
  ↳ =====
CASE_FOLDERS = [
    # "/proj/.../case1",
    # "/proj/.../case2",
    # "/proj/P10475_RA/STUDENTS/2026/Nur/simulations/wallHeightStudy/1m",
    # "/proj/P10475_RA/STUDENTS/2026/Nur/simulations/wallHeightStudy/2m",
    # "/proj/P10475_RA/STUDENTS/2026/Nur/simulations/wallHeightStudy/3m",
    # "/proj/P10475_RA/STUDENTS/2026/Nur/simulations/forcedVent/430bar-fire"
    ↳ /fullBore/4m",
    # "/proj/P10475_RA/STUDENTS/2026/Nur/simulations/wallHeightStudy/5m",
    # "/proj/P10475_RA/STUDENTS/2026/Nur/simulations/wallHeightStudy/6m",
    # "/proj/P10475_RA/STUDENTS/2026/Nur/simulations/wallHeightStudy/7m",
    # "/proj/P10475_RA/STUDENTS/2026/Nur/simulations/wallHeightStudy/8m"
]
# one r3d filename per folder in CASE_FOLDERS (matched by position)
CASE_FILES_A = [
    # "case_std_exit.r3d",
    # "case_std_exit.r3d",
]

```

```

# =====
↳ =====
# SCENARIO A (Option B - always active)
# =====
↳ =====
scenario_A_label = "Natural ventilation"

parent_folders_A = [
    f"/proj/P10475_RA/STUDENTS/2026/Nur/simulations/naturalVent/{pressure[p_
        ↳ res_i]}/{holeCat[hole_i]}",
]
explicit_folders_A = [] # alternative to parent_folders_A; leave empty
↳ if using parent

# r3d snapshots for Scenario A - same filenames expected in every caseN
↳ subfolder
r3d_files_A = [
    "case_std_00030049.r3d",
    "case_std_00060049.r3d",
    "case_std_00090000.r3d",
    "case_std_00120002.r3d",
    "case_std_exit.r3d",
]

match press_i:
    case 0: labels_A = ["-y", "+z", "-z"]
    case 1: labels_A = ["+x", "-x", "-y"]
    case 2: labels_A = ["-x", "+y", "-y"]

# =====
↳ =====
# SCENARIO B (Option B - only used when COMPARE_TWO_SCENARIOS = True)
# =====
↳ =====
scenario_B_label = "Forced ventilation"

parent_folders_B = [
    f"/proj/P10475_RA/STUDENTS/2026/Nur/simulations/forcedVent/{pressure[pr_
        ↳ ess_i]}/{holeCat[hole_i]}",
]
explicit_folders_B = []

# r3d snapshots for Scenario B - filenames may differ from Scenario A
r3d_files_B = [

```



```

        "case_std_00020004.r3d",
        "case_std_00040004.r3d",
        "case_std_00060003.r3d",
        "case_std_00080000.r3d",
        "case_std_exit.r3d",
    ]

    match press_i:
        case 0: labels_B = ["-y", "+z", "-z"]
        case 1: labels_B = ["+x", "-x", "-y"]
        case 2: labels_B = ["-x", "+y", "-y"]

# =====
# END OF CONFIGURATION
# =====

# --- Hole diameter lookup (used in plot titles)
# -----
match (press_i, hole_i):
    case (0, 0): diam = 1.2
    case (0, 1): diam = 3.8
    case (0, 2): diam = 12.0
    case (1, 0): diam = 1.0
    case (1, 1): diam = 3.2
    case (1, 2): diam = 10.0
    case (2, 0): diam = 1.0
    case (2, 1): diam = 2.5
    case (2, 2): diam = 8.0

# --- Save paths
# -----
_save_dir =
    f"/proj/P10475_RA/STUDENTS/2026/Nur/img/plots/naturalVentVsForcedVent"
_tag = f"{pressure[press_i]}_{holeCat[hole_i]}"
#_save_dir =
    f"/proj/P10475_RA/STUDENTS/2026/Nur/img/plots/wallHeightStudy"
#_tag = f"wallHeight"
save_path_ts = os.path.join(_save_dir, f"flamArea_timeseries_{_tag}.png")
save_path_fp = os.path.join(_save_dir, f"flamArea_floorplan_{_tag}.png")
save_path_csv_ts = os.path.join(_save_dir,
    f"flamArea_timeseries_{_tag}.csv") # set to None to skip

```

```

save_path_csv_fp = os.path.join(_save_dir,
    ↪ f"flamArea_floorplan_{_tag}.csv")          # set to None to skip

# --- Styling
    ↪ -----
colors = ["#e6194b", "#3cb44b", "#4363d8", "#f58231", "#911eb4", "#42d4f4",
    ↪ "#f032e6"]

FLAM_COLOR    = "#3cb44b"    # fill colour for flammable footprint patches
GEOM_EDGE     = "steelblue"
FLOORPLAN_EC = "steelblue" # edge colour for floorplan geometry rectangles

mpl.rcParams.update({
    "text.usetex":      False,
    "font.family":      "serif",
    "mathtext.fontset": "cm",
    "font.size":        14,
    "axes.titlesize":   16,
    "axes.labelsize":   15,
    "xtick.labelsize":  13,
    "ytick.labelsize":  13,
    "legend.fontsize":  13,
    "figure.dpi":       150,
    "savefig.dpi":      300,
})

# =====
    ↪ =====
# HELPERS
# =====
    ↪ =====

def build_cases_option_b(parent_folders, explicit_folders, labels, n_cases):
    """Return list of (label, folder_path) for Option B (parent/caseN
    ↪ layout)."""
    cases = []
    if parent_folders:
        for parent in parent_folders:
            for i in range(1, n_cases + 1):          # iterate case1
                ↪ caseN
                folder = Path(parent) / f"case{i}"
                idx    = len(cases)
                label  = labels[idx] if idx < len(labels) else folder.name
                cases.append((label, folder))

```

```

else:                                     # fall back to
    ↪ explicit list
    for idx, folder in enumerate(explicit_folders):
        label = labels[idx] if idx < len(labels) else Path(folder).name
        cases.append((label, Path(folder)))
return cases

def build_cases_option_a():
    """Return list of (label, folder_path, r3d_file) for Option A
    ↪ (explicit folders)."""
    cases = []
    for idx, folder in enumerate(CASE_FOLDERS):
        r3d_file = CASE_FILES_A[idx] if idx < len(CASE_FILES_A) else
            ↪ "case_std_exit.r3d"
        label = Path(folder).name # use folder name as label if none
            ↪ provided
        cases.append((label, Path(folder), r3d_file))
    return cases

def load_r3d(filepath):
    """Load an r3d file. Returns (gas_field, xc, yc, xs, ys, time)."""
    r = r3d.read(str(filepath))
    if r.r3d_type == 4:                  # type 4 files need conversion before
        ↪ reading fields
        r.to_type2()

    fieldnames = [f.strip() for f in r.fieldnames]
    try:
        gas_idx = fieldnames.index("Fuel gas (vol %)")
    except ValueError:
        print(f" Available fields: {fieldnames}")
        raise ValueError(f"'Fuel gas (vol %)' not found in {filepath}")

    gas = r.fielddata[gas_idx].astype(float) # shape: (nx, ny, nz)
    return (
        gas,
        np.asarray(r.xc), np.asarray(r.yc), # cell-centre coordinates
        np.asarray(r.xs), np.asarray(r.ys), # staggered (face)
            ↪ coordinates
        float(r.time), # simulation time in seconds
    )

```

```

def compute_cell_areas(xs, ys, nx, ny):
    """Compute XY cell areas (m2) from staggered face coordinates."""
    dx = np.zeros(nx)
    dy = np.zeros(ny)
    dx[1:] = xs[1:] - xs[:-1]    # width of each cell from face positions
    dy[1:] = ys[1:] - ys[:-1]
    return np.outer(dx, dy)      # (nx, ny) array of cell areas


def projected_flammable_area(gas, xs, ys):
    """
    Project flammable cells (LFL-UFL) onto the XY plane.
    A column (x,y) counts as flammable if ANY z-layer is within LFL-UFL.
    Returns (total_area_m2, projected_bool_mask).
    """
    nx, ny, _ = gas.shape
    cell_area = compute_cell_areas(xs, ys, nx, ny)
    flam_mask = (gas >= LFL) & (gas <= UFL)    # True where gas is
        ↪ flammable, shape (nx,ny,nz)
    proj_mask = np.any(flam_mask, axis=2)      # collapse z: True if
        ↪ any layer flammable
    area = float(np.sum(cell_area[proj_mask])) # sum areas of
        ↪ flammable columns
    return area, proj_mask


def max_distance_from_leak(proj_mask, xc, yc, leak_pos):
    """
    Find the flammable cell centroid furthest from the leak source in XY.
    Returns (max_dist_m, x_far, y_far).
    """
    ix_list, iy_list = np.where(proj_mask)    # indices of all flammable
        ↪ columns
    if len(ix_list) == 0:                     # no flammable gas - return
        ↪ zero distance
        return 0.0, leak_pos[0], leak_pos[1]

    fx = xc[ix_list]    # x-coordinates of flammable cell centres
    fy = yc[iy_list]    # y-coordinates of flammable cell centres
    dists = np.sqrt((fx - leak_pos[0])**2 + (fy - leak_pos[1])**2)    # 2D
        ↪ distance, no z
    i_max = np.argmax(dists)
    return float(dists[i_max]), float(fx[i_max]), float(fy[i_max])

```

```

def load_floorplan():
    import json
    if FLOORPLAN_JSON is None:
        return []
    p = Path(FLOORPLAN_JSON)
    if not p.exists():
        print(f"WARNING: {FLOORPLAN_JSON} not found - floor plan will not
              ↳ be drawn.")
        return []
    with open(p) as f:
        return json.load(f)

def draw_floorplan(ax, rects):
    """Draw floorplan geometry as unfilled rectangles on ax."""
    for r in rects:
        patch = mpatches.Rectangle(
            (r["x0"], r["y0"]), r["x1"] - r["x0"], r["y1"] - r["y0"],
            linewidth=1.2, edgecolor=FLOORPLAN_EC, facecolor="none",
            ↳ zorder=2,
        )
        ax.add_patch(patch)

# =====
# ↳ =====
# DATA LOADING
# =====
# ↳ =====

def load_timeseries(cases, r3d_files):
    """
    Load every r3d snapshot for each case (Option B).
    Returns list of dicts: {label, times, areas, snapshots}
    where snapshots is a list of (proj_mask, xc, yc) - one per timestep.
    """
    results = []
    for label, folder in cases:
        print(f"\n Case: {label} ({folder})")
        times, areas, snapshots = [], [], []
        for fname in r3d_files:
            fpath = folder / fname
            if not fpath.exists():

```

```

        print(f"    SKIPPED (not found): {fname}")
        continue
    try:
        gas, xc, yc, xs, ys, t = load_r3d(fpath)
    except Exception as e:
        print(f"    SKIPPED ({e}): {fname}")
        continue
    area, proj_mask = projected_flammable_area(gas, xs, ys)
    times.append(t)
    areas.append(area)
    snapshots.append((proj_mask, xc, yc))    # store mask + grid
        → for later plotting
    print(f"    {fname}  t={t:.1f}s  area={area:.2f} m2")
    results.append({
        "label":    label,
        "times":    times,
        "areas":    areas,
        "snapshots": snapshots,
    })
return results

def load_snapshot(cases, r3d_files, snapshot_index):
    """
    Load a single r3d snapshot (by index into r3d_files) for each case.
    Returns list of dicts: {label, time, area, proj_mask, xc, yc}
    """
    fname = r3d_files[snapshot_index]    # resolve index (e.g. -1 ? exit
        → file)
    results = []
    for label, folder in cases:
        fpath = folder / fname
        if not fpath.exists():
            print(f"    SKIPPED (not found): {fpath}")
            continue
        gas, xc, yc, xs, ys, t = load_r3d(fpath)
        area, proj_mask = projected_flammable_area(gas, xs, ys)
        results.append({
            "label":    label,
            "time":     t,
            "area":     area,
            "proj_mask": proj_mask,
            "xc":       xc,
            "yc":       yc,

```

```

    })
    return results

def load_snapshot_option_a():
    cases_a = build_cases_option_a()
    results = []
    for label, folder, fname in cases_a:
        fpath = folder / fname
        if not fpath.exists():
            print(f" SKIPPED (not found): {fpath}")
            continue
        try:
            gas, xc, yc, xs, ys, t = load_r3d(fpath)
        except Exception as e:
            print(f" SKIPPED (load error in {fpath}): {e}")
            continue
        area, proj_mask = projected_flammable_area(gas, xs, ys)
        results.append({
            "label":    label,
            "time":     t,
            "area":     area,
            "proj_mask": proj_mask,
            "xc":       xc,
            "yc":       yc,
        })
    return results

# =====
# PRINT TABLE
# =====

def print_area_table(results, scenario_label=""):
    """Print a formatted table of case / time / area to stdout."""
    header = f"{'Case':<20} {'Time (s)':>10} {'Area (m²)':>12}"
    sep = "=" * len(header)
    if scenario_label:
        print(f"\n{sep}\n {scenario_label}\n{sep}")
    else:
        print(f"\n{sep}")
    print(header)
    print("-" * len(header))

```

```

for r in results:
    if "times" in r:                                # time-series
        ↪ result: one row per timestep
        for t, a in zip(r["times"], r["areas"]):
            print(f" {r['label']:<18} {t:>10.1f} {a:>12.4f}")
        else:                                       # single-snapshot
            ↪ result: one row per case
            print(f" {r['label']:<18} {r['time']:>10.1f}
                  {r['area']:>12.4f}")
print(sep)

def save_area_csv(results_and_labels, filepath):
    """
    Save area results to CSV.
    results_and_labels: list of (results, scenario_label) tuples.
    Works for any number of scenarios, time-series or snapshots.
    """
    import csv
    rows = []
    for results, scenario_label in results_and_labels:
        for r in results:
            if "times" in r:                         # time-series: one row
                ↪ per timestep
                for t, a in zip(r["times"], r["areas"]):
                    rows.append({
                        "scenario": scenario_label,
                        "label":    r["label"],
                        "time_s":   t,
                        "area_m2": a,
                    })
            else:                                     # snapshot: one row per
                ↪ case
                rows.append({
                    "scenario": scenario_label,
                    "label":    r["label"],
                    "time_s":   r["time"],
                    "area_m2":  r["area"],
                })

    if not rows:
        print(" No results to save.")
        return

    filepath = Path(filepath)

```



```

    filepath.parent.mkdir(parents=True, exist_ok=True)
    with open(filepath, "w", newline="") as f:
        writer = csv.DictWriter(f, fieldnames=rows[0].keys())
        writer.writeheader()
        writer.writerows(rows)
    print(f"CSV saved in {filepath}")
# =====
# TIME-SERIES PLOT
# =====

def plot_timeseries(results_A, results_B=None):
    fig, ax = plt.subplots(figsize=(10, 6))

    scenario_styles = ["-", "--"]    # solid = Scenario A, dashed =
    # Scenario B

    # --- plot Scenario A ---
    for idx, res in enumerate(results_A):
        ax.plot(res["times"], res["areas"],
                color=colors[idx % len(colors)],
                linewidth=2, linestyle=scenario_styles[0])

    # --- plot Scenario B (if comparing) ---
    if results_B is not None:
        for idx, res in enumerate(results_B):
            ax.plot(res["times"], res["areas"],
                    color=colors[idx % len(colors)],
                    linewidth=2, linestyle=scenario_styles[1])

    # --- build legend handles ---
    case_labels = [r["label"] for r in results_A]
    case_handles = [
        # one coloured line
        # per leak direction
        Line2D([0], [0], color=colors[i % len(colors)], linewidth=2,
               linestyle="-", label=case_labels[i])
        for i in range(len(results_A))
    ]

    if results_B is not None:
        style_handles = [
            # one handle per
            # ventilation scenario
            Line2D([0], [0], color="k", linewidth=2, linestyle=ls, label=sl)

```

```

        for sl, ls in zip([scenario_A_label, scenario_B_label],
                          ↪ scenario_styles)
    ]
    # two separate legends so colours and line styles don't mix
    legend1 = ax.legend(handles=case_handles,
                        loc="lower left", bbox_to_anchor=(0, 0.21),
                        title="Leak direction", framealpha=0.9)
    ax.add_artist(legend1) # keep legend1 when adding legend2
    ax.legend(handles=style_handles,
              loc="lower left", bbox_to_anchor=(0, 0),
              title="Ventilation type", framealpha=0.9)
else:
    ax.legend(handles=case_handles, title="Leak direction",
              ↪ framealpha=0.9)

ax.set_xlim(x_lim)
ax.set_xlabel(r"Time $t$ (s)")
ax.set_ylabel(r"Projected flammable area (m$^2$)")
title_scen = (f"{scenario_A_label} vs {scenario_B_label}"
              if results_B is not None else scenario_A_label)
ax.set_title(
    f"Flammable cloud projected area | {title_scen}\n"
    f"{pressure[press_i]}, d={diam} mm"
)
ax.grid(True, linestyle="--", alpha=0.5)
plt.tight_layout()

if savePlot:
    plt.savefig(save_path_ts, dpi=300, bbox_inches="tight")
    print(f"\nSaved in {save_path_ts}")
if showPlot:
    plt.show()
plt.close(fig)

# =====
↪ =====
# FLOORPLAN / SHADED AREA PLOT
# =====
↪ =====

def _render_floorplan_subplot(ax, res, title_str, floorplan, leak_pos):
    """
    Draw the flammable footprint, max-extent line, and leak marker on ax.
    Returns max_dist (m) for reference.

```

```

"""
proj_mask = res["proj_mask"]
xc         = res["xc"]
yc         = res["yc"]

# --- cell widths from centre coordinates (midpoint rule) ---
def midpoint_widths(c):
    w = np.zeros(len(c))
    w[0]    = c[1] - c[0]
    w[-1]   = c[-1] - c[-2]
    w[1:-1] = (c[2:] - c[:-2]) / 2.0
    return w

dx = midpoint_widths(xc)
dy = midpoint_widths(yc)

# --- build one Rectangle patch per flammable column, then add as a
#     collection ---
from matplotlib.collections import PatchCollection
patches = []
ix_list, iy_list = np.where(proj_mask)           # indices of
#     flammable columns
for ix, iy in zip(ix_list, iy_list):
    x0 = xc[ix] - dx[ix] / 2                     # lower-left corner of
    #     cell
    y0 = yc[iy] - dy[iy] / 2
    patches.append(mpatches.Rectangle((x0, y0), dx[ix], dy[iy]))

if patches:
    pc = PatchCollection(patches, facecolor=FLAM_COLOR,
                        edgecolor="none", alpha=0.5, zorder=1)
    ax.add_collection(pc)                         # faster than adding
    #     patches one by one

# --- max distance line from leak to furthest flammable cell ---
max_dist, x_far, y_far = max_distance_from_leak(proj_mask, xc, yc,
#     leak_pos)
ax.plot(*leak_pos, marker="x", color="red", markersize=8,
#     linestyle='None',
        markeredgewidth=2, zorder=6, label="Leak source")  # x marker
#     at leak origin
ax.plot([leak_pos[0], x_far], [leak_pos[1], y_far],
        color="red", linewidth=1.5, linestyle="-", zorder=5,
        label=f"Max extent: {max_dist:.1f} m")

```

```

ax.legend(loc="lower left", fontsize=11, framealpha=0.85)    # picks up
    ↪ both red handles

draw_floorplan(ax, floorplan)
ax.set_xlim(floorplan_xlim)
ax.set_ylim(floorplan_ylim)
ax.set_aspect("equal")
ax.set_title(title_str, fontsize=13)

return max_dist


def plot_floorplan(snap_results, floorplan, scenario_label="",
    ↪ extra_tag=""):
    """
    One subplot per case, shared axis labels on edges only.
    snap_results: list of {label, time, area, proj_mask, xc, yc}
    """
    n      = len(snap_results)
    ncols  = min(n, 2)
    nrows  = int(np.ceil(n / ncols))

    fig = plt.figure(figsize=(7 * ncols, 6 * nrows))
    gs  = GridSpec(nrows, ncols, figure=fig, wspace=0.12, hspace=0.05)

    for idx, res in enumerate(snap_results):
        row, col = divmod(idx, ncols)
        ax = fig.add_subplot(gs[row, col])
        title_str = (
            f"{res['label']} | t={res['time']:.1f} s\n"
            f"area = {res['area']:.2f} m2"
        )
        _render_floorplan_subplot(ax, res, title_str, floorplan, LEAK_POS)
        ax.set_ylabel("Y (m)" if col == 0 else "")
        ax.set_xlabel("X (m)" if row == nrows - 1 else "")
        if col != 0:
            ax.set_yticklabels([])
        if row != nrows - 1:
            ax.set_xticklabels([])

    for idx in range(n, nrows * ncols):
        ↪ slots                                # hide unused subplot
        row, col = divmod(idx, ncols)

```

```

fig.add_subplot(gs[row, col]).set_visible(False)

scen_str = f" | {scenario_label}" if scenario_label else ""
fig.suptitle(
    f"Flammable cloud footprint{scen_str} | "
    f"{pressure[press_i]}, d={diam} mm",
    fontsize=15, y=.91,
)

fname = save_path_fp.replace(".png", f"{extra_tag}.png")
if savePlot:
    fig.savefig(fname, dpi=300, bbox_inches="tight")
    print(f"Saved in {fname}")
if showPlot:
    plt.show()
plt.close(fig)

def plot_floorplan_comparison(snap_A, snap_B, floorplan):
    """
    Global ranking by area across both scenarios, row-major layout, 2
    ↪ columns.
    Rank 1 (largest area) is top-left, reading order left-to-right,
    ↪ top-to-bottom.
    """
    for r in snap_A:
        r["scenario"] = scenario_A_label # tag each result with its
        ↪ scenario name
    for r in snap_B:
        r["scenario"] = scenario_B_label

    combined = sorted(snap_A + snap_B, key=lambda r: r["area"],
        ↪ reverse=True)
    n = len(combined)
    ncols = 2
    nrows = int(np.ceil(n / ncols))

    fig = plt.figure(figsize=(14, 6 * nrows))
    gs = GridSpec(nrows, ncols, figure=fig, wspace=0.12, hspace=0.05)

    for rank, res in enumerate(combined, 1):
        row, col = divmod(rank - 1, ncols) # row-major: rank
        ↪ 1?(0,0), 2?(0,1), 3?(1,0)
        ax = fig.add_subplot(gs[row, col])

```

```

    title_str = (
        f"#{rank}  [{res['scenario']}]  {res['label']}\n"
        f"t={res['time']:.1f} s  |  area={res['area']:.2f} m²"
    )
    _render_floorplan_subplot(ax, res, title_str, floorplan, LEAK_POS)
    ax.set_ylabel("Y (m)" if col == 0 else "")
    ax.set_xlabel("X (m)" if row == nrows - 1 else "")
    if col != 0:
        ax.set_yticklabels([])
    if row != nrows - 1:
        ax.set_xticklabels([])

for idx in range(n, nrows * ncols):
    # hide unused subplot
    ↪ slots
    row, col = divmod(idx, ncols)
    fig.add_subplot(gs[row, col]).set_visible(False)

fig.suptitle(
    f"Flammable cloud footprint  |  {scenario_A_label} vs
    ↪ {scenario_B_label}"
    f"  |  {pressure[press_i]}, d={diam} mm  |  global ranking by area",
    fontsize=15, y=.92,
)
if savePlot:
    fig.savefig(save_path_fp, dpi=300, bbox_inches="tight")
    print(f"Saved in {save_path_fp}")
if showPlot:
    plt.show()
plt.close(fig)

# =====
↪ =====
# MAIN
# =====
↪ =====

def main():
    floorplan = load_floorplan()
    option_a = bool(CASE_FOLDERS)  # True if user filled in Option A
    ↪ explicit list

# -----
↪ -----
# OPTION A - single snapshot, no time series

```

```

# -----
↪ -----
if option_a:
    print("\n=== Option A mode (explicit folders, single snapshot) ===")
    snap_results = load_snapshot_option_a()
    print_area_table(snap_results, scenario_label="Option A cases")
    save_area_csv([(snap_results, "")], save_path_csv_fp)

    if PLOT_TIMESERIES:
        print("NOTE: PLOT_TIMESERIES is not available in Option A mode
↪ - skipped.")
    if PLOT_FLOORPLAN:
        plot_floorplan(snap_results, floorplan)
    return

# -----
↪ -----
# OPTION B - time series + floorplan
# -----
↪ -----
cases_A = build_cases_option_b(parent_folders_A, explicit_folders_A,
                                labels_A, num_cases)

if not COMPARE_TWO_SCENARIOS:
    # --- single scenario ---
    print(f"\n=== {scenario_A_label} ===")
    ts_A = load_timeseries(cases_A, r3d_files_A)
    print_area_table(ts_A, scenario_label=scenario_A_label)
    save_area_csv([(ts_A, scenario_A_label)], save_path_csv_ts)

    if PLOT_TIMESERIES:
        plot_timeseries(ts_A)

    if PLOT_FLOORPLAN:
        snap_A = load_snapshot(cases_A, r3d_files_A, SNAPSHOT_INDEX)
        plot_floorplan(snap_A, floorplan,
                        scenario_label=scenario_A_label,
                        extra_tag=f"_{scenario_A_label.replace(' ',
↪ ' ')}")

else:
    # --- two scenarios ---
    cases_B = build_cases_option_b(parent_folders_B, explicit_folders_B,
                                    labels_B, num_cases_B)

```

```
print(f"\n=== {scenario_A_label} ===")
ts_A = load_timeseries(cases_A, r3d_files_A)
print_area_table(ts_A, scenario_label=scenario_A_label)

print(f"\n=== {scenario_B_label} ===")
ts_B = load_timeseries(cases_B, r3d_files_B)
print_area_table(ts_B, scenario_label=scenario_B_label)
save_area_csv([(ts_A, scenario_A_label),
               (ts_B, scenario_B_label)], save_path_csv_ts)

if PLOT_TIMESERIES:
    plot_timeseries(ts_A, ts_B)

if PLOT_FLOORPLAN:
    snap_A = load_snapshot(cases_A, r3d_files_A, SNAPSHOT_INDEX)
    snap_B = load_snapshot(cases_B, r3d_files_B, SNAPSHOT_INDEX)
    plot_floorplan_comparison(snap_A, snap_B, floorplan)
    save_area_csv([(snap_A, scenario_A_label),
                  (snap_B, scenario_B_label)], save_path_csv_fp)

if __name__ == "__main__":
    main()
```


B.2.2 Volume

```

import matplotlib.pyplot as plt
import math
import matplotlib as mpl
import os
from matplotlib.lines import Line2D

# =====
# CONFIGURATION - edit this section each time you use the script
# =====

# --- Comparison mode
# Set to True to compare two scenarios (e.g. natural vs forced
# ventilation)
# Set to False for single-scenario mode (original behaviour)
COMPARE_TWO_SCENARIOS = 1

# --- General parameters

x_column = " Time "
y_column = " Volume between LFL and UFL "
filename = "case_gascloud.out"

# case info
pressure = ["430bar", "630bar", "900bar"]
press_i = 2
holeCat = ["1area", "10area", "fullBore"]
hole_i = 1
savePlot = 1
showPlot = 0

num_cases = 3
x_lim = (0, 80)

# Spot-check times for the comparison table
z_values = list(range(10, 71, 10))
reference_case = 1

```

```

# =====
↳ =====
# SCENARIO A (always used)
# =====
↳ =====
scenario_A_label = "Natural ventilation"

parent_folders_A = [
    f"/proj/P10475_RA/STUDENTS/2026/Nur/simulations/naturalVent/{pressure[p_
        ↳ res_i]}/{holeCat[hole_i]}",
]
explicit_folders_A = []

match press_i:
    case 0:
        labels_A = ["-y", "+z", "-z"]
    case 1:
        labels_A = ["+x", "-x", "-y"]
    case 2:
        labels_A = ["-x", "+y", "-y"]

# =====
↳ =====
# SCENARIO B (only used when COMPARE_TWO_SCENARIOS = True)
# =====
↳ =====
scenario_B_label = "Forced ventilation"

parent_folders_B = [
    f"/proj/P10475_RA/STUDENTS/2026/Nur/simulations/forcedVent/{pressure[pr_
        ↳ ess_i]}/{holeCat[hole_i]}",
]
explicit_folders_B = []

# Labels for scenario B cases (must match num_cases_B below)
match press_i:
    case 0:
        labels_B = ["-y", "+z", "-z"]
    case 1:
        labels_B = ["+x", "-x", "-y"]
    case 2:
        labels_B = ["-x", "+y", "-y"]

num_cases_B = 3    # can differ from num_cases if needed

```

```

# =====
# END OF CONFIGURATION
# =====

# --- Hole diameter for plot title
-----
match (press_i, hole_i):
    case (0,0): diam = 1.2
    case (0,1): diam = 3.8
    case (0,2): diam = 12.0
    case (1,0): diam = 1.0
    case (1,1): diam = 3.2
    case (1,2): diam = 10.0
    case (2,0): diam = 1.0
    case (2,1): diam = 2.5
    case (2,2): diam = 8.0

# --- Styling
-----
colors = ["#e6194b", "#3cb44b", "#4363d8", "#f58231", "#911eb4", "#42d4f4",
         "#f032e6"]

mpl.rcParams.update({
    "text.usetex":      False,
    "font.family":      "serif",
    "mathtext.fontset": "cm",
    "font.size":        14,
    "axes.titlesize":   18,
    "axes.labelsize":   16,
    "xtick.labelsize":  13,
    "ytick.labelsize":  13,
    "legend.fontsize":  13,
    "figure.dpi":       150,
})

x_label = r"Time $t$ (s)"
y_label = r"Volume between LFL and UFL (m$^3$)"
save_path = f"/proj/P10475_RA/STUDENTS/2026/Nur/img/plots/naturalVentVsForc_
            edVent/gasCloud_comp_{pressure[press_i]}_{holeCat[hole_i]}.png"

```

```

# =====
# HELPERS
# =====

def build_cases(parent_folders, explicit_folders, labels, num_cases,
    filename):
    """Return a list of (label, filepath) pairs."""
    cases = []
    if parent_folders:
        for parent in parent_folders:
            for i in range(1, num_cases + 1):
                folder = os.path.join(parent, f"case{i}")
                filepath = os.path.join(folder, filename)
                idx = len(cases)
                label = labels[idx] if idx < len(labels) else
                    f"{os.path.basename(parent)}/case{i}"
                cases.append((label, filepath))
    else:
        for idx, folder in enumerate(explicit_folders):
            filepath = os.path.join(folder, filename)
            label = labels[idx] if idx < len(labels) else
                os.path.basename(folder)
            cases.append((label, filepath))
    return cases

def read_cases(cases, z_values):
    """Read data for a list of cases. Returns table_results and
    plot_results dicts."""
    table_results = {}
    plot_results = {}
    for case_idx, (label, filepath) in enumerate(cases):
        with open(filepath, "r") as f:
            header_line = f.readline()
            data_lines = f.readlines()

            headers = [h.strip().strip('\'') for h in header_line.split(";")]
            x_index = headers.index(x_column)
            y_index = headers.index(y_column)

            table_row = {}
            x_full, y_full = [], []

```

```

    for line in data_lines[1:]:
        if not line.strip():
            continue
        row = line.split()
        row_x = float(row[x_index])
        row_y = float(row[y_index])
        x_full.append(row_x)
        y_full.append(row_y)
        for z in z_values:
            if math.isclose(row_x, z, rel_tol=1e-2):
                table_row[z] = row_y

    table_results[case_idx] = table_row
    plot_results[case_idx] = (x_full, y_full)

    return table_results, plot_results


def print_comparison_table(cases, table_results, z_values, reference_case):
    """Print the spot-check comparison table to stdout."""
    ref_idx = reference_case - 1
    ref_label = cases[ref_idx][0]

    header = "t"
    for idx, (label, _) in enumerate(cases):
        header += f"\t{label}"
        if idx != ref_idx:
            header += f"\trel_err vs {ref_label} (%)"
    print(header)

    for z in z_values:
        ref_val = table_results[ref_idx].get(z, None)
        row = f"{z}"
        for idx in range(len(cases)):
            val = table_results[idx].get(z, None)
            row += f"\t{val if val is not None else 'N/A'}"
            if idx != ref_idx:
                if val is not None and ref_val is not None and ref_val !=
                    0:
                    rel_err = abs(val - ref_val) / abs(ref_val) * 100
                    row += f"\t{rel_err:.2f}"
                else:
                    row += "\tN/A"

```

```

        print(row)

# =====
↳ =====
# BUILD CASES
# =====
↳ =====

cases_A = build_cases(parent_folders_A, explicit_folders_A, labels_A,
↳ num_cases, filename)

if COMPARE_TWO_SCENARIOS:
    cases_B = build_cases(parent_folders_B, explicit_folders_B, labels_B,
↳ num_cases_B, filename)

# =====
↳ =====
# READ DATA
# =====
↳ =====

table_A, plot_A = read_cases(cases_A, z_values)

if COMPARE_TWO_SCENARIOS:
    table_B, plot_B = read_cases(cases_B, z_values)

# =====
↳ =====
# PRINT TABLES
# =====
↳ =====

if COMPARE_TWO_SCENARIOS:
    print(f"\n{'='*60}")
    print(f"  Scenario A: {scenario_A_label}")
    print(f"{'='*60}")

# print_comparison_table(cases_A, table_A, z_values, reference_case)

if COMPARE_TWO_SCENARIOS:
    print(f"\n{'='*60}")
    print(f"  Scenario B: {scenario_B_label}")
    print(f"{'='*60}")

```

```

    print_comparison_table(cases_B, table_B, z_values, reference_case)

# =====
# PLOT
# =====

fig, ax = plt.subplots(figsize=(10, 6))

if not COMPARE_TWO_SCENARIOS:
    # ---- original single-scenario mode ----
    title = (
        f"Volume between LFL and UFL over time\n"
        f"depending on leak direction ({pressure[press_i]}, d={diam}mm)"
    )
    for idx, (label, _) in enumerate(cases_A):
        x_vals, y_vals = plot_A[idx]
        ax.plot(x_vals, y_vals,
                color=colors[idx % len(colors)],
                linewidth=2,
                label=label)
    ax.legend(framealpha=0.9)

else:
    # ---- dual-scenario mode ----
    title = (
        f"Volume between LFL and UFL over time\n"
        f"{scenario_A_label} vs {scenario_B_label} "
        f"({pressure[press_i]}, d={diam}mm)"
    )

    # line style per scenario: A = solid, B = dashed
    scenario_styles = ["-", "--"]

    # Plot scenario A (solid)
    for idx, (label, _) in enumerate(cases_A):
        x_vals, y_vals = plot_A[idx]
        ax.plot(x_vals, y_vals,
                color=colors[idx % len(colors)],
                linewidth=2,
                linestyle=scenario_styles[0])

    # Plot scenario B (dashed), same colour palette

```

```

for idx, (label, _) in enumerate(cases_B):
    x_vals, y_vals = plot_B[idx]
    ax.plot(x_vals, y_vals,
            color=colors[idx % len(colors)],
            linewidth=2,
            linestyle=scenario_styles[1])

# Legend 1 - case colours (one entry per case, solid line as swatch)
case_labels = [label for label, _ in cases_A] # A and B share the
            ↪ same case labels
case_handles = [
    Line2D([0], [0], color=colors[i % len(colors)], linewidth=2,
           linestyle="--", label=case_labels[i])
    for i in range(len(cases_A))
]

# Legend 2 - scenario line styles
style_handles = [
    Line2D([0], [0], color="k", linewidth=2, linestyle=ls, label=s1)
    for s1, ls in zip([scenario_A_label, scenario_B_label],
            ↪ scenario_styles)
]

legend1 = ax.legend(handles=case_handles, loc="lower right",
            ↪ bbox_to_anchor=(1, 0),
            title="Leak direction", framealpha=0.9)
ax.add_artist(legend1)
ax.legend(handles=style_handles, loc="lower right",
            ↪ bbox_to_anchor=(0.8125, 0),
            title="Ventilation type", framealpha=0.9)

ax.set_xlim(x_lim)
ax.set_xlabel(x_label)
ax.set_ylabel(y_label)
ax.set_title(title)
ax.grid(True, linestyle="--", alpha=0.5)
plt.tight_layout()

if savePlot == 1:
    plt.savefig(save_path, dpi=300, bbox_inches="tight")
    print(f"\nSaved to {save_path}")
if showPlot == 1:
    plt.show()

```


B.3 Dynamic pressure

```

"""
dynamic_pressure_analysis.py - KFX dynamic pressure analysis
=====

Computes dynamic pressure ( $\frac{1}{2}\rho U^2$ ) at ground level from KFX velocity and
temperature fields. Density rho is derived from temperature via the ideal
gas law.

Two operating modes, each controlled by a boolean:

    MODE 1 - COMPARE (RUN_COMPARISON = True):
        Loops through all combinations of ventilation type / pressure /
        hole size / case number, ranks results by danger score and prints
        a sorted table. Danger score = max dynamic pressure; ties broken
        by area exceeding a threshold.

    MODE 2 - PLOT (RUN_PLOT = True):
        Plots a colourmap of dynamic pressure at a fixed Z slice on top of
        the floorplan for a chosen case. Case is selected either
        automatically (most dangerous from the comparison) or manually via
        index variables.

Field names expected in r3d file:
    'U(m/s)', 'V(m/s)', 'W(m/s)', 'Temp(K)', ...
"""

from pathlib import Path
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import matplotlib.colors as mcolors
from matplotlib.colors import LogNorm
from matplotlib.gridspec import GridSpec
from matplotlib.ticker import FuncFormatter, LogLocator
from kfxutils import r3d

# =====
#  ↳ =====
# CONFIGURATION
# =====
#  ↳ =====

```

```

# --- Operating modes
↳ -----
RUN_COMPARISON = 1    # loop all cases, print ranked danger table
RUN_PLOT        = 0    # plot dynamic pressure colourmap for a chosen case

# --- Physics constants
↳ -----
P_ATM          = 101325.0    # ambient pressure (Pa) - used in ideal gas law

# --- File -----
↳ -----
R3D_FILENAME    = "case_std_exit.r3d"    # present in every caseN subfolder
FLOORPLAN_JSON = "floorplan.json"        # set to None to skip floorplan
↳ overlay

# --- Domain structure
↳ -----
BASE_PATH      = Path("/proj/P10475_RA/STUDENTS/2026/Nur/simulations")
VENT_TYPES     = ["naturalVent", "forcedVent"]
PRESSURES      = ["430bar-fire", "630bar-fire", "900bar-fire"]
HOLE_CATS      = ["1area", "10area", "fullBore"]
NUM_CASES      = 3    # caseN subfolders per combination

# --- Cases to SKIP (relative path from BASE_PATH, e.g. if sim didn't
↳ converge)
# Format: "ventType/pressure/holeSize/caseN"
SKIP_CASES = [
    # "naturalVent/430bar-fire/fullBore/case2",
    # "forcedVent/900bar-fire/1area/case1",
]

# --- Ground-level Z slice (for colourmap plot)
↳ -----
SLICE_Z = 0.1    # metres - low enough to represent ground-level conditions

# --- Danger score tie-breaker threshold
↳ -----
SCORE_PERCENTILE = 99.9    # percentile used for danger score (tune after
↳ seeing data)

# --- Plot: case selection
↳ -----

```

```

PLOT_MOST_DANGEROUS = 0    # True ? auto-select top-ranked case from
    ↪ comparison
                                # False ? use manual indices below

# Manual case selection (only used when PLOT_MOST_DANGEROUS = False)
PLOT_VENT_I = 1    # index into VENT_TYPES
PLOT_PRESS_I = 0    # index into PRESSURES
PLOT_HOLE_I = 2    # index into HOLE_CATS
PLOT_CASE_N = 3    # caseN number (1-based)

# --- Plot appearance
    ↪ -----
SLICE_Z_PLOT = 0.1    # Z slice for the colourmap (can differ from SLICE_Z
    ↪ above)

# --- colormap config (smooth, continuous) ---
DP_CMAP = "YlOrRd"    # or "hot_r", "plasma", "inferno" - any
    ↪ continuous cmap
DP_VMIN = 1    # Pa - log scale can't include 0, set a small positive
    ↪ floor
DP_VMAX = 1e4    # auto-scale to data max, or fix e.g. 1e6

FLOORPLAN_EC = "steelblue"
GEOM_EDGE = "steelblue"

# --- Save paths
    ↪ -----
SAVE_DIR =
    ↪ Path("/proj/P10475_RA/STUDENTS/2026/Nur/img/plots/dynamicPressure")
SAVE_TABLE = SAVE_DIR / "dynpres_ranking.txt"

savePlot = 1
showPlot = 0

# --- plot styling
    ↪ -----

mpl.rcParams.update({
    "text.usetex":    False,
    "font.family":    "serif",
    "mathtext.fontset": "cm",
    "font.size":    14,
    "axes.titlesize": 16,
    "axes.labelsize": 15,

```

```

    "xtick.labelsize": 13,
    "ytick.labelsize": 13,
    "legend.fontsize": 13,
    "figure.dpi": 150,
    "savefig.dpi": 300,
})

# --- Spatial zones with weights
# -----
# Weight = 0 means the zone is completely ignored in the score.
# Lower weight = less dangerous even if pressure is high there.
# Tune these after seeing results - the relative ratios matter, not
#   ↪ absolutes.
ZONES = [
    {
        "label": "container_interior",
        "x0": -3.5, "y0": -4.0, "x1": -1.5, "y1": 1.0,
        "weight": 0.25,    # fully confined, low risk
    },
    {
        "label": "firewall_zone1",
        "x0": -9.0, "y0": -9.0, "x1": -4.0, "y1": 4.0,
        "weight": 0.5,    # some risk (equipment damage, worker injury)
    },
    {
        "label": "firewall_zone2",
        "x0": -4.0, "y0": -2.5, "x1": 11.0, "y1": 4.0,
        "weight": 0.5,    # some risk (equipment damage, worker injury)
    },
    {
        "label": "firewall_zone3",
        "x0": -4.0, "y0": -6.0, "x1": 1.0, "y1": -2.5,
        "weight": 0.5,    # some risk (equipment damage, worker injury)
    },
    {
        "label": "outside",
        "x0": -999, "y0": -999, "x1": 999, "y1": 999,
        "weight": 1.0,    # full weight - public safety risk
    },
]

# Zones are evaluated in order. A cell is assigned to the FIRST zone it
#   ↪ falls into.
# So put the most specific (smallest) zones first, and the catch-all last.

```

```

# =====
# ↪ =====
# HELPERS - FILE I/O
# =====
# ↪ =====

def load_r3d(filepath):
    """
    Load a KFX r3d file and return raw reader object.
    Handles type-4 files by converting to type-2 in place.
    """
    reader = r3d.read(str(filepath))
    if reader.r3d_type == 4:
        reader.to_type2()
    return reader

def get_field(reader, fieldname):
    """
    Extract a named field from an r3d reader as a float64 array (nx, ny,
    ↪ nz).
    Raises a clear ValueError if the field is absent.
    """
    fieldnames = [f.strip() for f in reader.fieldnames]
    try:
        idx = fieldnames.index(fieldname)
    except ValueError:
        raise ValueError(
            f"Field '{fieldname}' not found. Available: {fieldnames}"
        )
    return reader.fielddata[idx].astype(float)

def load_floorplan():
    """Load floorplan rectangles from JSON. Returns [] if file missing or
    ↪ None."""
    import json
    if FLOORPLAN_JSON is None:
        return []
    p = Path(FLOORPLAN_JSON)
    if not p.exists():
        print(f"WARNING: {FLOORPLAN_JSON} not found - floorplan will not be
        ↪ drawn.")
    return []

```

```

    with open(p) as f:
        return json.load(f)

def draw_floorplan(ax, rects):
    """Overlay floorplan geometry as unfilled rectangles on ax."""
    for r in rects:
        patch = mpatches.Rectangle(
            (r["x0"], r["y0"]), r["x1"] - r["x0"], r["y1"] - r["y0"],
            linewidth=1.2, edgecolor=FLOORPLAN_EC, facecolor="none",
            ↪ zorder=2,
        )
        ax.add_patch(patch)

# =====
↪ =====
# HELPERS - PHYSICS
# =====
↪ =====

def compute_dynamic_pressure(reader):
    """
    Compute dynamic pressure  $q = 1/2 \cdot \rho \cdot U^2$  with mixture density.
     $\rho_{\text{mix}}$  is computed cell-by-cell from the H2/air volume fractions
    and local temperature via the ideal gas law.
    """

    U = get_field(reader, "U(m/s)")
    V = get_field(reader, "V(m/s)")
    W = get_field(reader, "W(m/s)")
    T = get_field(reader, "Temp(K)")
    xH2 = get_field(reader, "Fuel gas (vol %)") / 100.0    # volume
    ↪ fraction H2

    T = np.clip(T, 1.0, None)    # guard against unphysical zero/negative
    ↪ temps

    # --- specific gas constants ---
    R_AIR = 287.058    # J/kg/K
    R_H2 = 4157.0    # J/kg/K (R_universal / M_H2 = 8314.46 / 2.016)

    # volume fraction ? mole fraction (ideal gas: they're equal)
    # mixture specific gas constant weighted by mole fractions
    R_mix = xH2 * R_H2 + (1.0 - xH2) * R_AIR    # J/kg/K, cell-by-cell

```

```

# ideal gas:  $\rho = P / (R_{mix} * T)$ 
rho = P_ATM / (R_mix * T)

speed_sq = U**2 + V**2 + W**2
q        = 0.5 * rho * speed_sq

xc = np.asarray(reader.xc)
yc = np.asarray(reader.yc)
zc = np.asarray(reader.zc)

return q, xc, yc, zc

def nearest_z_index(zc, z_target):
    """Return index of the z cell-centre closest to z_target."""
    return int(np.argmin(np.abs(zc - z_target)))

def assign_zone_weights(xc, yc):
    """
    Build a 2D weight array (nx, ny) where each cell gets the weight of
    the first zone in ZONES whose rectangle contains it.
    Zones are checked in order - put most specific first, catch-all last.
    """
    nx, ny = len(xc), len(yc)
    weights = np.ones((nx, ny)) # default to 1.0 (outside catch-all)

    # iterate zones in reverse so that more specific zones overwrite
    #   ↪ general ones
    for zone in reversed(ZONES):
        xi = (xc >= zone["x0"]) & (xc <= zone["x1"])
        yi = (yc >= zone["y0"]) & (yc <= zone["y1"])
        weights[np.ix_(xi, yi)] = zone["weight"]

    return weights

def danger_score(q_2d, xc, yc):
    """
    Weighted percentile-based danger score.

    Steps:
    1. Apply spatial weights - cells inside the container get weight 0,
       firewall zone gets partial weight, outside gets full weight.
    2. Compute the weighted dynamic pressure field.
    """

```

3. Score = high percentile of the weighted field.

Tie-breaker = number of cells exceeding that percentile value.

This means a confined spike inside the container contributes nothing, a spike inside the firewalls contributes partially, and only pressure in the open zone drives the score fully.

"""

```
weights      = assign_zone_weights(xc, yc)
q_weighted   = q_2d * weights                # zero out
    ↪ container, downweight firewall

# compute percentile only over cells that actually contribute (weight
    ↪ > 0)
active       = q_weighted[weights > 0]
if active.size == 0:
    return 0.0, 0

p_score      = float(np.nanpercentile(active, SCORE_PERCENTILE))
area_above   = int(np.sum(q_weighted > p_score))    # spatial extent
    ↪ of hazard
return p_score, area_above
```

```
# =====
    ↪ =====
# MODE 1 - COMPARISON LOOP
# =====
    ↪ =====
```

```
def run_comparison():
    """
    Loop through all combinations of vent type / pressure / hole size /
    ↪ case.
    For each, load the exit r3d file, compute dynamic pressure at ground
    ↪ level,
    and store the danger score.
    Returns a list of result dicts, sorted by danger (descending).
    """
    results = []

    for vent in VENT_TYPES:
        for pressure in PRESSURES:
            for hole in HOLE_CATS:
                for case_n in range(1, NUM_CASES + 1):
```



```

# build relative ID and absolute path
rel_id    = f"{vent}/{pressure}/{hole}/case{case_n}"
case_path = BASE_PATH / vent / pressure / hole /
    ↪ f"case{case_n}"
r3d_path  = case_path / R3D_FILENAME

# --- skip list check ---
if rel_id in SKIP_CASES:
    print(f"  SKIPPED (in skip list): {rel_id}")
    continue

# --- file existence check ---
if not r3d_path.exists():
    print(f"  SKIPPED (file not found): {r3d_path}")
    continue

print(f"  Processing: {rel_id}")

try:
    reader      = load_r3d(r3d_path)
    q, xc, yc, zc = compute_dynamic_pressure(reader)
except Exception as e:
    print(f"    SKIPPED (load error): {e}")
    continue

# extract ground-level Z slice
iz    = nearest_z_index(zc, SLICE_Z)
z_act = float(zc[iz])
q_2d  = q[:, :, iz]    # shape (nx, ny)

max_q, area_above = danger_score(q_2d, xc, yc)

results.append({
    "id":          rel_id,
    "path":        case_path,
    "max_q":       max_q,
    "area_above":  area_above,
    "z_actual":    z_act,
})

# sort: primary = max_q descending, secondary = area_above descending
results.sort(key=lambda r: (r["max_q"], r["area_above"]), reverse=True)
return results

```

```

def print_comparison_table(results):
    """Print and optionally save the ranked danger table."""
    col_id = 55
    header = f"{'Rank':<6} {'Case ID':<{col_id}} {'Max q (Pa)':>12}
    ↪ {'Area > threshold':>18} {'Z actual (m)':>13}"
    sep = "=" * len(header)

    print(f"\n{sep}")
    print(f" Dynamic pressure danger ranking | Z = {SLICE_Z} m ")
    print(f" Sorted by: max dynamic pressure (Pa); ties broken by area
    ↪ exceeding threshold")
    print(sep)
    print(header)
    print("-" * len(header))

    lines = [header, "-" * len(header)]
    for rank, r in enumerate(results, 1):
        row = (f"{'rank':<6} {r['id']:<{col_id}} {r['max_q']:>12.2f}"
              f" {r['area_above']:>18d} {r['z_actual']:>13.3f}")
        print(row)
        lines.append(row)

    print(sep)
    lines.append(sep)

    # save to text file
    SAVE_DIR.mkdir(parents=True, exist_ok=True)
    with open(SAVE_TABLE, "w") as f:
        f.write("\n".join(lines))
    print(f"\nTable saved in {SAVE_TABLE}")

# =====
# ↪ =====
# MODE 2 - PLOT
# =====
# ↪ =====

def resolve_plot_case(comparison_results):
    """
    Return the Path of the case to plot, based on PLOT_MOST_DANGEROUS
    ↪ flag.
    If PLOT_MOST_DANGEROUS is True, uses the top-ranked result from the
    comparison. Otherwise builds the path from the manual index variables.

```

```

"""
if PLOT_MOST_DANGEROUS:
    if not comparison_results:
        raise RuntimeError("No comparison results available to
            ↪ auto-select from.")
    top = comparison_results[0]
    print(f"\nAuto-selected most dangerous case: {top['id']}")
    return top["path"], top["id"]
else:
    # manual selection via index variables
    vent      = VENT_TYPES[PLOT_VENT_I]
    pressure   = PRESSURES[PLOT_PRESS_I]
    hole       = HOLE_CATS[PLOT_HOLE_I]
    rel_id     = f"{vent}/{pressure}/{hole}/case{PLOT_CASE_N}"
    path       = BASE_PATH / vent / pressure / hole / f"case{PLOT_CASE_N}"
    print(f"\nManually selected case: {rel_id}")
    return path, rel_id

def plot_dynamic_pressure(case_path, case_id, floorplan):
    """
    Load the exit r3d for case_path and plot the dynamic pressure
    ↪ colourmap
    at SLICE_Z_PLOT height, overlaid on the floorplan.
    """
    r3d_path = case_path / R3D_FILENAME
    if not r3d_path.exists():
        print(f"ERROR: r3d file not found for plot: {r3d_path}")
        return

    reader      = load_r3d(r3d_path)
    q, xc, yc, zc = compute_dynamic_pressure(reader)

    iz      = nearest_z_index(zc, SLICE_Z_PLOT)
    z_act   = float(zc[iz])
    q_2d    = q[:, :, iz]    # ground-level 2D slice

    print(f"  Z slice: requested {SLICE_Z_PLOT:.2f} m, nearest {z_act:.3f}
        ↪ m (index {iz})")

    X, Y = np.meshgrid(xc, yc, indexing="ij")

    # --- figure setup using GridSpec for shared colourbar ---
    fig = plt.figure(figsize=(10, 8))

```

```

gs = GridSpec(1, 2, figure=fig,
              width_ratios=[1, 0.04],
              wspace=0.08)
ax = fig.add_subplot(gs[0, 0])
cax = fig.add_subplot(gs[0, 1])    # dedicated colourbar axis

# smooth colourmap - pcolormesh maps every cell to its own colour with
    ↪ no binning
vmax = DP_VMAX if DP_VMAX is not None else float(np.nanmax(q_2d))
vmin = max(DP_VMIN, float(np.nanmin(q_2d[q_2d > 0])))    # avoid log(0)

cf = ax.pcolormesh(X, Y, np.clip(q_2d, vmin, None),    # clip zeros to
    ↪ vmin
                  cmap=DP_CMAP,
                  norm=LogNorm(vmin=vmin, vmax=vmax),
                  shading="nearest",
                  alpha=0.85, zorder=1)

# --- isolines on top ---
#ax.contour(X, Y, q_2d, levels=DP_LEVELS,
#           colors="black", linewidths=0.6, zorder=3)

# --- colourbar ---
cb = fig.colorbar(cf, cax=cax)
cb.ax.yaxis.set_major_locator(LogLocator())
cb.ax.yaxis.set_major_formatter(FuncFormatter(lambda x, _: f"{x:,.0f}"))
cb.set_label("Dynamic pressure (Pa)", fontsize=13)
cb.ax.tick_params(labelsize=11)

draw_floorplan(ax, floorplan)

ax.set_xlim(-15, 15)
ax.set_ylim(-15, 7)
ax.set_aspect("equal")
ax.set_xlabel("X (m)")
ax.set_ylabel("Y (m)")
ax.set_title(
    f"Dynamic pressure | {case_id}\n"
    f"Z = {z_act:.2f} m | max = {float(np.nanmax(q_2d)):.1f} Pa",
    fontsize=13,
)

fig.suptitle(r"Dynamic pressure  $\frac{1}{2}\rho U^2$  | ideal gas density",
    ↪ fontsize=15, y=.95)

```

```

if savePlot:
    SAVE_DIR.mkdir(parents=True, exist_ok=True)
    # case_id looks like "naturalVent/430bar-fire/1area/case1"
    # replace slashes with underscores to get a valid flat filename
    safe_id = case_id.replace("/", "_")
    save_plot = SAVE_DIR / f"dynpres_{safe_id}.png"
    fig.savefig(save_plot, dpi=300, bbox_inches="tight")
    print(f" Plot saved in {save_plot}")
if showPlot:
    plt.show()
plt.close(fig)

# =====
↳ =====
# MAIN
# =====
↳ =====

def main():
    floorplan = load_floorplan()
    comparison_results = [] # populated by MODE 1, consumed by MODE 2

    # -----
    ↳ -----
    # MODE 1 - COMPARISON
    # -----
    ↳ -----
    if RUN_COMPARISON:
        print("\n" + "=" * 60)
        print(" MODE 1: Running comparison across all cases")
        print("=" * 60)
        comparison_results = run_comparison()
        print_comparison_table(comparison_results)

    # -----
    ↳ -----
    # MODE 2 - PLOT
    # -----
    ↳ -----
    if RUN_PLOT:
        print("\n" + "=" * 60)
        print(" MODE 2: Plotting dynamic pressure colourmap")
        print("=" * 60)

```

```
# if comparison wasn't run but PLOT_MOST_DANGEROUS is True, warn
→ user
if PLOT_MOST_DANGEROUS and not RUN_COMPARISON:
    print("WARNING: PLOT_MOST_DANGEROUS = True but RUN_COMPARISON =
→ False.")
    print("        Run comparison first, or set
→ PLOT_MOST_DANGEROUS = False.")
    return

case_path, case_id = resolve_plot_case(comparison_results)
plot_dynamic_pressure(case_path, case_id, floorplan)

if __name__ == "__main__":
    main()
```

B.3.1 Pressure peaking phenomenon

Evaluation

```

"""
compare_cases_concentration.py - KFX peak concentration and UFL volume
=====
Extracts two quantities from KFX simulation results:

    1. Peak H2 concentration (vol %) at steady state - from the exit r3d
       ↪ file
    2. Volume exceeding 100% UFL over time - from case_gascloud.out

Produces:
    - Time-series plot of Volume > 100% UFL for all cases
    - Bar chart of peak steady-state concentration per case
    - CSV with all extracted values
"""

import os
import csv
from pathlib import Path

import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D

from kfxutils import r3d

# =====
# ↪ =====
# CONFIGURATION
# =====
# ↪ =====

# --- booleans
# ↪ -----
PLOT_TIMESERIES = True    # Volume > 100% UFL over time
PLOT_PEAK_CONC  = True    # bar chart of peak concentration at steady state
SAVE_CSV        = True    # write results to CSV
savePlot = True
showPlot = False

```

```

# --- Case info
↳ -----
BASE_PATH = Path("/proj/P10475_RA/STUDENTS/2026/Nur/simulations")
VENT_TYPES = ["naturalVent", "forcedVent"]
PRESSURES = ["430bar", "630bar", "900bar"]
HOLE_CATS = ["1area", "10area", "fullBore"]
NUM_CASES = 3

# --- threshold: only plot cases where peak conc exceeds this value
↳ -----
PEAK_CONC_THRESHOLD = 99.0    # vol % - set to 0 to plot all cases

# Cases to skip - format: "ventType/pressure/holeSize/caseN"
SKIP_CASES = [
    # "naturalVent/900bar-fire/10area/case3",
]

# =====
↳ =====
# OPTION A - explicit folder list (leave empty to use domain structure
↳ above)
# =====
↳ =====
CASE_FOLDERS = [
    # "/proj/.../case1",
    # "/proj/.../case2",
]

# --- Time axis limits for the time-series plot
↳ -----
x_lim = (0, 80)    # seconds

# --- R3D file for peak concentration (steady state exit snapshot)
↳ -----
R3D_EXIT_FILE = "case_std_exit.r3d"
CONC_FIELDNAME = "Fuel gas (vol %)"

# --- Gascloud output file
↳ -----
GASCLOUD_FILE = "case_gascloud.out"
GASCLOUD_COLUMN = ' " Volume > 100% UFL "'    # exact name including spaces
TIME_COLUMN = ' " Time "'                      # exact name including spaces

```



```

# =====
↳ =====
# END OF CONFIGURATION
# =====
↳ =====

# --- Save paths
↳ -----
_save_dir    = "/proj/P10475_RA/STUDENTS/2026/Nur/img/plots/ppp"
_tag         = f"allCases"
save_path_ts = os.path.join(_save_dir, f"uflVolume_timeseries_{_tag}.png")
save_path_bar = os.path.join(_save_dir, f"peakConc_bar_{_tag}.png")
save_path_csv = os.path.join(_save_dir, f"concentration_{_tag}.csv")
save_path_exceeding = os.path.join(_save_dir, f"exceeding_cases_{_tag}.txt")

# --- Styling
↳ -----
# colours cycle through cases (same index = same direction in both
↳ scenarios)
colors = ["#e6194b", "#3cb44b", "#4363d8", "#f58231",
          "#911eb4", "#42d4f4", "#f032e6"]
markers = ["o", "s", "^", "D", "v", "P", "X"] # different symbols per
↳ case

mpl.rcParams.update({
    "text.usetex":      False,
    "font.family":      "serif",
    "mathtext.fontset": "cm",
    "font.size":        14,
    "axes.titlesize":   16,
    "axes.labelsize":   15,
    "xtick.labelsize":  13,
    "ytick.labelsize":  13,
    "legend.fontsize":  13,
    "figure.dpi":       150,
    "savefig.dpi":      300,
})

# =====
↳ =====
# HELPERS - FOLDER BUILDING
# =====
↳ =====

```

```

def build_cases_option_a():
    """Return list of (label, Path) for Option A (explicit folders)."""
    return [(Path(f).name, Path(f)) for f in CASE_FOLDERS]

def build_cases_all():
    """
    Return list of (rel_id, label, Path) for all combinations of
    vent type / pressure / hole size / case number.
    Skips entries in SKIP_CASES and missing folders.
    """
    cases = []
    for vent in VENT_TYPES:
        for pressure in PRESSURES:
            for hole in HOLE_CATS:
                for case_n in range(1, NUM_CASES + 1):
                    rel_id = f"{vent}/{pressure}/{hole}/case{case_n}"
                    case_path = BASE_PATH / vent / pressure / hole /
                        f"case{case_n}"

                    if rel_id in SKIP_CASES:
                        print(f" SKIPPED (skip list): {rel_id}")
                        continue
                    if not case_path.exists():
                        print(f" SKIPPED (not found): {rel_id}")
                        continue

                    # label shows vent type + direction shorthand
                    label = rel_id
                    cases.append((rel_id, label, case_path))

    return cases

# =====
# ↪ =====
# HELPERS - DATA LOADING
# =====
# ↪ =====

def load_peak_concentration(folder):
    """
    Load the exit r3d file and return the global max fuel gas
    ↪ concentration
    (vol %) across the entire domain - represents steady-state peak.
    """

```

```

fpath = folder / R3D_EXIT_FILE
if not fpath.exists():
    print(f"    SKIPPED peak conc (not found): {fpath}")
    return np.nan

reader = r3d.read(str(fpath))
if reader.r3d_type == 4:
    reader.to_type2()

fieldnames = [f.strip() for f in reader.fieldnames]
if CONC_FIELDNAME not in fieldnames:
    print(f"    SKIPPED peak conc (field not found): {CONC_FIELDNAME}")
    return np.nan

gas = reader.fielddata[fieldnames.index(CONC_FIELDNAME)].astype(float)
return float(np.nanmax(gas))    # global max across all cells

def load_ufl_volume_timeseries(folder):
    """
    Read case_gascloud.out and extract time and Volume > 100% UFL columns.
    Returns (times, volumes) as numpy arrays, or (None, None) if file
    ↪ missing.
    """
    fpath = folder / GASCLOUD_FILE
    if not fpath.exists():
        print(f"    SKIPPED UFL volume (not found): {fpath}")
        return None, None

    with open(fpath, "r") as f:
        header_line = f.readline()
        data_lines = f.readlines()

    # parse semicolon-separated header - keep spaces intact, no stripping
    headers = [h for h in header_line.split(";")]

    # find column indices - names include surrounding spaces
    try:
        t_idx = headers.index(TIME_COLUMN)
        ufl_idx = headers.index(GASCLOUD_COLUMN)
    except ValueError:
        print(f"    Column not found. Available: {headers}")
        return None, None

```

```

times, volumes = [], []
for line in data_lines[1:]:
    if not line.strip():
        continue
    parts = line.split()
    try:
        times.append(float(parts[t_idx]))
        volumes.append(float(parts[ufl_idx]))
    except (IndexError, ValueError):
        continue

return np.array(times), np.array(volumes)

def load_cases(cases):
    """
    Load all data for a list of (rel_id, label, Path) cases.
    Returns list of dicts with keys: rel_id, label, times,
    ufl_volumes, peak_conc.
    """
    results = []
    for rel_id, label, folder in cases:
        print(f"\n Processing: {rel_id}")

        times, vols = load_ufl_volume_timeseries(folder)
        peak_conc = load_peak_concentration(folder)

        print(f"    peak conc = {peak_conc:.2f} vol% | "
              f"UFL timesteps = {len(times) if times is not None else 0}")

        results.append({
            "rel_id": rel_id,
            "label": label,
            "times": times,
            "ufl_volumes": vols,
            "peak_conc": peak_conc,
        })
    return results

# =====
# CSV SAVE
# =====

```

```

def save_results_csv(results_and_labels, filepath):
    """
    Save all results to CSV.
    results_and_labels: list of (results, scenario_label) tuples.
    Writes one row per timestep for UFL volume, plus a row for peak conc.
    """
    rows = []
    for results, scenario_label in results_and_labels:
        for r in results:
            # one row per timestep for the time series
            if r["times"] is not None:
                for t, v in zip(r["times"], r["ufl_volumes"]):
                    rows.append({
                        "scenario": scenario_label,
                        "label": r["label"],
                        "type": "ufl_volume_timeseries",
                        "time_s": t,
                        "ufl_volume_m3": v,
                        "peak_conc_volpct": "",
                    })
            # one row for steady-state peak concentration
            rows.append({
                "scenario": scenario_label,
                "label": r["label"],
                "type": "peak_concentration",
                "time_s": "",
                "ufl_volume_m3": "",
                "peak_conc_volpct": r["peak_conc"],
            })

    if not rows:
        print(" No results to save.")
        return

    Path(filepath).parent.mkdir(parents=True, exist_ok=True)
    with open(filepath, "w", newline="") as f:
        writer = csv.DictWriter(f, fieldnames=rows[0].keys())
        writer.writeheader()
        writer.writerows(rows)
    print(f"CSV Saved in {filepath}")

# =====
↪ =====

```

```

# PLOT 1 - UFL VOLUME TIME SERIES
# =====
↳ =====

def plot_timeseries(results):
    fig, ax = plt.subplots(figsize=(10, 6))

    for idx, res in enumerate(results):
        if res["times"] is None:
            continue
        ax.plot(res["times"], res["ufl_volumes"],
                color=colors[idx % len(colors)],
                marker=markers[idx % len(markers)],
                markevery=max(1, len(res["times"]) // 8),
                linewidth=2, markersize=6,
                label=res["label"])

    ax.legend(loc="upper right", fontsize=9, framealpha=0.9)
    ax.set_xlim(x_lim)
    ax.set_xlabel(r"Time $t$ (s)")
    ax.set_ylabel(r"Volume  $> 100\%$  UFL ( $m^3$ )")
    ax.set_title("Volume exceeding 100% UFL")
    ax.grid(True, linestyle="--", alpha=0.5)
    plt.tight_layout()

    if savePlot:
        Path(_save_dir).mkdir(parents=True, exist_ok=True)
        plt.savefig(save_path_ts, dpi=300, bbox_inches="tight")
        print(f"Saved in {save_path_ts}")
    if showPlot:
        plt.show()
    plt.close(fig)

# =====
↳ =====

# PLOT 2 - PEAK CONCENTRATION BAR CHART
# =====
↳ =====

def plot_peak_concentration(results):
    """
    Bar chart of peak steady-state H2 concentration per case.
    Only cases where peak_conc >= PEAK_CONC_THRESHOLD are shown.
    x-axis: case label. One bar per case, coloured by index.

```

```

LFL and UFL reference lines included.
"""
# filter to cases above threshold and with valid data
filtered = [r for r in results
             if r["peak_conc"] is not None
             and not np.isnan(r["peak_conc"])
             and r["peak_conc"] >= PEAK_CONC_THRESHOLD]

if not filtered:
    print(f" No cases above threshold ({PEAK_CONC_THRESHOLD} vol%) "
          f"- peak concentration plot skipped.")
    return

n      = len(filtered)
x_pos = np.arange(n)
bar_w  = 0.6

fig, ax = plt.subplots(figsize=(max(8, n * 0.5), 6))

for i, res in enumerate(filtered):
    ax.bar(x_pos[i], res["peak_conc"], width=bar_w,
           color=colors[i % len(colors)],
           edgecolor="black", linewidth=0.6,
           alpha=0.85, zorder=2)

# reference lines
ax.axhline(4.0, color="orange", linewidth=1.2, linestyle="--",
           label="LFL (4%)", zorder=3)
ax.axhline(75.0, color="red", linewidth=1.2, linestyle="--",
           label="UFL (75%)", zorder=3)

ax.set_xticks(x_pos)
ax.set_xticklabels([r["label"] for r in filtered],
                    rotation=45, ha="right", fontsize=8)
ax.set_ylabel("Peak H2 concentration (vol %)")
ax.set_title(
    rf"Peak H$_2$ concentration at steady state\n"
    rf"(showing cases $\geq$ {PEAK_CONC_THRESHOLD} vol%, "
    rf"{n} of {len(results)} total)"
)
ax.legend(framealpha=0.9)
ax.grid(axis="y", linestyle="--", alpha=0.5)
plt.tight_layout()

```

```

if savePlot:
    Path(_save_dir).mkdir(parents=True, exist_ok=True)
    plt.savefig(save_path_bar, dpi=300, bbox_inches="tight")
    print(f"Saved in {save_path_bar}")
if showPlot:
    plt.show()
plt.close(fig)

def print_and_save_exceeding_cases(results, filepath=None):
    """
    Print a list of cases where peak concentration >= PEAK_CONC_THRESHOLD
    and save them in a format ready to paste into SCENARIOS in the
    analytical model script.

    If filepath is given, also writes the list to a text file.
    """
    exceeding = [r for r in results
                  if r["peak_conc"] is not None
                  and not np.isnan(r["peak_conc"])
                  and r["peak_conc"] >= PEAK_CONC_THRESHOLD]

    if not exceeding:
        print(f"\n No cases exceed {PEAK_CONC_THRESHOLD} vol% - nothing to
              ↳ report.")
        return

    header = (f"\n{'='*60}\n"
              f" Cases exceeding {PEAK_CONC_THRESHOLD} vol% peak
              ↳ concentration\n"
              f" ({len(exceeding)} of {len(results)} cases)\n"
              f"{'='*60}")
    print(header)

    # print table
    col = 55
    print(f" {'Case ID':<{col}} {'Peak conc (vol%)':>18}")
    print(f" {'-'*col} {'-----':>18}")
    for r in sorted(exceeding, key=lambda x: x["peak_conc"], reverse=True):
        print(f" {r['rel_id']:<{col}} {r['peak_conc']:>18.3f}")

    # print ready-to-paste SCENARIOS entries
    # we don't have the mass flow here, but we print the case ID so the
    # user can look it up in the SCENARIOS table in the analytical script

```



```

print(f"\n --- Ready to use in analytical_enclosure_model.py ---")
print(f" Cross-reference these case IDs with the SCENARIOS table")
print(f" and comment in the matching (pressure, diameter) entries:\n")
seen = set()
for r in sorted(exceeding, key=lambda x: x["peak_conc"], reverse=True):
    # extract pressure and hole from rel_id: vent/pressure/hole/caseN
    parts = r["rel_id"].split("/")
    if len(parts) >= 3:
        press_str = parts[1] # e.g. "430bar-fire"
        hole_str = parts[2] # e.g. "fullBore"
        key = (press_str, hole_str)
        if key not in seen:
            seen.add(key)
            print(f" # {press_str} / {hole_str} "
                  f"? peak = {r['peak_conc']:.2f} vol%")

print(f"'='*60)\n")

# optionally write to file
if filepath is not None:
    Path(filepath).parent.mkdir(parents=True, exist_ok=True)
    with open(filepath, "w") as f:
        f.write(header + "\n\n")
        f.write(f" {'Case ID':<{col}} {'Peak conc (vol%)'}\n")
        f.write(f" {'-'*col} -----\n")
        for r in sorted(exceeding,
                        key=lambda x: x["peak_conc"], reverse=True):
            f.write(f" {r['rel_id']:<{col}} {r['peak_conc']:.3f}\n")
        f.write("\n Unique (pressure, hole) combinations:\n")
        seen = set()
        for r in sorted(exceeding,
                        key=lambda x: x["peak_conc"], reverse=True):
            parts = r["rel_id"].split("/")
            if len(parts) >= 3:
                key = (parts[1], parts[2])
                if key not in seen:
                    seen.add(key)
                    f.write(f" {parts[1]} / {parts[2]}"
                          f" ? peak = {r['peak_conc']:.2f} vol%\n")
        print(f"Exceeding cases saved ? {filepath}")

# =====
# MAIN

```

```

# =====
↳ =====

def main():
    option_a = bool(CASE_FOLDERS)

    if option_a:
        print("\n=== Option A mode (explicit folders) ===")
        cases = [(f, Path(f).name, Path(f)) for f in CASE_FOLDERS]
        results = load_cases(cases)

        if SAVE_CSV:
            save_results_csv([(results, "explicit")], save_path_csv)
        if PLOT_TIMESERIES:
            print("NOTE: time-series not available in Option A - skipped.")
        if PLOT_PEAK_CONC:
            plot_peak_concentration(results)
        return

    # Option B - loop all combinations
    print("\n=== Option B mode (full parameter space loop) ===")
    cases = build_cases_all()
    results = load_cases(cases)

    if SAVE_CSV:
        save_results_csv([(results, "all")], save_path_csv)
    if PLOT_TIMESERIES:
        plot_timeseries(results)
    if PLOT_PEAK_CONC:
        plot_peak_concentration(results)

    print_and_save_exceeding_cases(results, filepath=save_path_exceeding)

if __name__ == "__main__":
    main()

```

Model

```

"""
analytical_enclosure_model.py - H2 enclosure pressurisation model
=====
Implements the analytical time-stepping model for H2 accumulation in a
partially occupied enclosure with pressure-driven ventilation.

Governing equations (Euler forward integration):
     $m(t+dt) = m(t) + (m_{\dot{nozz}} - m_{\dot{vent}}) * dt$ 
     $n(t+dt) = n(t) + (m_{\dot{nozz}}/M_{H2} - m_{\dot{vent}}*n(t)/m(t)) * dt$ 
     $P(t+dt) = n(t+dt) * R * T / V$ 
     $m_{\dot{vent}} = C * (m*A/V) * \sqrt{2*(P-P0)*V/m}$ 

where:
    m          = total gas mass in enclosure (kg)
    n          = moles of H2 in enclosure (mol)
    P          = enclosure pressure (Pa)
    m_dot_nozz= nozzle mass flow rate (kg/s) - constant per scenario
    m_dot_vent= vent mass flow rate (kg/s) - pressure-driven, updated
               ↪ each step
    M_H2       = molar mass of H2 (kg/mol)
    R          = universal gas constant (J/mol/K)
    T          = temperature (K)
    V          = effective enclosure volume (m³)
    C          = discharge coefficient (-)
    A          = aggregated vent area (m²)
    P0         = ambient pressure (Pa)

Outputs:
    - Time series of pressure, H2 mass fraction, and vent mass flow
    - CSV of all results
    - Configurable plots
"""

import csv
import os
from pathlib import Path

import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D

```

```

# =====
#  ↳ =====
# PHYSICAL CONSTANTS
# =====
#  ↳ =====

R_UNIVERSAL = 8.314      # universal gas constant (J/mol/K)
M_H2        = 2.016e-3   # molar mass of hydrogen (kg/mol)
M_AIR       = 28.97e-3   # molar mass of air (kg/mol)

# =====
#  ↳ =====
# ENCLOSURE PARAMETERS
# =====
#  ↳ =====

T_K          = 15.0 + 273.15  # temperature (K) - 15°C
V_TOTAL      = 20.0          # total container volume (m³)
EQUIP_FRAC   = 0.20          # fraction occupied by equipment
V_EFF        = V_TOTAL * (1 - EQUIP_FRAC)  # effective free volume (m³)

P0           = 101325.0      # ambient pressure (Pa)
A_VENT       = 0.2856       # aggregated vent area (m²)
C_DISCHARGE  = 0.6          # discharge coefficient (-)

# =====
#  ↳ =====
# SIMULATION PARAMETERS
# =====
#  ↳ =====

T_END = 20.0      # simulation end time (s)
DT    = 1e-3      # time step (s) - small enough for stability

# =====
#  ↳ =====
# LEAK SCENARIOS
# Each entry: (pressure_bar, diameter_mm, mass_flow_kg_s, label)
# Comment out entries you don't want to include.
# =====
#  ↳ =====

SCENARIOS = [
    # pressure   diam   flow      label
    #(430,      1.2,   0.028619, "430 bar - 1.2 mm"),

```

```

(430,      3.8,   0.28698,  "430 bar - 3.8 mm"),
#(430,      12.0,  2.8619,  "430 bar - 12 mm"),
#(630,      1.0,   0.028219, "630 bar - 1.0 mm"),
(630,      3.2,   0.28896,  "630 bar - 3.2 mm"),
#(630,      10.0,  2.8219,  "630 bar - 10 mm"),
#(900,      1.0,   0.03878,  "900 bar - 1.0 mm"),
(900,      2.5,   0.24238,  "900 bar - 2.5 mm"),
#(900,      8.0,   2.482,    "900 bar - 8.0 mm"),
]

# =====
↳ =====
# PLOT TOGGLES
# =====
↳ =====
PLOT_PRESSURE      = 1          # overpressure over time
PLOT_H2_FRACTION    = 0          # H2 mass fraction over time
PLOT_VENT_FLOW      = 0          # vent mass flow rate over time

savePlot = 1
showPlot = 1

# =====
↳ =====
# SAVE PATHS
# =====
↳ =====
_save_dir      = "/proj/P10475_RA/STUDENTS/2026/Nur/img/plots/ppp"
_tag           = "_model_10area"
save_path_pres = os.path.join(_save_dir, f"pressure_{_tag}.png")
save_path_h2   = os.path.join(_save_dir, f"h2fraction_{_tag}.png")
save_path_vent = os.path.join(_save_dir, f"ventflow_{_tag}.png")
save_path_csv  = os.path.join(_save_dir, f"results_{_tag}.csv")

# =====
↳ =====
# STYLING
# =====
↳ =====
# Colours cycle by pressure level, markers by hole size within each
↳ pressure
colors = ["#e6194b", "#3cb44b", "#4363d8",  # 430 / 630 / 900 bar
          "#f58231", "#911eb4", "#42d4f4",
          "#f032e6", "#a9a9a9", "#000075"]

```

```

markers = ["o", "s", "^",    # small / medium / large hole per pressure
           "o", "s", "^",
           "o", "s", "^"]

mpl.rcParams.update({
    "text.usetex":      False,
    "font.family":      "serif",
    "mathtext.fontset": "cm",
    "font.size":        14,
    "axes.titlesize":   16,
    "axes.labelsize":   15,
    "xtick.labelsize":  13,
    "ytick.labelsize":  13,
    "legend.fontsize":  12,
    "figure.dpi":       150,
    "savefig.dpi":      300,
})

# =====
# MODEL
# =====

def run_scenario(m_dot_nozz):
    """
    Time-step the overpressure model for a given nozzle mass flow rate.

    Initial conditions:
    - enclosure filled with air at ambient pressure
    -  $m(0) = \rho_{\text{air}} * V_{\text{eff}}$  (air mass)
    -  $n(0) = 0$  (no H2 initially)
    -  $m_{\text{dot\_vent}}(0) = 0$  (no pressure difference initially)

    Returns dict of time arrays:
    times, pressures, h2_fractions, vent_flows
    """
    n_steps = int(T_END / DT) + 1
    times      = np.zeros(n_steps)
    pressures  = np.zeros(n_steps)
    h2_fractions = np.zeros(n_steps)    # H2 mass fraction (kg H2 / kg
    vent_flows  = np.zeros(n_steps)

```

```

# initial conditions - enclosure full of air at ambient
rho_air_init = P0 * M_AIR / (R_UNIVERSAL * T_K) # kg/m³
m = rho_air_init * V_EFF # total mass (kg)
n_h2 = 0.0 # moles of H2
m_dot_vent = 0.0 # vent flow starts at zero

for i in range(n_steps):
    t = i * DT
    P = n_h2 * R_UNIVERSAL * T_K / V_EFF # partial pressure from H2
    # total pressure: H2 partial + air partial
    # air moles from remaining mass: n_air = (m - n_h2*M_H2) / M_AIR
    n_air = max(0.0, (m - n_h2 * M_H2) / M_AIR)
    P_total = (n_h2 + n_air) * R_UNIVERSAL * T_K / V_EFF

    # store state
    times[i] = t
    pressures[i] = P_total
    h2_fractions[i] = (n_h2 * M_H2 / m) if m > 0 else 0.0
    vent_flows[i] = m_dot_vent

    # vent flow: only active when enclosure pressure exceeds ambient
    dP = P_total - P0
    if dP > 0 and m > 0:
        m_dot_vent_new = (C_DISCHARGE * (m * A_VENT / V_EFF)
                          * np.sqrt(2 * dP * V_EFF / m))
    else:
        m_dot_vent_new = 0.0

    # Euler step: update mass and H2 moles
    dm = (m_dot_nozz - m_dot_vent) * DT
    dn = (m_dot_nozz / M_H2
          - m_dot_vent * n_h2 / m if m > 0 else 0.0) * DT

    m = max(0.0, m + dm)
    n_h2 = max(0.0, n_h2 + dn)
    m_dot_vent = m_dot_vent_new

return {
    "times": times,
    "pressures": pressures,
    "h2_fractions": h2_fractions,
    "vent_flows": vent_flows,
}

```

```

# =====
# CSV SAVE
# =====

def save_csv(all_results):
    """
    Save all scenario results to a single CSV.
    One row per timestep per scenario.
    """
    rows = []
    for (pres, diam, flow, label), result in zip(SCENARIOS, all_results):
        for i, t in enumerate(result["times"]):
            rows.append({
                "label": label,
                "pressure_bar": pres,
                "diameter_mm": diam,
                "flow_kg_s": flow,
                "time_s": t,
                "pressure_Pa": result["pressures"][i],
                "h2_fraction": result["h2_fractions"][i],
                "vent_flow_kg_s": result["vent_flows"][i],
            })

    Path(save_path_csv).parent.mkdir(parents=True, exist_ok=True)
    with open(save_path_csv, "w", newline="") as f:
        writer = csv.DictWriter(f, fieldnames=rows[0].keys())
        writer.writeheader()
        writer.writerows(rows)
    print(f"CSV saved ? {save_path_csv}")

# =====
# PLOTTING
# =====

def _make_legend_handles():
    """Build legend handles: one per scenario, colour + marker."""
    return [
        Line2D([0], [0],

```



```

        color=colors[i % len(colors)],
        marker=markers[i % len(markers)],
        linewidth=2, markersize=7,
        label=label)
    for i, (_, _, _, label) in enumerate(SCENARIOS)
]

def _plot_quantity(all_results, y_key, y_label, title, save_path,
    ↪ ref_lines=None):
    """
    Generic time-series plotter for one quantity across all scenarios.

    Parameters
    -----
    all_results : list of result dicts from run_scenario()
    y_key       : key in result dict to plot on y-axis
    y_label     : y-axis label string
    title       : plot title string
    save_path   : output file path
    ref_lines   : list of (y_value, label, colour) for horizontal
    ↪ reference lines
    """
    fig, ax = plt.subplots(figsize=(11, 6))

    for i, ((pres, diam, flow, label), result) in enumerate(
        zip(SCENARIOS, all_results)):
        t = result["times"]
        y = result[y_key]
        # sparse markers so the line is still visible
        markevery = max(1, len(t) // 10)
        ax.plot(t, y,
            color=colors[i % len(colors)],
            marker=markers[i % len(markers)],
            markevery=markevery,
            linewidth=1.8, markersize=6)

    # optional reference lines (e.g. LFL, ambient pressure)
    if ref_lines:
        for y_ref, ref_label, ref_color in ref_lines:
            ax.axhline(y_ref, color=ref_color, linewidth=1.2,
                linestyle="--", label=ref_label, zorder=1)

    ax.set_xlim(0, T_END)

```

```

ax.set_xlabel(r"Time $t$ (s)")
ax.set_ylabel(y_label)
ax.set_title(title)
ax.grid(True, linestyle="--", alpha=0.5)
ax.legend(handles=_make_legend_handles(),
          loc="upper right", fontsize=11, framealpha=0.9,
          title="Pressure - diameter")

plt.tight_layout()
if savePlot:
    Path(save_path).parent.mkdir(parents=True, exist_ok=True)
    plt.savefig(save_path, dpi=300, bbox_inches="tight")
    print(f"Saved in {save_path}")
if showPlot:
    plt.show()
plt.close(fig)

# =====
↪ =====
# MAIN
# =====
↪ =====

def main():
    print("\n" + "=" * 55)
    print("  Running analytical overpressure model")
    print(f"  V_eff = {V_EFF:.1f} m³ | T = {T_K - 273.15:.1f} °C "
          f"| dt = {DT} s | t_end = {T_END} s")
    print("=" * 55)

    # run model for every scenario
    all_results = []
    for pres, diam, flow, label in SCENARIOS:
        print(f"  {label} ? m_dot = {flow:.5f} kg/s")
        result = run_scenario(flow)
        all_results.append(result)
        print(f"    final P = {result['pressures'][-1]:.1f} Pa | "
              f"final H2 fraction = {result['h2_fractions'][-1]*100:.3f} %")

    # save CSV
    save_csv(all_results)

    # LFL in mass fraction: H2 LFL = 4 vol% ? approx mass fraction

```

```

# x_LFL -> 0.04 * M_H2 / (0.04*M_H2 + 0.96*M_AIR)
lfl_mass_frac = (0.04 * M_H2) / (0.04 * M_H2 + 0.96 * M_AIR)

# plots
if PLOT_PRESSURE:
    _plot_quantity(
        all_results, "pressures",
        y_label="Overpressure (Pa)",
        title="Overpressure over time (analytical model)",
        save_path=save_path_pres,
        ref_lines=[(P0, "Ambient P0", "gray")],
    )

if PLOT_H2_FRACTION:
    _plot_quantity(
        all_results, "h2_fractions",
        y_label=r"H$_2$ mass fraction (kg H$_2$ / kg total)",
        title=r"H$_2$ mass fraction over time | analytical model",
        save_path=save_path_h2,
        ref_lines=[(lfl_mass_frac, "LFL (4 vol%)", "orange")],
    )

if PLOT_VENT_FLOW:
    _plot_quantity(
        all_results, "vent_flows",
        y_label=r"Vent mass flow $\dot{m}_{\text{vent}}$ (kg/s)",
        title="Vent mass flow rate over time | analytical model",
        save_path=save_path_vent,
    )

if __name__ == "__main__":
    main()

```

B.4 Radiation

```

"""
compare_cases_radiation.py - KFX radiation severity comparison
=====

...
"""

import json
from pathlib import Path

import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import matplotlib.colors as mcolors
from matplotlib.lines import Line2D
import numpy as np
from kfxutils import r3d
from matplotlib.gridspec import GridSpec

# =====
#  ↳ =====
# CONFIGURATION
# =====
#  ↳ =====

RAD_FILENAME = "case_bmg_exit.r3d"
RAD_FIELDNAME = "Radiation(W/m2)"

SLICE_Z = 1.5

savePlot      = 1
showPlot      = 0
autoLabels    = 1
COMPARE_TWO_SCENARIOS = 1

pressure = ["430bar", "630bar", "900bar"]
press_i  = 0
holeCat  = ["1area", "10area", "fullBore"]
hole_i   = 2

NUM_CASES    = 3
NUM_CASES_B = 3

```

```

# =====
# SCENARIO A (always active)
# =====

SCENARIO_A_LABEL = "Natural ventilation"

CASE_FOLDERS_A = [
    #"/proj/P10475_RA/STUDENTS/2026/Nur/simulations/wallHeightStudy/1m",
    #"/proj/P10475_RA/STUDENTS/2026/Nur/simulations/wallHeightStudy/2m",
    #"/proj/P10475_RA/STUDENTS/2026/Nur/simulations/wallHeightStudy/3m",
    #"/proj/P10475_RA/STUDENTS/2026/Nur/simulations/forcedVent/430bar-fire
    ↪ /fullBore/4m",
    #"/proj/P10475_RA/STUDENTS/2026/Nur/simulations/wallHeightStudy/5m",
    #"/proj/P10475_RA/STUDENTS/2026/Nur/simulations/wallHeightStudy/6m",
    #"/proj/P10475_RA/STUDENTS/2026/Nur/simulations/wallHeightStudy/7m",
    #"/proj/P10475_RA/STUDENTS/2026/Nur/simulations/wallHeightStudy/8m"
]

PARENT_FOLDERS_A = [
    f"/proj/P10475_RA/STUDENTS/2026/Nur/simulations/naturalVent/{pressure[p_
    ↪ res_i]}-fire/{holeCat[hole_i]}",
]

# =====
# SCENARIO B (only used when COMPARE_TWO_SCENARIOS = True)
# =====

SCENARIO_B_LABEL = "Forced ventilation"

CASE_FOLDERS_B = []
PARENT_FOLDERS_B = [
    f"/proj/P10475_RA/STUDENTS/2026/Nur/simulations/forcedVent/{pressure[pr
    ↪ ess_i]}-fire/{holeCat[hole_i]}",
]

# =====
# LABELS
# =====

CASE_LABELS_A = []
CASE_LABELS_B = []

```

```

if autoLabels:
    match press_i:
        case 0:
            CASE_LABELS_A = ["dir=-y", "dir=+z", "dir=-z"]
            CASE_LABELS_B = ["dir=-y", "dir=+z", "dir=-z"]
        case 1:
            CASE_LABELS_A = ["dir=+x", "dir=-x", "dir=-y"]
            CASE_LABELS_B = ["dir=+x", "dir=-x", "dir=-y"]
        case 2:
            CASE_LABELS_A = ["dir=-x", "dir=+y", "dir=-y"]
            CASE_LABELS_B = ["dir=-x", "dir=+y", "dir=-y"]

if COMPARE_TWO_SCENARIOS:
    out_png = f"/proj/P10475_RA/STUDENTS/2026/Nur/img/plots/naturalVentVsForcedVent/radComparison_{pressure[press_i]}_{holeCat[hole_i]}.png"
else:
    out_png = f"radComparison_{pressure[press_i]}_{holeCat[hole_i]}.png"
    #out_png = "/proj/P10475_RA/STUDENTS/2026/Nur/img/plots/wallHeightStudy/radComp_wallHeight.png"

# --- CSV path

saveCSV = f"/proj/P10475_RA/STUDENTS/2026/Nur/img/plots/naturalVentVsForcedVent/radComparison_{pressure[press_i]}_{holeCat[hole_i]}.csv" # set to None to skip
#saveCSV = "/proj/P10475_RA/STUDENTS/2026/Nur/img/plots/wallHeightStudy/radComp_wallHeight.csv" # set to None to skip

EXCLUSION_ZONES = [
    {"x0": -3.5, "y0": -4, "x1": -1.5, "y1": 1, "label": "container_interior"},
]

FLOORPLAN_JSON = "floorplan.json"

# --- Radiation thresholds ---
RADIATION_THRESHOLDS = [
    (1500, 1, "No damage on long exposures", "gra, raj2008"),
    (5000, 3, "Second-degree burns if exposure is longer than 30s", "raj2008"),
    (17000, 8, "FAA aircraft escape route", "raj2008"),
    (37500, 20, "Immediate fatality", "raj2008, gra"),

```

```

]

# --- Colormap: one colour per interval between threshold levels ---
# Ordered low ? high radiation; "extend both" adds one bin on each side,
# but since we already cover <threshold_min and >threshold_max explicitly
# with the first/last colours, extend="neither" gives full control.
RAD_LEVELS = [t for t, *_ in RADIATION_THRESHOLDS]          # [1500, 2500,
    ↪ 5000, 12500]
RAD_COLORS = [
    "#fee8c8",    # < 1500    W/m2 (very light orange)
    "#fdbb84",    # 1500-5000 W/m2 (light orange)
    "#fc8d59",    # 5000-17000 W/m2 (medium orange)
    "#d7301f",    # 17000-37500 W/m2 (red-orange)
    "#7f0000",    # > 37500   W/m2 (dark red)
]
RAD_CMAP = mcolors.ListedColormap(RAD_COLORS[1:-1])        # 3 interior colours
RAD_NORM = mcolors.BoundaryNorm(RAD_LEVELS, ncolors=3)
RAD_CMAP.set_under(RAD_COLORS[0])                          # < 1500 ? very light orange (under
    ↪ triangle)
RAD_CMAP.set_over(RAD_COLORS[-1])                          # > 37500 ? dark red (over triangle)

GEOM_COLOR = "steelblue"
GEOM_EDGE  = "steelblue"

mpl.rcParams.update({
    "text.usetex":      False,
    "font.family":      "serif",
    "mathtext.fontset": "cm",
    "font.size":         14,
    "axes.titlesize":    18,
    "axes.labelsize":    16,
    "xtick.labelsize":   13,
    "ytick.labelsize":   13,
    "legend.fontsize":   13,
    "figure.dpi":         300,
    "savefig.dpi":        300,
    "lines.linewidth":    1.0,
})

# =====
    ↪ =====
# HOLE DIAMETER LOOKUP
# =====
    ↪ =====

```

```

match (press_i, hole_i):
    case (0, 0): diam = 1.2
    case (0, 1): diam = 3.8
    case (0, 2): diam = 12.0
    case (1, 0): diam = 1.0
    case (1, 1): diam = 3.2
    case (1, 2): diam = 10.0
    case (2, 0): diam = 1.0
    case (2, 1): diam = 2.5
    case (2, 2): diam = 8.0

# =====
# ↪ =====
# HELPERS
# =====
# ↪ =====

def resolve_case_folders(case_folders, parent_folders, num_cases,
    ↪ case_labels):
    folders = []
    if case_folders:
        for p in case_folders:
            folders.append(Path(p))
    else:
        for parent in parent_folders:
            for i in range(1, num_cases + 1):
                folders.append(Path(parent) / f"case{i}")
    labelled = []
    for i, p in enumerate(folders):
        label = case_labels[i] if i < len(case_labels) else p.name
        labelled.append((label, p))
    return labelled

def load_radiation(case_path: Path):
    fpath = case_path / RAD_FILENAME
    if not fpath.exists():
        raise FileNotFoundError(f"Radiation file not found: {fpath}")
    r = r3d.read(str(fpath))
    if r.r3d_type == 4:
        r.to_type2()
    fieldnames = [f.strip() for f in r.fieldnames]
    try:
        idx = fieldnames.index(RAD_FIELDNAME)

```



```

except ValueError:
    print(f"    Available fields: {fieldnames}")
    raise ValueError(f"Field '{RAD_FIELDNAME}' not found in
        ↪ {fpath.name}")
field = r.fielfddata[idx].astype(float)
return field, np.asarray(r.xc), np.asarray(r.yc), np.asarray(r.zc)

def nearest_z_index(zc, z_target):
    return int(np.argmin(np.abs(zc - z_target)))

def build_exclusion_mask(xc, yc):
    nx, ny = len(xc), len(yc)
    mask = np.zeros((nx, ny), dtype=bool)
    for zone in EXCLUSION_ZONES:
        xi = (xc >= zone["x0"]) & (xc <= zone["x1"])
        yi = (yc >= zone["y0"]) & (yc <= zone["y1"])
        mask[np.ix_(xi, yi)] = True
    return mask

def compute_cell_areas(xc, yc):
    def midpoint_widths(c):
        w = np.zeros(len(c))
        w[0] = (c[1] - c[0])
        w[-1] = (c[-1] - c[-2])
        w[1:-1] = (c[2:] - c[:-2]) / 2.0
        return w
    return np.outer(midpoint_widths(xc), midpoint_widths(yc))

def compute_severity(field_2d, xc, yc, exclusion_mask):
    cell_areas = compute_cell_areas(xc, yc)
    valid = ~exclusion_mask
    score = 0.0
    exceedance = []
    for thresh, weight, desc, source in RADIATION_THRESHOLDS:
        above = (field_2d > thresh) & valid
        area = float(np.sum(cell_areas[above]))
        score += weight * area
        exceedance.append((thresh, area))
    return score, exceedance

```

```

def load_floorplan():
    if FLOORPLAN_JSON is None:
        return []
    p = Path(FLOORPLAN_JSON)
    if not p.exists():
        print(f"WARNING: {FLOORPLAN_JSON} not found - floor plan will not
            ↪ be drawn.")
        return []
    with open(p) as f:
        return json.load(f)

def draw_floorplan(ax, rects):
    for r in rects:
        patch = mpatches.Rectangle(
            (r["x0"], r["y0"]), r["x1"] - r["x0"], r["y1"] - r["y0"],
            linewidth=1.2, edgecolor=GEOM_EDGE, facecolor="none", zorder=2,
        )
        ax.add_patch(patch)

def draw_exclusion_zones(ax):
    for zone in EXCLUSION_ZONES:
        patch = mpatches.Rectangle(
            (zone["x0"], zone["y0"]), zone["x1"] - zone["x0"], zone["y1"] -
            ↪ zone["y0"],
            linewidth=1.0, edgecolor="gray", facecolor="gray",
            alpha=0.3, linestyle="--", zorder=3,
        )
        ax.add_patch(patch)

def process_cases(cases):
    results = []
    for label, case_path in cases:
        print(f"\nProcessing: {label} ({case_path})")

        if not case_path.exists():
            # ? add this
            print(f" SKIPPED (folder not found): {case_path}")
            continue

        try:
            field, xc, yc, zc = load_radiation(case_path)

```

```

except FileNotFoundError as e:
    print(f"  SKIPPED: {e}")
    continue
except Exception as e:                                     # ? catch any
    ↪ other load error
    print(f"  SKIPPED (load error): {e}")
    continue

iz      = nearest_z_index(zc, SLICE_Z)
z_actual = float(zc[iz])
print(f"  Z slice: requested {SLICE_Z:.2f} m ? nearest
    ↪ {z_actual:.3f} m (index {iz})")
field_2d      = field[:, :, iz]
exclusion_mask = build_exclusion_mask(xc, yc)
score, exceed  = compute_severity(field_2d, xc, yc, exclusion_mask)
results.append({
    "label":      label,
    "score":      score,
    "exceedance": exceed,
    "z_actual":   z_actual,
    "field_2d":   field_2d,
    "xc":         xc,
    "yc":         yc,
})
return results

# =====
↪ =====
# PRINT TABLE
# =====
↪ =====

def print_table(results, title=""):
    thresholds = [t for t, *_ in RADIATION_THRESHOLDS]
    col_w = 28 # wider to fit "scenario / label"
    header = f"{'Rank':<5} {'Case':<{col_w}} {'Score':>10}"
    for t in thresholds:
        header += f" {'>' + str(t) + ' W/m²':>12}"

    sep = "=" * len(header)
    if title:
        print(f"\n{sep}\n {title}\n{sep}")
    else:
        print(f"\n{sep}")

```

```

print(header)
print("-" * len(header))
print("  Weights:", ", ".join(f"{t} W/m2×{w}" for t, w, *_ in
    ↪ RADIATION_THRESHOLDS))
print("  Sources:", "; ".join(f"{t} W/m2: {src}" for t, _, __, src in
    ↪ RADIATION_THRESHOLDS))
print("-" * len(header))

for rank, r in enumerate(results, 1):
    row = f"{rank:<5} {r['label']:<{col_w}} {r['score']:>10.2f}"
    for _, area in r["exceedance"]:
        row += f"  {area:>11.2f}m2"
    row += f"    (Z={r['z_actual']:.2f}m)"
    print(row)

print(sep)
print("Score = weighted sum of exceedance areas (m2), excluding
    ↪ exclusion zones.")
print("Higher score = more severe.\n")

def save_table_csv(results_and_labels, filepath):
    """
    Save ranked radiation results to CSV.
    results_and_labels: list of (results, scenario_label) tuples.
    Works for one or two scenarios.
    """
    import csv
    rows = []
    for results, scenario_label in results_and_labels:
        for rank, r in enumerate(results, 1):
            row = {
                "scenario": scenario_label,
                "rank": rank,
                "label": r["label"],
                "score": r["score"],
                "z_actual": r["z_actual"],
            }
            for (thresh, *_), (_, area) in zip(RADIATION_THRESHOLDS,
                ↪ r["exceedance"]):
                row[f"area_above_{thresh}_W/m2"] = area
            rows.append(row)

    if not rows:
        print("  No results to save.")

```

```

        return

    filepath = Path(filepath)
    filepath.parent.mkdir(parents=True, exist_ok=True)
    with open(filepath, "w", newline="") as f:
        writer = csv.DictWriter(f, fieldnames=rows[0].keys())
        writer.writeheader()
        writer.writerows(rows)
    print(f"CSV saved ? {filepath}")

# =====
# PLOT HELPERS
# =====

def _render_subplot(ax, res, title_str, floorplan):
    field_2d = res["field_2d"]
    xc, yc = res["xc"], res["yc"]
    X, Y = np.meshgrid(xc, yc, indexing="ij")

    cf = ax.contourf(X, Y, field_2d,
                    levels=RAD_LEVELS,
                    cmap=RAD_CMAP, norm=RAD_NORM,
                    extend="both",
                    alpha=0.75, zorder=1)
    ax.contour(X, Y, field_2d, levels=RAD_LEVELS,
              colors="black", linewidths=0.8, zorder=4)
    draw_floorplan(ax, floorplan)
    draw_exclusion_zones(ax)
    ax.set_xlim(-15, 15)
    ax.set_ylim(-15, 7)
    ax.set_title(title_str, fontsize=13)
    ax.set_aspect("equal")
    return cf    # no colorbar here - caller handles it

def _add_colorbar(fig, gs, cf):
    cax = fig.add_subplot(gs[:, -1])
    cb = fig.colorbar(cf, cax=cax, extend="both")
    cb.set_ticks(RAD_LEVELS)
    cb.set_ticklabels([f"{t:," for t in RAD_LEVELS])
    cb.set_label("W/m2", fontsize=13)
    cb.ax.tick_params(labelsize=11)

```

```

# =====
↳ =====
# SINGLE-SCENARIO PLOT (original layout, global ranking within one
↳ scenario)
# =====
↳ =====

def plot_single(results, floorplan):
    n = len(results)
    ncols = min(n, 2)
    nrows = int(np.ceil(n / ncols))

    fig = plt.figure(figsize=(7 * ncols + 1, 6 * nrows))
    gs = GridSpec(nrows, ncols + 1, figure=fig,
                  width_ratios=[1] * ncols + [0.04],
                  wspace=0.15, hspace=0.25)

    cf_last = None
    for rank, res in enumerate(results, 1):
        row, col = divmod(rank - 1, ncols)
        ax = fig.add_subplot(gs[row, col])
        cf_last = _render_subplot(ax, res,
                                  f"#{rank} {res['label']}"
                                  ↳ (score={res['score']:.1f})),
                                  floorplan)
        ax.set_ylabel("Y (m)" if col == 0 else "")
        ax.set_xlabel("X (m)" if row == nrows - 1 else "")
        if col != 0:
            ax.set_yticklabels([])
        if row != nrows - 1:
            ax.set_xticklabels([])

    for idx in range(n, n * nrows):
        row, col = divmod(idx, n * nrows)
        fig.add_subplot(gs[row, col]).set_visible(False)

    _add_colorbar(fig, gs, cf_last)

    fig.suptitle(
        f"Radiation comparison | {pressure[press_i]}, {diam} mm |"
        ↳ ranked by severity",
        fontsize=16, y=.91,
    )

```

```

    if savePlot:
        fig.savefig(out_png, bbox_inches="tight")
        print(f"Saved ? {out_png}")
    if showPlot:
        plt.show()
    plt.close(fig)

# =====
# TWO-SCENARIO PLOT (global ranking, row-major layout)
# =====

def plot_comparison(results_A, results_B, floorplan):
    for r in results_A:
        r["scenario"] = SCENARIO_A_LABEL
    for r in results_B:
        r["scenario"] = SCENARIO_B_LABEL

    combined = sorted(results_A + results_B, key=lambda r: r["score"],
        ↪ reverse=True)
    n = len(combined)
    ncols = 2
    nrows = int(np.ceil(n / ncols))

    fig = plt.figure(figsize=(15, 6 * nrows))
    gs = GridSpec(nrows, ncols + 1, figure=fig,
        width_ratios=[1, 1, 0.05],
        wspace=0.18, hspace=0.05)

    cf_last = None
    for rank, res in enumerate(combined, 1):
        row, col = divmod(rank - 1, ncols)
        ax = fig.add_subplot(gs[row, col])
        title_str = (
            f"#{rank}  [{res['scenario']}] \n"
            f"{res['label']} (score={res['score']:.1f})"
        )
        cf_last = _render_subplot(ax, res, title_str, floorplan)
    if col != 0:
        ax.set_ylabel("")
        ax.set_yticklabels([])
    else:
        ax.set_ylabel("Y (m)")

```

```

        if row != nrows - 1:
            ax.set_xlabel("")
            ax.set_xticklabels([])
        else:
            ax.set_xlabel("X (m)")

cax = fig.add_subplot(gs[:, -1])
cb = fig.colorbar(cf_last, cax=cax)
cb.set_ticks(RAD_LEVELS)
cb.set_ticklabels([f"{t:," for t in RAD_LEVELS])
cb.set_label("W/m2", fontsize=13)
cb.ax.tick_params(labelsize=11)

fig.suptitle(
    f"Radiation comparison | {SCENARIO_A_LABEL} vs {SCENARIO_B_LABEL}
    ↪ "
    f"| {pressure[press_i]}, {diam} mm | global ranking",
    fontsize=16, y=.92
)
if savePlot:
    fig.savefig(out_png, bbox_inches="tight")
    print(f"Saved ? {out_png}")
if showPlot:
    plt.show()
plt.close(fig)

# =====
↪ =====
# MAIN
# =====
↪ =====

def main():
    floorplan = load_floorplan()

    cases_A = resolve_case_folders(CASE_FOLDERS_A, PARENT_FOLDERS_A,
    ↪ NUM_CASES, CASE_LABELS_A)
    results_A = process_cases(cases_A)

    if not results_A:
        print("No cases loaded for Scenario A: check paths.")
        return

    if not COMPARE_TWO_SCENARIOS:

```



```

results_A.sort(key=lambda r: r["score"], reverse=True)
print_table(results_A, title="Global ranking")
save_table_csv([(results_A, SCENARIO_A_LABEL)], saveCSV)
plot_single(results_A, floorplan)

else:
    cases_B = resolve_case_folders(CASE_FOLDERS_B, PARENT_FOLDERS_B,
        ↪ NUM_CASES_B, CASE_LABELS_B)
    results_B = process_cases(cases_B)

    if not results_B:
        print("No cases loaded for Scenario B: check paths.")
        return

    # Tag before merging so the table also shows scenario provenance
    for r in results_A:
        r["scenario"] = SCENARIO_A_LABEL
    for r in results_B:
        r["scenario"] = SCENARIO_B_LABEL

    combined = sorted(results_A + results_B, key=lambda r: r["score"],
        ↪ reverse=True)

    # Widen label column to fit "scenario / label"
    print_table(combined, title=f"Global ranking | {SCENARIO_A_LABEL}
        ↪ vs {SCENARIO_B_LABEL}")
    save_table_csv([(results_A, SCENARIO_A_LABEL),
        (results_B, SCENARIO_B_LABEL)], saveCSV)
    plot_comparison(results_A, results_B, floorplan)

if __name__ == "__main__":
    main()

```

B.5 Consolidated analysis

B.5.1 Main

```

"""
consolidated_analysis.py - KFX simulation results consolidation and
    ↪ analysis
=====
    ↪ =====
This file contains ONLY configuration and the main loop.
All computation is in kfx_scores.py.
All plotting is in kfx_plots.py.
All r3d I/O is in kfx_utils.py.

To add a new metric: add a function to kfx_scores.py and call it here.
To change a plot: edit kfx_plots.py.
To change physics: edit kfx_scores.py.
"""

from pathlib import Path
import numpy as np
import pandas as pd
import matplotlib as mpl

from kfx_scores import (compute_radiation_score_full,
                        compute_flammable_area_full,
                        ↪ compute_flammable_volume,
                        compute_dynpres_score)
from kfx_plots import (plot_heatmap, plot_scatter_matrix,
                        ↪ plot_main_effects,
                        plot_areaBars, plot_flamvol_bars,
                        plot_rad_bars_grouped, plot_rad_bars_stacked,
                        ↪ plot_combined_score_bars)

# =====
    ↪ =====
# CONFIGURATION
# =====
    ↪ =====

# --- Plot toggles
    ↪ -----
PLOT_HEATMAP = 0
PLOT_SCATTER = 0

```

```

PLOT_ANOVA = 0
PLOT_AREA_BARS = 1
PLOT_AREA_BARS_HOLE_FIRST = 1
PLOT_RAD_BARS_GROUP = 1
PLOT_RAD_BARS_STACK = 1
PLOT_RAD_BARS_GROUP_HOLE_FIRST = 1
PLOT_RAD_BARS_STACK_HOLE_FIRST = 1
PLOT_FLAMVOL_BARS = 1
PLOT_FLAMVOL_BARS_HOLE_FIRST = 1
PLOT_COMBINED_BARS = 1

savePlot = 1
showPlot = 0

# --- Domain structure
↳ -----
BASE_PATH = Path("/proj/P10475_RA/STUDENTS/2026/Nur/simulations")
VENT_TYPES = ["naturalVent", "forcedVent"]
PRESSURES_FIRE = ["430bar-fire", "630bar-fire", "900bar-fire"]
PRESSURES = ["430bar", "630bar", "900bar"]
HOLE_CATS = ["1area", "10area", "fullBore"]
NUM_CASES = 3

# Leak direction labels per pressure level (same order as
↳ case1/case2/case3)
DIR_LABELS_BY_PRESS = [
    ["-y", "+z", "-z"], # 430bar-fire
    ["+x", "-x", "-y"], # 630bar-fire
    ["-x", "+y", "-y"], # 900bar-fire
]

# Leak source position (x, y) in metres - one per pressure level
LEAK_POSITIONS = {
    "430bar-fire": (-2.5, -3.75),
    "630bar-fire": (-2.5, -1.7),
    "900bar-fire": (-2.5, 0.4), # update with actual positions
}

# --- Cases to skip
↳ -----
SKIP_CASES = [
    #"naturalVent/900bar-fire/10area/case3",
    # "naturalVent/430bar-fire/fullBore/case2",
]

```

```

# --- R3D filenames
↳ -----
R3D_RADIATION = "case_bmg_exit.r3d"
R3D_FLAMMABLE = "case_std_exit.r3d"
R3D_DYNPRES   = "case_std_exit.r3d"
R3D_FLAM_VOL  = "case_std_exit.r3d"

FLOORPLAN_JSON = "floorplan.json"

# =====
↳ =====
# RADIATION CONFIG
# =====
↳ =====
RAD_FIELDNAME = "Radiation(W/m2)"
RAD_SLICE_Z   = 1.5    # m

RADIATION_THRESHOLDS = [
    (1500, 1, "No damage on long exposures"),
    (5000, 3, "Second-degree burns if exposure is longer than 30s"),
    (17000, 8, "FAA aircraft escape route"),
    (37500, 20, "Immediate fatality"),
]

RAD_EXCLUSION_ZONES = [
    {"x0": -3.5, "y0": -4.0, "x1": -1.5, "y1": 1.0},
]

# Colourmap for radiation plots - one colour per interval (including
↳ under/over)
# Intervals: <1500 | 1500-2500 | 2500-5000 | 5000-12500 | >12500
RAD_COLORS = [
    "#fee8c8",    # < 1500    W/m2
    "#fdbb84",    # 1500-5000 W/m2
    "#fc8d59",    # 5000-17000 W/m2
    "#d7301f",    # 17000-37500 W/m2
    "#7f0000",    # > 37500    W/m2
]

# =====
↳ =====
# FLAMMABLE AREA CONFIG

```

```

# =====
↳ =====
LFL = 4.0      # vol %
UFL = 75.0     # vol %

# =====
↳ =====
# DYNAMIC PRESSURE CONFIG
# =====
↳ =====
P_ATM          = 101325.0    # Pa
R_AIR          = 287.058    # J/kg/K
R_H2           = 4157.0     # J/kg/K
DP_SLICE_Z     = 0.1        # m - ground level
SCORE_PERCENTILE = 99.9

DP_ZONES = [
    {"label": "container_interior", "x0": -3.5, "y0": -4.0,
     "x1": -1.5, "y1": 1.0, "weight": 0.0},
    {"label": "firewall_zone_1", "x0": -8.0, "y0": -8.0,
     "x1": 8.0, "y1": 4.0, "weight": 0.3},
    {"label": "firewall_zone_2", "x0": 2.0, "y0": 4.0,
     "x1": 6.0, "y1": 8.0, "weight": 0.3},
    {"label": "outside", "x0": -999, "y0": -999,
     "x1": 999, "y1": 999, "weight": 1.0},
]

# =====
↳ =====
# COMBINED RANKING CONFIG
# =====
↳ =====
METRIC_WEIGHTS = {
    "radiation_score": 1/4,
    "flammable_area": 1/4,
    "dynpres_score": 1/4,
    "flammable_volume": 1/4,
}

# =====
↳ =====
# PLOT STYLING
# =====
↳ =====

```

```

MODE_BAR          = "per_metric"          # "per_metric" or "per_parameter"
COLOR_NAT         = "#4363d8"            # natural ventilation bar colour
COLOR_FORCED      = "#e6194b"            # forced ventilation bar colour
HATCH_FORCED      = "//"                 # hatch pattern for forced vent bars
PRESSURE_BG_COLORS = ["#fff7ec", "#fee8c8", "#fdd49e"] # one per pressure

SAVE_DIR          = Path("/proj/P10475_RA/STUDENTS/2026/Nur/img/plots/consolidatedAnalysis")
SAVE_CSV          = SAVE_DIR / "all_scores.csv"
SAVE_HEATMAP      = SAVE_DIR / "heatmap_ranking.png"
SAVE_SCATTER      = SAVE_DIR / "scatter_metric_correlation.png"
SAVE_AREA_BARS    = SAVE_DIR / "summary_area_bars.png"
SAVE_RAD_GROUPED  = SAVE_DIR / "summary_rad_bars_grouped.png"
SAVE_RAD_STACKED  = SAVE_DIR / "summary_rad_bars_stacked.png"
SAVE_AREA_BARS_HOLE = SAVE_DIR / "summary_area_bars_holefirst.png"
SAVE_RAD_GROUP_HOLE = SAVE_DIR / "summary_rad_bars_grouped_holefirst.png"
SAVE_RAD_STACK_HOLE = SAVE_DIR / "summary_rad_bars_stacked_holefirst.png"
SAVE_FLAM_VOL_BARS = SAVE_DIR / "summary_flamvol_bars.png"
SAVE_FLAM_VOL_BARS_HOLE = SAVE_DIR / "summary_flamvol_bars_holefirst.png"
SAVE_COMBINED_BARS = SAVE_DIR / "combined_score_bars.png"

mpl.rcParams.update({
    "text.usetex":      False,
    "font.family":      "serif",
    "mathtext.fontset": "cm",
    "font.size":        13,
    "axes.titlesize":   15,
    "axes.labelsize":   14,
    "xtick.labelsize":  11,
    "ytick.labelsize":  11,
    "legend.fontsize":  11,
    "figure.dpi":       150,
    "savefig.dpi":      300,
})

# =====
# MAIN LOOP
# =====

def compute_all_scores():
    """
    Single loop over all parameter combinations.

```

```

Radiation and dynamic pressure use PRESSURES_FIRE folder names.
Flammable area and volume use PRESSURES folder names (no -fire
    ↪ suffix).
Both are stored in the same row, keyed by the fire pressure string
so the df stays consistent for plotting.
"""
rows = []

for vent in VENT_TYPES:
    for press_fire, press_plain in zip(PRESSURES_FIRE, PRESSURES):
        for hole in HOLE_CATS:
            for case_n in range(1, NUM_CASES + 1):

                rel_id = f"{vent}/{press_fire}/{hole}/case{case_n}"

                if rel_id in SKIP_CASES:
                    print(f"  SKIPPED (skip list): {rel_id}")
                    continue

                # paths for the two folder conventions
                path_fire = BASE_PATH / vent / press_fire / hole /
                    ↪ f"case{case_n}"
                path_plain = BASE_PATH / vent / press_plain / hole /
                    ↪ f"case{case_n}"

                # skip only if BOTH folders are missing
                if not path_fire.exists() and not path_plain.exists():
                    print(f"  SKIPPED (folders not found): {rel_id}")
                    continue

                print(f"  Processing: {rel_id}")
                row = {
                    "id": rel_id,
                    "vent_type": vent,
                    "pressure": press_fire,  # always use fire
                    ↪ string as key
                    "hole_size": hole,
                    "case_n": case_n,
                }

                # --- radiation (fire folder) ---
                fpath = path_fire / R3D_RADIATION
                if not path_fire.exists() or not fpath.exists():
                    print(f"    SKIPPED radiation (not found)")

```

```

        row["radiation_score"] = np.nan
        for thresh, *_ in RADIATION_THRESHOLDS:
            row[f"exceed_{thresh}"] = np.nan
    else:
        try:
            score, exceed = compute_radiation_score_full(
                path_fire, R3D_RADIATION, RAD_FIELDNAME,
                RAD_SLICE_Z, RAD_EXCLUSION_ZONES,
                RADIATION_THRESHOLDS)
            row["radiation_score"] = score
            for thresh, *_ in RADIATION_THRESHOLDS:
                row[f"exceed_{thresh}"] =
                    ↪ exceed.get(thresh, 0.0)
            print(f"    radiation_score: {score:.4f}")
        except Exception as e:
            print(f"    ERROR radiation: {e}")
            row["radiation_score"] = np.nan
            for thresh, *_ in RADIATION_THRESHOLDS:
                row[f"exceed_{thresh}"] = np.nan

# --- dynamic pressure (fire folder) ---
fpath = path_fire / R3D_DYNPRES
if not path_fire.exists() or not fpath.exists():
    print(f"    SKIPPED dynamic pressure (not found)")
    row["dynpres_score"] = np.nan
else:
    try:
        score = compute_dynpres_score(
            path_fire, R3D_DYNPRES, DP_SLICE_Z,
            SCORE_PERCENTILE, DP_ZONES,
            P_ATM, R_AIR, R_H2)
        row["dynpres_score"] = score
        print(f"    dynpres_score: {score:.4f}")
    except Exception as e:
        print(f"    ERROR dynamic pressure: {e}")
        row["dynpres_score"] = np.nan

# --- flammable area (plain folder) ---
fpath = path_plain / R3D_FLAMMABLE
if not path_plain.exists() or not fpath.exists():
    print(f"    SKIPPED flammable area (not found)")
    row["flammable_area"] = np.nan
    row["max_distance"] = np.nan
else:

```



```

        try:
            leak_pos = LEAK_POSITIONS.get(press_fire, (0.0,
                ↪ 0.0))
            area, max_dist = compute_flammable_area_full(
                path_plain, R3D_FLAMMABLE, LFL, UFL,
                ↪ leak_pos)
            row["flammable_area"] = area
            row["max_distance"] = max_dist
            print(f"    flammable_area: {area:.4f} "
                f"max_distance: {max_dist:.4f}")
        except Exception as e:
            print(f"    ERROR flammable area: {e}")
            row["flammable_area"] = np.nan
            row["max_distance"] = np.nan

    # --- flammable volume (plain folder) ---
    fpath = path_plain / R3D_FLAM_VOL
    if not path_plain.exists() or not fpath.exists():
        print(f"    SKIPPED flammable volume (not found)")
        row["flammable_volume"] = np.nan
    else:
        try:
            vol = compute_flammable_volume(
                path_plain, R3D_FLAM_VOL, LFL, UFL)
            row["flammable_volume"] = vol
            print(f"    flammable_volume: {vol:.4f} m3")
        except Exception as e:
            print(f"    ERROR flammable volume: {e}")
            row["flammable_volume"] = np.nan

    rows.append(row)

df = pd.DataFrame(rows)

for metric in ["radiation_score", "flammable_area",
               "flammable_volume", "dynpres_score"]:
    col_min = df[metric].min()
    col_max = df[metric].max()
    rng = col_max - col_min
    df[f"{metric}_norm"] = (
        (df[metric] - col_min) / rng if rng > 0 else 0.0
    )

df["combined_score"] = sum(

```

```

        METRIC_WEIGHTS[m] * df[f"{m}_norm"].fillna(0)
    for m in METRIC_WEIGHTS
)

df = df.sort_values("combined_score",
    ↪ ascending=False).reset_index(drop=True)
df.insert(0, "rank", range(1, len(df) + 1))
return df

def print_table(df):
    """Print ranked results table to stdout."""
    header = (f"{'Rank':<5} {'Case ID':<50} {'Radiation':>12}"
              f" {'Flam. Area':>12} {'Flam. Vol':>12}"
              f" {'Dyn. Pres.':>12} {'Combined':>10}")
    sep = "=" * len(header)
    print(f"\n{sep}\n Consolidated hazard ranking\n{sep}")
    print(header)
    print("-" * len(header))
    for _, r in df.iterrows():
        print(f" {int(r['rank']):<4} {r['id']:<50} "
              f"{r['radiation_score']:>12.2f} {r['flammable_area']:>12.4f}"
              f" {r['flammable_volume']:>12.4f}"
              ↪ {r['dynpres_score']:>12.2f}"
              f" {r['combined_score']:>10.4f}")
    print(sep)

def save_csv(df):
    """Save full results table to CSV."""
    SAVE_DIR.mkdir(parents=True, exist_ok=True)
    df.to_csv(SAVE_CSV, index=False)
    print(f"\nResults saved in {SAVE_CSV}")

# =====
# ↪ =====
# MAIN
# =====
# ↪ =====

def main():
    print("\n" + "=" * 60)
    print(" Computing all hazard scores across full parameter space")

```

```
print("=" * 60)

df = compute_all_scores()
print_table(df)
save_csv(df)

print("\n" + "=" * 60)
print("  Generating analysis plots")
print("=" * 60)

if PLOT_HEATMAP:
    plot_heatmap(df, SAVE_HEATMAP, savePlot, showPlot)

if PLOT_COMBINED_BARS:
    plot_combined_score_bars(df, SAVE_COMBINED_BARS, savePlot, showPlot)

if PLOT_SCATTER:
    plot_scatter_matrix(df, SAVE_SCATTER, savePlot, showPlot)

if PLOT_ANOVA:
    plot_main_effects(df, MODE_BAR, VENT_TYPES, PRESSURES_FIRE,
        ↪ HOLE_CATS,
        SAVE_DIR, savePlot, showPlot)

if PLOT_AREA_BARS:
    plot_area_bars(df, PRESSURES_FIRE, HOLE_CATS, NUM_CASES,
        DIR_LABELS_BY_PRESS, COLOR_NAT, COLOR_FORCED,
        HATCH_FORCED, PRESSURE_BG_COLORS,
        SAVE_AREA_BARS, savePlot, showPlot,
        hierarchy="press_first")

if PLOT_AREA_BARS_HOLE_FIRST:
    plot_area_bars(df, PRESSURES_FIRE, HOLE_CATS, NUM_CASES,
        DIR_LABELS_BY_PRESS, COLOR_NAT, COLOR_FORCED,
        HATCH_FORCED, PRESSURE_BG_COLORS,
        SAVE_AREA_BARS_HOLE, savePlot, showPlot,
        hierarchy="hole_first")

if PLOT_FLAMVOL_BARS:
    plot_flamvol_bars(df, PRESSURES_FIRE, HOLE_CATS, NUM_CASES,
        DIR_LABELS_BY_PRESS, COLOR_NAT, COLOR_FORCED,
        HATCH_FORCED, PRESSURE_BG_COLORS,
        SAVE_FLAM_VOL_BARS, savePlot, showPlot,
        hierarchy="press_first")
```

```

if PLOT_FLAMVOL_BARS_HOLE_FIRST:
    plot_flamvol_bars(df, PRESSURES_FIRE, HOLE_CATS, NUM_CASES,
                      DIR_LABELS_BY_PRESS, COLOR_NAT, COLOR_FORCED,
                      HATCH_FORCED, PRESSURE_BG_COLORS,
                      SAVE_FLAM_VOL_BARS_HOLE, savePlot, showPlot,
                      hierarchy="hole_first")

if PLOT_RAD_BARS_GROUP or PLOT_RAD_BARS_STACK or \
PLOT_RAD_BARS_GROUP_HOLE_FIRST or PLOT_RAD_BARS_STACK_HOLE_FIRST:

    if PLOT_RAD_BARS_GROUP:
        plot_rad_bars_grouped(df, PRESSURES_FIRE, HOLE_CATS, NUM_CASES,
                              DIR_LABELS_BY_PRESS, RAD_COLORS,
                              RADIATION_THRESHOLDS, HATCH_FORCED,
                              PRESSURE_BG_COLORS, SAVE_RAD_GROUPED,
                              savePlot, showPlot,
                              ↪ hierarchy="press_first")

    if PLOT_RAD_BARS_STACK:
        plot_rad_bars_stacked(df, PRESSURES_FIRE, HOLE_CATS, NUM_CASES,
                              DIR_LABELS_BY_PRESS, RAD_COLORS,
                              RADIATION_THRESHOLDS, HATCH_FORCED,
                              PRESSURE_BG_COLORS, SAVE_RAD_STACKED,
                              savePlot, showPlot,
                              ↪ hierarchy="press_first")

    if PLOT_RAD_BARS_GROUP_HOLE_FIRST:
        plot_rad_bars_grouped(df, PRESSURES_FIRE, HOLE_CATS, NUM_CASES,
                              DIR_LABELS_BY_PRESS, RAD_COLORS,
                              RADIATION_THRESHOLDS, HATCH_FORCED,
                              PRESSURE_BG_COLORS, SAVE_RAD_GROUP_HOLE,
                              savePlot, showPlot,
                              ↪ hierarchy="hole_first")

    if PLOT_RAD_BARS_STACK_HOLE_FIRST:
        plot_rad_bars_stacked(df, PRESSURES_FIRE, HOLE_CATS, NUM_CASES,
                              DIR_LABELS_BY_PRESS, RAD_COLORS,
                              RADIATION_THRESHOLDS, HATCH_FORCED,
                              PRESSURE_BG_COLORS, SAVE_RAD_STACK_HOLE,
                              savePlot, showPlot,
                              ↪ hierarchy="hole_first")

```

```
if __name__ == "__main__":  
    main()
```

B.5.2 Plots

```

"""
kfx_plots.py - analysis and summary plot functions
=====
All matplotlib figure-building logic lives here.
Configuration (colours, save paths, labels) is passed in as arguments.

Imported by consolidated_analysis.py.

spacing configuration details:
fig.subplots_adjust(
    top=0.93,      # space for supitle / lower = more space above plots
    bottom=0.18,   # space for legend and x-axis labels / higher = more
                  ↪ space below
    hspace=0.6,    # vertical gap between subplots / higher = more space
    left=0.10,     # left margin
    right=0.97,    # right margin / lower if legend goes outside right
)
"""

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from matplotlib.lines import Line2D
from matplotlib.patches import Patch
from matplotlib.gridspec import GridSpec

def _fmt_pressure(p):
    """Remove -fire suffix from pressure strings for display."""
    return p.replace("-fire", "")

# =====
# ↪ =====
# HEATMAP
# =====
# ↪ =====

def plot_heatmap(df, save_path, savePlot, showPlot):
    """
    Heatmap of normalised scores for all cases.
    Rows = cases sorted by combined score, columns = metrics.
    Cell colour encodes normalised value in [0, 1].
    """

```

```

metrics      = ["radiation_score_norm", "flammable_area_norm",
                "flammable_volume_norm", "dynpres_score_norm",
                "combined_score"]
metric_labels = ["Radiation", "Flam. Area", "Flam. Volume",
                "Dyn. Pressure", "Combined"]

data      = df[metrics].values
case_ids  = df["id"].tolist()

fig, ax = plt.subplots(figsize=(8, max(6, len(case_ids) * 0.3)))
im      = ax.imshow(data, aspect="auto", cmap="YlOrRd",
                    vmin=0, vmax=1, interpolation="nearest")

case_ids = [_fmt_pressure(cid) for cid in df["id"].tolist()]

ax.set_xticks(range(len(metrics)))
ax.set_xticklabels(metric_labels, rotation=20, ha="right")
ax.set_yticks(range(len(case_ids)))
ax.set_yticklabels([f"#{i+1} {cid}" for i, cid in enumerate(case_ids)],
                    fontsize=8)
ax.set_title("Normalised hazard scores: all cases", fontsize=14)

cb = plt.colorbar(im, ax=ax, shrink=0.6, pad=0.02)
cb.set_label("Normalised score [0,1]", fontsize=11)

plt.tight_layout()
if savePlot:
    save_path.parent.mkdir(parents=True, exist_ok=True)
    fig.savefig(save_path, dpi=300, bbox_inches="tight")
    print(f"Saved in {save_path}")
if showPlot:
    plt.show()
plt.close(fig)

# =====
# COMBINED SCORE BARS
# =====

def plot_combined_scoreBars(df, save_path, savePlot, showPlot):
    """
    Bar chart of combined hazard score for all cases, ranked highest to
    lowest.

```

```

x-axis: case ID (cleaned, rotated 90 degrees).
y-axis: combined normalised score.
Colour encodes ventilation type.
"""

# df is already sorted by combined_score descending
n      = len(df)
x_pos  = np.arange(n)
bar_w  = 0.7

# clean pressure labels for display
def _clean_id(id_str):
    """Shorten case ID strings for use in plot labels."""
    return (id_str
            .replace("-fire", "")
            .replace("naturalVent/", "")
            .replace("forcedVent/", "")
            )

vent_colors = {
    "naturalVent": "#4363d8",
    "forcedVent":  "#e6194b",
}

fig, ax = plt.subplots(figsize=(max(12, n * 0.35), 6))

for i, (_, row) in enumerate(df.iterrows()):
    color = vent_colors.get(row["vent_type"], "gray")
    ax.bar(i, row["combined_score"], width=bar_w,
           color=color, edgecolor="black",
           linewidth=0.5, alpha=0.85, zorder=2)

ax.set_xticks(x_pos)
ax.set_xticklabels([f"#{int(r['rank'])} {_clean_id(r['id'])}"
                    for _, r in df.iterrows()],
                    rotation=90, ha="center", fontsize=10)
ax.set_ylabel("Combined normalised hazard score")
ax.set_title("Combined hazard ranking | all cases")
ax.set_xlim(-0.5, n - 0.5)
ax.set_ylim(0, 1.05)
ax.grid(axis="y", linestyle="--", alpha=0.5)

# legend for ventilation type
from matplotlib.patches import Patch
legend_handles = [

```



```

        Patch(facecolor=vent_colors["naturalVent"], edgecolor="black",
              label="Natural ventilation"),
        Patch(facecolor=vent_colors["forcedVent"], edgecolor="black",
              label="Forced ventilation"),
    ]
    ax.legend(handles=legend_handles, framealpha=0.9,
              loc="upper right", title="Ventilation type")

plt.tight_layout()

if savePlot:
    save_path.parent.mkdir(parents=True, exist_ok=True)
    fig.savefig(save_path, dpi=300, bbox_inches="tight")
    print(f"Saved -> {save_path}")
if showPlot:
    plt.show()
plt.close(fig)

# =====
# SCATTER MATRIX
# =====

def plot_scatter_matrix(df, save_path, savePlot, showPlot,
    vent_colors=None):
    """
    Scatter matrix for three hazard metrics.
    Off-diagonal: scatter of metric[row] vs metric[col], coloured by vent
    type.
    Diagonal: metric name as a text label.
    Pearson r annotated on each off-diagonal panel.
    """
    if vent_colors is None:
        vent_colors = {"naturalVent": "#e6194b", "forcedVent": "#4363d8"}

    metrics = ["radiation_score", "flammable_area", "flammable_volume",
        "dynpres_score"]
    labels = ["Radiation score (m2)", "Flammable area (m2)", "Flammable
        volume (m3)", "Dyn. pressure score (Pa)"]
    n = len(metrics)

    fig, axes = plt.subplots(n, n, figsize=(4 * n, 4 * n))

```

```

for i in range(n):
    for j in range(n):
        ax = axes[i][j]

        if i == j:
            ax.text(0.5, 0.5, labels[i],
                    ha="center", va="center",
                    transform=ax.transAxes,
                    fontsize=12, fontweight="bold")
            ax.set_axis_off()
        else:
            for vent, grp in df.groupby("vent_type"):
                ax.scatter(grp[metrics[j]], grp[metrics[i]],
                           color=vent_colors.get(vent, "gray"),
                           alpha=0.7, s=30)
            valid = df[[metrics[i], metrics[j]]].dropna()
            if len(valid) > 2:
                r_val = np.corrcoef(valid[metrics[i]],
                                    valid[metrics[j]])[0, 1]
                ax.annotate(f"r = {r_val:.2f}", xy=(0.05, 0.92),
                            xycoords="axes fraction", fontsize=10)

            if j == 0 and i != j:
                ax.set_ylabel(labels[i], fontsize=10)
            elif i != j:
                ax.set_yticklabels([])
            if i == n - 1 and i != j:
                ax.set_xlabel(labels[j], fontsize=10)
            elif i != j:
                ax.set_xticklabels([])

handles = [
    Line2D([0], [0], marker="o", color="w",
            markerfacecolor=c, markersize=8, label=v)
    for v, c in vent_colors.items()
]
fig.legend(handles=handles, loc="upper right",
           title="Ventilation", fontsize=10, framealpha=0.9)
fig.suptitle("Metric correlations across all cases", fontsize=14,
             ↪ y=1.01)
plt.tight_layout()

if savePlot:

```

```

        save_path.parent.mkdir(parents=True, exist_ok=True)
        fig.savefig(save_path, dpi=300, bbox_inches="tight")
        print(f"Saved in {save_path}")
    if showPlot:
        plt.show()
    plt.close(fig)

# =====
# MAIN EFFECTS BAR CHARTS -- one-way anova
# =====

def plot_main_effects(df, mode, vent_types, pressures, hole_cats,
                      save_dir, savePlot, showPlot):
    """
    Bar charts of mean score at each level of each parameter.
    Individual data points overlaid as scatter -> no error bars.
    mode = 'per_metric' -> one figure per metric, subplots per
        ↪ parameter
    mode = 'per_parameter' -> one figure per parameter, subplots per
        ↪ metric
    """
    metrics = ["radiation_score", "flammable_area",
               "flammable_volume", "dynpres_score"]
    metric_labels = [r"Radiation score (m$^2$)", r"Flammable area (m$^2$)",
                     r"Flammable volume (m$^3$)", "Dynamic pressure score
                     ↪ (Pa)"]
    parameters = ["vent_type", "pressure", "hole_size"]
    param_labels = ["Ventilation type", "Pressure", "Hole size"]
    param_levels = {"vent_type": vent_types,
                    "pressure": pressures,
                    "hole_size": hole_cats}
    bar_colors = ["#e6194b", "#3cb44b", "#4363d8", "#f58231"]

    save_dir.mkdir(parents=True, exist_ok=True)

    if mode == "per_metric":
        for metric, m_label in zip(metrics, metric_labels):
            fig, axes = plt.subplots(1, len(parameters),
                                    figsize=(5 * len(parameters), 5))
            for p_idx, (param, p_label) in enumerate(
                zip(parameters, param_labels)):

```

```

ax      = axes[p_idx]
levels = param_levels[param]
means  = [df.groupby(param)[metric].mean().get(lv, np.nan)
          for lv in levels]

# bars show group mean only -> no error bars
ax.bar(range(len(levels)), means,
       color=bar_colors[:len(levels)], alpha=0.6, zorder=1)

# individual points overlaid with jitter
for lv_idx, lv in enumerate(levels):
    vals = df[df[param] == lv][metric].dropna().values
    jitter = np.random.uniform(-0.15, 0.15, size=len(vals))
    ax.scatter(np.full(len(vals), lv_idx) + jitter, vals,
               color="black", s=18, alpha=0.6, zorder=3)

ax.set_xticks(range(len(levels)))
ax.set_xticklabels(levels, rotation=0, ha="center")
#ax.set_xlabel(p_label)
ax.set_ylabel(m_label if p_idx == 0 else "")
ax.set_title(_fmt_pressure(p_label))
ax.grid(axis="y", linestyle="--", alpha=0.5)

fig.suptitle(f"One-way ANOVA | {m_label}", fontsize=14)
plt.tight_layout()
fname = save_dir / f"main_effects_{metric}.png"
if savePlot:
    fig.savefig(fname, dpi=300, bbox_inches="tight")
    print(f"Saved -> {fname}")
if showPlot:
    plt.show()
plt.close(fig)

elif mode == "per_parameter":
    for param, p_label in zip(parameters, param_labels):
        fig, axes = plt.subplots(1, len(metrics),
                                figsize=(5 * len(metrics), 5))
        levels = param_levels[param]
        for m_idx, (metric, m_label) in enumerate(
            zip(metrics, metric_labels)):
            ax = axes[m_idx]
            means = [df.groupby(param)[metric].mean().get(lv, np.nan)
                      for lv in levels]

```

```

# bars show group mean only -> no error bars
ax.bar(range(len(levels)), means,
        color=bar_colors[:len(levels)], alpha=0.6, zorder=1)

# individual points overlaid with jitter
for lv_idx, lv in enumerate(levels):
    vals = df[df[param] == lv][metric].dropna().values
    jitter = np.random.uniform(-0.15, 0.15, size=len(vals))
    ax.scatter(np.full(len(vals), lv_idx) + jitter, vals,
               color="black", s=18, alpha=0.6, zorder=3)

ax.set_xticks(range(len(levels)))
ax.set_xticklabels(levels, rotation=15, ha="right")
ax.set_xlabel(p_label)
ax.set_ylabel(m_label)
ax.set_title(m_label)
ax.grid(axis="y", linestyle="--", alpha=0.5)

fig.suptitle(f"Main effects | {p_label}", fontsize=14)
plt.tight_layout()
fname = save_dir / f"main_effects_{param}.png"
if savePlot:
    fig.savefig(fname, dpi=300, bbox_inches="tight")
    print(f"Saved -> {fname}")
if showPlot:
    plt.show()
plt.close(fig)
else:
    print(f"WARNING: unknown mode '{mode}' - "
          f"use 'per_metric' or 'per_parameter'")

# =====
# HIERARCHICAL X-AXIS (shared by all summary bar charts)
# =====

def _draw_hierarchical_xaxis(ax, groups, group_size, level1_key,
                             level2_key, level3_key,
                             top_level_list, top_level_bg_colors):
    """
    Draw a 3-level hierarchical x-axis annotation below the plot.

    Level 1 (closest to bars): innermost category (e.g. leak direction)

```

```

Level 2 (middle):          mid-level category (e.g. hole size or
    ↪ pressure)
Level 3 (furthest):       outermost category + background shading

groups                    : list of (cat1, cat2, cat3) tuples - one per bar
    ↪ group,
                        ordered outermost ? innermost
group_size                : total number of bars per group (including gaps)
level1/2/3_key            : string labels for the three annotation levels
top_level_list            : ordered list of outermost category values, used for
                        background shading colour lookup
top_level_bg_colors      : list of background colours, one per top-level
    ↪ value
"""

# --- font sizes for each label level -> adjust these ---
FONTSIZE_L1 = 12          # innermost: direction labels
FONTSIZE_L2 = 12          # middle: hole size or pressure brackets
FONTSIZE_L3 = 9           # outermost: pressure or hole size brackets

ax.set_xticks([])         # suppress default ticks - all labels drawn
    ↪ manually
n_groups = len(groups)

# --- level 1: innermost labels (one per group) --- (e.g. direction)
for i, (cat1, cat2, cat3) in enumerate(groups):
    x_centre = i * group_size + (group_size - 1) / 2
    ax.annotate(cat3,
                xy=(x_centre, 0), xycoords=("data", "axes fraction"),
                xytext=(0, -18), textcoords="offset points",
                ha="center", va="top", fontsize=FONTSIZE_L1,
                annotation_clip=False)

def _span_label(i_start, i_end, label, y_offset, fontsize=FONTSIZE_L2):
    """
    Draw a labelled horizontal bracket spanning groups i_start to
    ↪ i_end.
    The bracket is a horizontal line with a centred rounded-box label.
    Vertical tick marks at each end anchor the bracket visually.
    """

    x0 = i_start * group_size - 0.4
    x1 = i_end * group_size + (group_size - 1) + 0.4
    xm = (x0 + x1) / 2

```

```

# horizontal spanning line
ax.annotate("",
            xy=(x1, 0), xycoords=("data", "axes fraction"),
            xytext=(x0, 0), textcoords=("data", "axes fraction"),
            annotation_clip=False,
            arrowprops=dict(arrowstyle="--", color="gray", lw=1.0,
                            shrinkA=0, shrinkB=0))

# small vertical ticks at each end to delimit the span
for x_tick in [x0, x1]:
    ax.annotate("",
                xy=(x_tick, 0), xycoords=("data", "axes fraction"),
                xytext=(0, -6), textcoords="offset points",
                annotation_clip=False,
                arrowprops=dict(arrowstyle="--", color="gray",
                                ↪ lw=1.0,
                                shrinkA=0, shrinkB=0))

# centred label with rounded white box
ax.annotate(label,
            xy=(xm, 0), xycoords=("data", "axes fraction"),
            xytext=(0, y_offset), textcoords="offset points",
            ha="center", va="top", fontsize=fontsize,
            annotation_clip=False,
            bbox=dict(boxstyle="round,pad=0.25", fc="white",
                    ec="gray", lw=0.8, alpha=0.95))

# --- level 2: mid-level spans ---
# find contiguous runs of the same (cat1, cat2) combination
i = 0
while i < n_groups:
    cat1, cat2, _ = groups[i]
    j = i
    while j < n_groups and groups[j][0] == cat1 and groups[j][1] ==
        ↪ cat2:
        j += 1
    _span_label(i, j - 1, _fmt_pressure(cat2), y_offset=-33)
    i = j

# --- level 3: outermost spans + background shading --- (same as
    ↪ subplot title)
i = 0
while i < n_groups:

```

```

    cat1, _, _ = groups[i]
    j = i
    while j < n_groups and groups[j][0] == cat1:
        j += 1
    #_span_label(i, j - 1, cat1, y_offset=-50, fontsize=FONTSIZE_L3)

    # shaded background for this top-level zone
    x0 = i * group_size - 0.5
    x1 = (j - 1) * group_size + (group_size - 1) + 0.5
    p_i = (top_level_list.index(cat1)
           if cat1 in top_level_list else 0)
    ax.axvspan(x0, x1,
               color=top_level_bg_colors[p_i %
                                         len(top_level_bg_colors)],
               alpha=0.3, zorder=0)

    i = j

# vertical separator lines between bar groups (at every group
↳ boundary)
for i in range(1, n_groups):
    x_sep = i * group_size - 0.5
    ax.axvline(x_sep, color="gray", linewidth=0.5,
               linestyle="--", alpha=0.5, zorder=1)

plt.subplots_adjust(bottom=0.22)

# =====
↳ =====
# HELPERS SHARED BY BAR CHART FUNCTIONS
# =====
↳ =====

def _build_groups_press_first(pressures, hole_cats, num_cases,
                             dir_labels_by_press):
    """
    Build ordered group list with hierarchy: pressure > hole > direction.
    Returns list of (pressure, hole, direction) and matching case_n list.
    """
    groups = []
    case_ns = []
    for p_idx, pres in enumerate(pressures):
        dirs = dir_labels_by_press[p_idx]
        for hole in hole_cats:

```



```

        for case_n in range(1, num_cases + 1):
            direc = dirs[case_n - 1] if case_n <= len(dirs) else
                ↳ str(case_n)
            groups.append((pres, hole, direc))
            case_ns.append(case_n)
    return groups, case_ns

def _build_groups_hole_first(pressures, hole_cats, num_cases,
                             dir_labels_by_press):
    """
    Build ordered group list with hierarchy: hole > pressure > direction.
    Returns list of (hole, pressure, direction) and matching case_n list.
    """
    groups = []
    case_ns = []
    for hole in hole_cats:
        for p_idx, pres in enumerate(pressures):
            dirs = dir_labels_by_press[p_idx]
            for case_n in range(1, num_cases + 1):
                direc = dirs[case_n - 1] if case_n <= len(dirs) else
                    ↳ str(case_n)
                groups.append((hole, pres, direc))
                case_ns.append(case_n)
    return groups, case_ns

def _get_row(df, pres, hole, case_n, vent):
    """Fetch one row from df matching pressure / hole / case_n /
    ↳ vent_type."""
    mask = ((df["pressure"] == pres) & (df["hole_size"] == hole) &
            (df["case_n"] == case_n) & (df["vent_type"] == vent))
    return df[mask]

def _exceedance_cols(df):
    """Return exceed_* column names in the order they appear in df."""
    return [c for c in df.columns if c.startswith("exceed_")]

# =====
# ↳ =====
# FLAMMABLE AREA + MAX DISTANCE BAR CHART

```

```

# =====
↳ =====

def _plot_area_bars_on_ax(ax, ax2, df, groups, case_ns, group_size, bar_w,
                        color_nat, color_forced, hatch_forced,
                        pressures, hole_cats, hierarchy):
    """
    Core bar-drawing logic shared by pressure-first and hole-first area
    ↳ charts.
    groups      : list of (cat1, cat2, direction) - order depends on
    ↳ hierarchy
    hierarchy    : "press_first" or "hole_first"
    ax2          : twin y-axis for max distance (may be None)
    """
    total_w = len(groups) * group_size

    for g_idx, (group_tuple, case_n) in enumerate(zip(groups, case_ns)):
        if hierarchy == "press_first":
            pres, hole, _ = group_tuple
        else:
            hole, pres, _ = group_tuple

        # 4 bars per group: nat_area, nat_dist, forced_area, forced_dist
        bar_specs = [
            ("naturalVent", color_nat, "", 0, "area"),
            ("naturalVent", color_nat, hatch_forced, 1, "dist"),
            ("forcedVent", color_forced, "", 2, "area"),
            ("forcedVent", color_forced, hatch_forced, 3, "dist"),
        ]

        for vent, color, hatch, offset, quantity in bar_specs:
            x_pos = g_idx * group_size + offset
            row = _get_row(df, pres, hole, case_n, vent)
            if row.empty:
                continue

            if quantity == "area":
                val = float(row["flammable_area"].iloc[0])
                ax.bar(x_pos, val, width=bar_w, color=color, hatch=hatch,
                      edgecolor="black", linewidth=0.6, alpha=0.85,
                      ↳ zorder=2)
            elif quantity == "dist" and ax2 is not None:
                if "max_distance" not in row.columns:
                    continue

```

```

        val = float(row["max_distance"].iloc[0])
        ax2.bar(x_pos, val, width=bar_w, color=color, hatch=hatch,
                edgecolor="black", linewidth=0.6, alpha=0.85,
                ↪ zorder=2)

    ax.set_xlim(-0.5, total_w - 0.5)

def plot_area_bars(df, pressures, hole_cats, num_cases, dir_labels_by_press,
                  color_nat, color_forced, hatch_forced,
                  ↪ pressure_bg_colors,
                  save_path, savePlot, showPlot,
                  hierarchy="press_first"):
    """
    Grouped bar chart of flammable area and max distance.
    One subplot per top-level value, stacked vertically, independent
    ↪ y-axes.
    4 bars per group: nat area / nat dist / forced area / forced dist.
    Legend placed below all subplots.
    """
    has_dist = "max_distance" in df.columns

    if hierarchy == "press_first":
        groups, case_ns = _build_groups_press_first(
            pressures, hole_cats, num_cases, dir_labels_by_press)
        top_levels = pressures
        top_level_colors = pressure_bg_colors
        subplot_title_prefix = "Pressure:"
    else:
        groups, case_ns = _build_groups_hole_first(
            pressures, hole_cats, num_cases, dir_labels_by_press)
        top_levels = hole_cats
        top_level_colors = ["#eff3ff", "#bdd7e7", "#6baed6"]
        subplot_title_prefix = "Hole size:"

    group_size = 5 # nat_area + nat_dist + forced_area + forced_dist +
    ↪ gap
    bar_w = 0.7
    n_top = len(top_levels)

    n_per_top = (len(hole_cats) * num_cases if hierarchy == "press_first"
                 else len(pressures) * num_cases)
    subplot_w = max(6, n_per_top * group_size * 0.45)

```

```

fig, axes = plt.subplots(n_top, 1,
                          figsize=(subplot_w, 5 * n_top + 1),
                          sharey=False)

if n_top == 1:
    axes = [axes]

ax2_list = []    # twin axes for distance

for ax_idx, (ax, top_val) in enumerate(zip(axes, top_levels)):
    filtered = [(g, cn) for g, cn in zip(groups, case_ns)
                 if g[0] == top_val]
    total_w = len(filtered) * group_size
    ax.set_xlim(-0.5, total_w - 0.5)

    ax2 = ax.twinx() if has_dist else None
    ax2_list.append(ax2)

    for g_idx, (group_tuple, case_n) in enumerate(filtered):
        if hierarchy == "press_first":
            pres, hole, _ = group_tuple
        else:
            hole, pres, _ = group_tuple

        bar_specs = [
            ("naturalVent", color_nat, "", 0, "area"),
            ("naturalVent", color_nat, hatch_forced, 1, "dist"),
            ("forcedVent", color_forced, "", 2, "area"),
            ("forcedVent", color_forced, hatch_forced, 3, "dist"),
        ]

        for vent, color, hatch, offset, quantity in bar_specs:
            x_pos = g_idx * group_size + offset
            row = _get_row(df, pres, hole, case_n, vent)
            if row.empty:
                continue
            if quantity == "area":
                val = float(row["flammable_area"].iloc[0])
                if not np.isnan(val):
                    ax.bar(x_pos, val, width=bar_w, color=color,
                           hatch=hatch, edgecolor="black",
                           linewidth=0.6, alpha=0.85, zorder=2)
            elif quantity == "dist" and ax2 is not None:
                val = float(row["max_distance"].iloc[0])
                if not np.isnan(val):
                    ax2.bar(x_pos, val, width=bar_w, color=color,

```

```

        hatch=hatch, edgecolor="black",
        linewidth=0.6, alpha=0.85, zorder=2)

ax.set_ylabel(r"Flammable area (m$^2$)")
if ax2 is not None:
    ax2.set_ylabel("Max distance (m)", color="gray")
    ax2.tick_params(axis="y", labelcolor="gray")
ax.set_title(f"{subplot_title_prefix} {_fmt_pressure(top_val)}",
    ↪ fontsize=12)
ax.grid(axis="y", linestyle="--", alpha=0.4)

sub_groups = [g for g, _ in filtered]
_draw_hierarchical_xaxis(ax, sub_groups, group_size,
    "dir", "mid", "top",
    top_levels, top_level_colors)

legend_handles = [
    Patch(facecolor=color_nat, edgecolor="black",
        label="Nat. vent - area"),
    Patch(facecolor=color_nat, edgecolor="black", hatch=hatch_forced,
        label="Nat. vent - distance"),
    Patch(facecolor=color_forced, edgecolor="black",
        label="Forced vent - area"),
    Patch(facecolor=color_forced, edgecolor="black", hatch=hatch_forced,
        label="Forced vent - distance"),
]
fig.legend(handles=legend_handles,
    loc="lower center",
    bbox_to_anchor=(0.5, 0),
    ncol=4, framealpha=0.9)

hier_label = ("pressure > hole > direction"
    if hierarchy == "press_first"
    else "hole > pressure > direction")
fig.suptitle(
    f"Flammable area and max extent | {hier_label}",
    fontsize=13)
fig.subplots_adjust(top=0.94, bottom=0.08, hspace=0.3)

if savePlot:
    save_path.parent.mkdir(parents=True, exist_ok=True)
    fig.savefig(save_path, dpi=300, bbox_inches="tight")
    print(f"Saved -> {save_path}")
if showPlot:

```

```

        plt.show()
    plt.close(fig)

# =====
# FLAMMABLE VOLUME BAR CHARTS (grouped and stacked, with subplots)
# =====

def plot_flamvol_bars(df, pressures, hole_cats, num_cases,
    dir_labels_by_press,
        color_nat, color_forced, hatch_forced,
        pressure_bg_colors,
        save_path, savePlot, showPlot,
        hierarchy="press_first"):
    """
    Grouped bar chart of flammable volume at steady state.
    One subplot per top-level value stacked vertically.
    Independent y-axes so hole-size subplots are readable.
    Legend placed below all subplots.
    """
    if hierarchy == "press_first":
        groups, case_ns = _build_groups_press_first(
            pressures, hole_cats, num_cases, dir_labels_by_press)
        top_levels = pressures
        top_level_colors = pressure_bg_colors
        subplot_title_prefix = "Pressure:"
    else:
        groups, case_ns = _build_groups_hole_first(
            pressures, hole_cats, num_cases, dir_labels_by_press)
        top_levels = hole_cats
        top_level_colors = ["#eff3ff", "#bdd7e7", "#6baed6"]
        subplot_title_prefix = "Hole size:"

    group_size = 3
    bar_w = 0.8
    n_top = len(top_levels)

    n_per_top = (len(hole_cats) * num_cases if hierarchy == "press_first"
        else len(pressures) * num_cases)
    subplot_w = max(6, n_per_top * group_size * 0.45)

    # sharey=False so each subplot scales independently
    fig, axes = plt.subplots(n_top, 1,

```

```

figsize=(subplot_w, 5 * n_top + 1),
sharey=False)

if n_top == 1:
    axes = [axes]

legend_handles = [
    Patch(facecolor=color_nat, edgecolor="black",
          label="Natural ventilation"),
    Patch(facecolor=color_forced, edgecolor="black",
          hatch=hatch_forced, label="Forced ventilation"),
]

for ax_idx, (ax, top_val) in enumerate(zip(axes, top_levels)):
    filtered = [(g, cn) for g, cn in zip(groups, case_ns)
                if g[0] == top_val]
    total_w = len(filtered) * group_size
    ax.set_xlim(-0.5, total_w - 0.5)

    for g_idx, (group_tuple, case_n) in enumerate(filtered):
        if hierarchy == "press_first":
            pres, hole, _ = group_tuple
        else:
            hole, pres, _ = group_tuple

        base_x = g_idx * group_size
        for v_idx, (vent, color, hatch) in enumerate([
            ("naturalVent", color_nat, ""),
            ("forcedVent", color_forced, hatch_forced)]):
            row = _get_row(df, pres, hole, case_n, vent)
            if row.empty:
                continue
            val = float(row["flammable_volume"].iloc[0])
            if np.isnan(val):
                continue
            ax.bar(base_x + v_idx, val, width=bar_w,
                  color=color, hatch=hatch,
                  edgecolor="black", linewidth=0.6,
                  alpha=0.85, zorder=2)

    ax.set_ylabel(r"Flammable volume (m$^3$)")
    ax.set_title(f"{subplot_title_prefix} {_fmt_pressure(top_val)}",
                 ↪ fontsize=12)
    ax.grid(axis="y", linestyle="--", alpha=0.4)

```

```

sub_groups = [g for g, _ in filtered]
_draw_hierarchical_xaxis(ax, sub_groups, group_size,
                        "dir", "mid", "top",
                        top_levels, top_level_colors)

hier_label = ("pressure > hole > direction"
              if hierarchy == "press_first"
              else "hole > pressure > direction")

# legend below all subplots
fig.legend(handles=legend_handles,
          loc="lower center",
          bbox_to_anchor=(0.5, 0),
          ncol=2, framealpha=0.9,
          title="Ventilation type")

fig.suptitle(
    f"Flammable cloud volume (steady state) | {hier_label}",
    fontsize=13)

# leave room at bottom for legend and at top for suptitle
fig.subplots_adjust(top=0.93, bottom=0.09, hspace=0.3)

if savePlot:
    save_path.parent.mkdir(parents=True, exist_ok=True)
    fig.savefig(save_path, dpi=300, bbox_inches="tight")
    print(f"Saved -> {save_path}")
if showPlot:
    plt.show()
plt.close(fig)

# =====
#  ↳ =====
# RADIATION EXCEEDANCE BAR CHARTS (grouped and stacked, with subplots)
# =====
#  ↳ =====

def _draw_rad_grouped_on_ax(ax, df, groups, case_ns, group_size, bar_w,
                          exceed_cols, rad_colors, hatch_forced,
                          pressures, hole_cats, hierarchy,
                          top_level_filter, top_level_list,
                          top_level_bg_colors):
    """
    Draw grouped radiation bars on a single axes, optionally filtered to

```



```

one top-level value (e.g. one pressure or one hole size).
"""
n_thresh = len(exceed_cols)

# filter groups to the requested top-level slice
filtered = [(g, cn) for g, cn in zip(groups, case_ns)
            if g[0] == top_level_filter]

total_w = len(filtered) * group_size
ax.set_xlim(-0.5, total_w - 0.5)

for g_idx, (group_tuple, case_n) in enumerate(filtered):
    if hierarchy == "press_first":
        pres, hole, _ = group_tuple
    else:
        hole, pres, _ = group_tuple

    base_x = g_idx * group_size
    for v_idx, (vent, hatch) in enumerate([
        ("naturalVent", ""),
        ("forcedVent", hatch_forced)]):
        row = _get_row(df, pres, hole, case_n, vent)
        if row.empty:
            continue
        for t_idx, (col, color) in enumerate(
            zip(exceed_cols, rad_colors[1:])):
            x = base_x + v_idx * n_thresh + t_idx
            val = (float(row[col].iloc[0])
                   if not row[col].isna().all() else 0.0)
            ax.bar(x, val, width=bar_w, color=color, hatch=hatch,
                   edgecolor="black", linewidth=0.5, alpha=0.85,
                   zorder=2)

# draw hierarchical axis for the filtered subset
sub_groups = [g for g, _ in filtered]
_draw_hierarchical_xaxis(ax, sub_groups, group_size,
                        "dir", "mid", "top",
                        top_level_list, top_level_bg_colors)

def _draw_rad_stacked_on_ax(ax, df, groups, case_ns, group_size, bar_w,
                           exceed_cols, rad_colors, hatch_forced,
                           pressures, hole_cats, hierarchy,
                           top_level_filter, top_level_list,

```

```

        top_level_bg_colors):
    """
    Draw stacked radiation bars on a single axes, filtered to one
    ↪ top-level
    value. Each bar is stacked by exceedance threshold.
    """
    filtered = [(g, cn) for g, cn in zip(groups, case_ns)
                if g[0] == top_level_filter]

    total_w = len(filtered) * group_size
    ax.set_xlim(-0.5, total_w - 0.5)

    for g_idx, (group_tuple, case_n) in enumerate(filtered):
        if hierarchy == "press_first":
            pres, hole, _ = group_tuple
        else:
            hole, pres, _ = group_tuple

        base_x = g_idx * group_size
        for v_idx, (vent, hatch) in enumerate([
            ("naturalVent", ""),
            ("forcedVent", hatch_forced)]):
            row = _get_row(df, pres, hole, case_n, vent)
            if row.empty:
                continue
            x = base_x + v_idx
            bottom = 0.0
            for col, color in zip(exceed_cols, rad_colors[1:]):
                val = (float(row[col].iloc[0])
                      if not row[col].isna().all() else 0.0)
                ax.bar(x, val, width=bar_w, bottom=bottom,
                      color=color, hatch=hatch,
                      edgecolor="black", linewidth=0.5, alpha=0.85,
                      ↪ zorder=2)
                bottom += val

    sub_groups = [g for g, _ in filtered]
    _draw_hierarchical_xaxis(ax, sub_groups, group_size,
                            "dir", "mid", "top",
                            top_level_list, top_level_bg_colors)

def _rad_legends(ax, exceed_cols, rad_colors, thresholds, hatch_forced):
    """Add threshold colour legend and ventilation hatch legend to ax."""

```

```

thresh_handles = [
    Patch(facecolor=rad_colors[i + 1], edgecolor="black",
          label=f">{thresholds[i][0]:,} W/m2"),
    for i in range(len(exceed_cols))
]
vent_handles = [
    Patch(facecolor="gray", edgecolor="black", hatch="",
          label="Natural ventilation"),
    Patch(facecolor="gray", edgecolor="black", hatch=hatch_forced,
          label="Forced ventilation"),
]
leg1 = ax.legend(handles=thresh_handles, loc="lower center",
    ↪   bbox_to_anchor=(0.5, 0),
          title="Threshold", framealpha=0.9)
ax.add_artist(leg1)
ax.legend(handles=vent_handles, loc="upper right", framealpha=0.9)

def _make_rad_figure(df, pressures, hole_cats, num_cases,
    ↪   dir_labels_by_press,
          rad_colors, thresholds, hatch_forced,
          pressure_bg_colors, save_path, savePlot, showPlot,
          chart_type, hierarchy):

    exceed_cols = _exceedance_cols(df)
    if not exceed_cols:
        print(" No exceedance columns in df - skipping radiation chart.")
        return

    if hierarchy == "press_first":
        groups, case_ns = _build_groups_press_first(
            pressures, hole_cats, num_cases, dir_labels_by_press)
        top_levels = pressures
        top_level_colors = pressure_bg_colors
        subplot_title_prefix = "Pressure:"
    else:
        groups, case_ns = _build_groups_hole_first(
            pressures, hole_cats, num_cases, dir_labels_by_press)
        top_levels = hole_cats
        top_level_colors = ["#eff3ff", "#bdd7e7", "#6baed6"]
        subplot_title_prefix = "Hole size:"

    n_thresh = len(exceed_cols)
    n_top = len(top_levels)

```

```

if chart_type == "grouped":
    group_size = 2 * n_thresh + 1
    bar_w      = 0.7
    draw_fn    = _draw_rad_grouped_on_ax
    y_label    = r"Exceedance area (m$^2$)"
    type_label = "grouped by threshold"
else:
    group_size = 3
    bar_w      = 0.8
    draw_fn    = _draw_rad_stacked_on_ax
    y_label    = r"Exceedance area (m$^2$)"
    type_label = "stacked"

n_cases_per_top = (len(hole_cats) * num_cases
                    if hierarchy == "press_first"
                    else len(pressures) * num_cases)
subplot_w = max(6, n_cases_per_top * group_size * 0.45)

fig, axes = plt.subplots(n_top, 1,
                        figsize=(subplot_w, 7 * n_top),
                        sharey=False)

if n_top == 1:
    axes = [axes]

# --- draw each subplot ---
for ax_idx, (ax, top_val) in enumerate(zip(axes, top_levels)):
    draw_fn(ax, df, groups, case_ns, group_size, bar_w,
            exceed_cols, rad_colors, hatch_forced,
            pressures, hole_cats, hierarchy,
            top_val, top_levels, top_level_colors)

    ax.set_ylabel(y_label)
    ax.set_title(f"{subplot_title_prefix} {_fmt_pressure(top_val)}",
                ↪ fontsize=12)
    ax.grid(axis="y", linestyle="--", alpha=0.4)

# --- figure-level legends below all subplots ---
thresh_handles = [
    Patch(facecolor=rad_colors[i + 1], edgecolor="black",
          label=rf">{thresholds[i][0]:,} W/m$^2$")
    for i in range(n_thresh)
]
vent_handles = [

```

```

    Patch(facecolor="gray", edgecolor="black", hatch="",
          label="Natural ventilation"),
    Patch(facecolor="gray", edgecolor="black", hatch=hatch_forced,
          label="Forced ventilation"),
]

fig.legend(handles=thresh_handles,
           loc="lower center",
           bbox_to_anchor=(0.40, 0),
           ncol=2,
           framealpha=0.9,
           fontsize=11,
           title="Radiation threshold")

fig.legend(handles=vent_handles,
           loc="lower center",
           bbox_to_anchor=(0.63, 0),
           ncol=1,
           framealpha=0.9,
           fontsize=11,
           title="Ventilation type")

hier_label = ("pressure > hole > direction"
              if hierarchy == "press_first"
              else "hole > pressure > direction")

fig.suptitle(
    f"Radiation exceedance areas - {type_label} | {hier_label}",
    fontsize=14)
fig.subplots_adjust(top=0.94, bottom=0.09, hspace=0.23)

if savePlot:
    save_path.parent.mkdir(parents=True, exist_ok=True)
    fig.savefig(save_path, dpi=300, bbox_inches="tight")
    print(f"Saved -> {save_path}")
if showPlot:
    plt.show()
plt.close(fig)

# --- Public wrappers so consolidated_analysis.py calls stay simple
↪ -----

def plot_radBars_grouped(df, pressures, hole_cats, num_cases,
                        dir_labels_by_press, rad_colors, thresholds,
```

```

        hatch_forced, pressure_bg_colors,
        save_path, savePlot, showPlot,
        hierarchy="press_first"):
"""Radiation exceedance - grouped bars, one subplot per top-level
    ↪ value."""
_make_rad_figure(df, pressures, hole_cats, num_cases,
    ↪ dir_labels_by_press,
        rad_colors, thresholds, hatch_forced,
        ↪ pressure_bg_colors,
        save_path, savePlot, showPlot,
        chart_type="grouped", hierarchy=hierarchy)

def plot_radBars_stacked(df, pressures, hole_cats, num_cases,
    dir_labels_by_press, rad_colors, thresholds,
    hatch_forced, pressure_bg_colors,
    save_path, savePlot, showPlot,
    hierarchy="press_first"):
"""Radiation exceedance - stacked bars, one subplot per top-level
    ↪ value."""
_make_rad_figure(df, pressures, hole_cats, num_cases,
    ↪ dir_labels_by_press,
        rad_colors, thresholds, hatch_forced,
        ↪ pressure_bg_colors,
        save_path, savePlot, showPlot,
        chart_type="stacked", hierarchy=hierarchy)

```

B.5.3 Scores

```

"""
kfx_scores.py - hazard score computation functions
=====
All physics and scoring logic lives here.
Configuration (thresholds, zones, constants) is passed in as arguments
so this module has no global state - it works for any set of parameters.

Imported by consolidated_analysis.py.
"""

import numpy as np
from kfx_utils import load_r3d_file, get_field, nearest_z_index,
    ↪ midpoint_widths

# =====
# ↪ =====
# RADIATION
# =====
# ↪ =====

def compute_radiation_score_full(case_path, r3d_filename, fieldname,
                                slice_z, exclusion_zones, thresholds):
    """
    Load radiation r3d, extract a Z slice, compute weighted exceedance
    ↪ area
    outside exclusion zones for each threshold.

    Parameters
    -----
    case_path      : Path - folder containing the r3d file
    r3d_filename   : str - e.g. "case_bmg_exit.r3d"
    fieldname      : str - e.g. "Radiation(W/m2)"
    slice_z        : float - Z height to evaluate at (m)
    exclusion_zones : list of dicts with x0,y0,x1,y1 keys
    thresholds     : list of (threshold_W/m2, weight, description) tuples

    Returns
    -----
    score    : float - weighted exceedance area (higher = more severe)
    exceed   : dict - {threshold: area_m2} for each threshold
    """

```

```

reader = load_r3d_file(case_path / r3d_filename)

fieldnames = [f.strip() for f in reader.fieldnames]
idx         = fieldnames.index(fieldname)
field       = reader.fielddata[idx].astype(float)    # (nx, ny, nz)

xc = np.asarray(reader.xc)
yc = np.asarray(reader.yc)
zc = np.asarray(reader.zc)

iz         = nearest_z_index(zc, slice_z)
field_2d   = field[:, :, iz]    # horizontal slice at slice_z

# cell areas from midpoint rule
cell_area = np.outer(midpoint_widths(xc), midpoint_widths(yc))

# exclusion mask - True where cells should NOT be counted
nx, ny = len(xc), len(yc)
excl    = np.zeros((nx, ny), dtype=bool)
for zone in exclusion_zones:
    xi = (xc >= zone["x0"]) & (xc <= zone["x1"])
    yi = (yc >= zone["y0"]) & (yc <= zone["y1"])
    excl[np.ix_(xi, yi)] = True
valid = ~excl

score = 0.0
exceed = {}
for thresh, weight, *_ in thresholds:
    above = (field_2d > thresh) & valid
    area   = float(np.sum(cell_area[above]))
    score  += weight * area
    exceed[thresh] = area

return score, exceed

# =====
#  ↪ =====
# FLAMMABLE AREA AND VOLUME
# =====
#  ↪ =====

def compute_flammable_area_full(case_path, r3d_filename, lfl, ufl,
    ↪ leak_pos):

```



```

"""
Load gas concentration r3d, project flammable cells (LFL-UFL) onto XY
    ↪ plane,
and compute the max distance from the leak source to any flammable
    ↪ cell.

Parameters
-----
case_path      : Path - folder containing the r3d file
r3d_filename   : str - e.g. "case_std_exit.r3d"
lfl, ufl       : float - lower/upper flammability limits (vol %)
leak_pos       : (x, y) tuple - leak source position in metres

Returns
-----
area           : float - projected flammable area (m2)
max_dist       : float - max distance from leak to flammable cell (m)
proj_mask      : ndarray bool (nx, ny) - True where column is flammable
xc, yc         : ndarray - cell-centre coordinates
"""
reader = load_r3d_file(case_path / r3d_filename)

gas = get_field(reader, "Fuel gas (vol %)") # (nx, ny, nz)
xs = np.asarray(reader.xs)
ys = np.asarray(reader.ys)
xc = np.asarray(reader.xc)
yc = np.asarray(reader.yc)

nx, ny, _ = gas.shape

# cell areas from staggered (face) coordinates
dx = np.zeros(nx); dx[1:] = xs[1:] - xs[:-1]
dy = np.zeros(ny); dy[1:] = ys[1:] - ys[:-1]
cell_area = np.outer(dx, dy)

flam_mask = (gas >= lfl) & (gas <= ufl) # True where gas is
    ↪ flammable (3D)
proj_mask = np.any(flam_mask, axis=2) # collapse z: any layer
    ↪ flammable?
area = float(np.sum(cell_area[proj_mask]))

# max XY distance from leak to furthest flammable cell centre
ix_list, iy_list = np.where(proj_mask)
if len(ix_list) == 0:

```

```

        max_dist = 0.0
    else:
        fx      = xc[ix_list]
        fy      = yc[iy_list]
        dists    = np.sqrt((fx - leak_pos[0])**2 + (fy - leak_pos[1])**2)
        max_dist = float(np.max(dists))

    return area, max_dist


def compute_flammable_volume(case_path, r3d_filename, lfl, ufl):
    """
    Load the exit r3d file and compute the total volume of cells where
    H2 concentration is within the flammable range (LFL-UFL).
    Uses r3d cell volumes directly rather than projected XY area.

    Returns scalar volume in m3.
    """
    reader = load_r3d_file(case_path / r3d_filename)

    gas = get_field(reader, "Fuel gas (vol %)")    # (nx, ny, nz)

    # cell volumes from the r3d reader
    cell_vol = reader.compute_cell_volume()        # (nx, ny, nz)

    flam_mask = (gas >= lfl) & (gas <= ufl)        # flammable cells
    return float(np.sum(cell_vol[flam_mask]))


# =====
# ↪ =====
# DYNAMIC PRESSURE
# =====
# ↪ =====


def assign_dp_zone_weights(xc, yc, dp_zones):
    """
    Build a 2D weight array (nx, ny).
    Zones are applied in reverse order so specific (smaller) zones
    overwrite the general catch-all outside zone.
    """
    weights = np.ones((len(xc), len(yc)))
    for zone in reversed(dp_zones):
        xi = (xc >= zone["x0"]) & (xc <= zone["x1"])

```

```

        yi = (yc >= zone["y0"]) & (yc <= zone["y1"])
        weights[np.ix_(xi, yi)] = zone["weight"]
    return weights

def compute_dynpres_score(case_path, r3d_filename, slice_z,
                          score_percentile, dp_zones,
                          p_atm, r_air, r_h2):
    """
    Load velocity and temperature fields, compute dynamic pressure  $q = \frac{1}{2} \rho U^2$ 
    with mixture density derived from the H2/air ideal gas law, apply
    spatial zone weights, and return a percentile-based danger score.

    Using a high percentile (e.g. 99.9th) instead of the max naturally
    ignores confined spikes from the jet impingement inside the container.

    Parameters
    -----
    case_path      : Path - folder containing the r3d file
    r3d_filename   : str - e.g. "case_std_exit.r3d"
    slice_z        : float - Z height to evaluate at (m), ideally
        ↪ ground level
    score_percentile : float - e.g. 99.9
    dp_zones       : list of dicts with x0,y0,x1,y1,weight keys
    p_atm          : float - ambient pressure (Pa)
    r_air, r_h2    : float - specific gas constants (J/kg/K)

    Returns
    -----
    score : float - weighted percentile dynamic pressure (Pa)
    """
    reader = load_r3d_file(case_path / r3d_filename)

    U = get_field(reader, "U(m/s)")
    V = get_field(reader, "V(m/s)")
    W = get_field(reader, "W(m/s)")
    T = get_field(reader, "Temp(K)")
    xH2 = get_field(reader, "Fuel gas (vol %)") / 100.0 # volume fraction
        ↪ [0-1]

    T = np.clip(T, 1.0, None) # guard against unphysical zero/negative
        ↪ temperatures

```

```

# mixture specific gas constant weighted by H2 mole fraction
R_mix = xH2 * r_h2 + (1.0 - xH2) * r_air

rho = p_atm / (R_mix * T)          # ideal gas:  $\rho = P/(R \cdot T)$ 
    ↪ [kg/m³]
q    = 0.5 * rho * (U**2 + V**2 + W**2) # dynamic pressure [Pa]

xc = np.asarray(reader.xc)
yc = np.asarray(reader.yc)
zc = np.asarray(reader.zc)

iz    = nearest_z_index(zc, slice_z)
q_2d = q[:, :, iz]    # ground-level 2D slice

weights    = assign_dp_zone_weights(xc, yc, dp_zones)
q_weighted = q_2d * weights    # zero out container, downweight
    ↪ firewall zone

# percentile over cells that actually contribute to the score (weight
    ↪ > 0)
active = q_weighted[weights > 0]
if active.size == 0:
    return 0.0

return float(np.nanpercentile(active, score_percentile))

```

B.5.4 Utilities

```

"""
kfx_utils.py - shared low-level helpers for KFX r3d file handling
=====
Imported by kfx_scores.py, kfx_plots.py, and consolidated_analysis.py.
Do not put configuration or physics constants here - only generic tools.
"""

import json
from pathlib import Path

import numpy as np
import matplotlib.patches as mpatches

from kfxutils import r3d

def load_r3d_file(filepath):
    """Load a KFX r3d file, converting type-4 to type-2 if needed."""
    reader = r3d.read(str(filepath))
    if reader.r3d_type == 4:
        reader.to_type2()
    return reader

def get_field(reader, fieldname):
    """
    Extract a named 3-D field (nx, ny, nz) from an r3d reader as float64.
    Raises a clear ValueError listing available fields if not found.
    """
    fieldnames = [f.strip() for f in reader.fieldnames]
    try:
        idx = fieldnames.index(fieldname)
    except ValueError:
        raise ValueError(f"Field '{fieldname}' not found. Available:
        ↪ {fieldnames}")
    return reader.fielddata[idx].astype(float)

def nearest_z_index(zc, z_target):
    """Return the index of the z cell-centre closest to z_target."""
    return int(np.argmin(np.abs(np.asarray(zc) - z_target)))

```

```

def midpoint_widths(c):
    """
    Compute cell widths from centre coordinates using the midpoint rule.
    Used to approximate cell sizes when only cell centres are available.
    """
    c = np.asarray(c)
    w = np.zeros(len(c))
    w[0] = c[1] - c[0]
    w[-1] = c[-1] - c[-2]
    w[1:-1] = (c[2:] - c[:-2]) / 2.0
    return w

def load_floorplan(floorplan_json):
    """
    Load floorplan rectangles from a JSON file.
    Returns an empty list if the file is None or not found.
    """
    if floorplan_json is None:
        return []
    p = Path(floorplan_json)
    if not p.exists():
        print(f"WARNING: {floorplan_json} not found - floorplan will not be
              ↪ drawn.")
        return []
    with open(p) as f:
        return json.load(f)

def draw_floorplan(ax, rects, edgecolor="steelblue"):
    """Draw floorplan geometry as unfilled rectangles on ax."""
    for r in rects:
        patch = mpatches.Rectangle(
            (r["x0"], r["y0"]), r["x1"] - r["x0"], r["y1"] - r["y0"],
            linewidth=1.2, edgecolor=edgecolor, facecolor="none", zorder=2,
        )
        ax.add_patch(patch)

```

Appendix C

Geometry files

C.1 Forced ventilation

```
p DOOZER
c 0.7 0.7 0.7
p compressor
c 0 0.4 1
p rbg 0 0.4 1
mxyz -3.6000 1.1000 0.0000
box 2.2000 -0.1 1.6
box 0.1 -5.2 2.1
mxyz -3.6 1.1 1.5
box 1.6 -0.1 0.6
mxyz -3.6 1.1 2.1
box 2.2 -5.2 0.1
mxyz -1.4 -4.1 0
box -2.2 0.1 1.6
box -0.1 5.2 2.1
mxyz -1.4 -4.1 1.5
box -1.6 0.1 0.6
cp 0.7 0.7 0.7 compressor_inside
mxyz -2.5 -3.7 0.5
d 1
y 4
c 0.3 0.3 0.3
p rbg 0.3 0.3 0.3
p compressor_enclosure
mxyz -4.0000 4.0000 0.0000
box 15.0000 0.1000 4.0000
box 0.1000 -13.0000 4.0000
gx 14.9000
box 0.1000 -4.5000 4.0000
```

```

poro 0.0000 0.5000 0.5000 0.0000
gy -4.5000
box 0.1000 -2.0000 4.0000
poro 0.0000 0.0000 0.0000 0.0000
gy -2.0000
box -10.0000 0.1000 4.0000
gx -10.0000
box 0.1000 -3.5000 4.0000
gy -3.5000
box -4.9000 0.1000 4.0000
c 0.7 0.7 0.7
p rgb 0.7 0.7 0.7
p storage_and_firewalls
mxyz -0.3000 -0.5000 0.0000
box 0.1000 3.0000 3.0000
c 0.8 0.8 0.85
p rgb 0.8 0.8 0.85
mxyz -0.2000 0.0000 0.4500
d 0.5
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
gxyz -7.0000 0.7000 -2.1000
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
gxyz -7.0000 0.7000 -2.1000
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
gxyz -7.0000 0.7000 -2.1000
x 7.0000

```



```

gxyz -7.0000 0.0000 0.7000
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
c 0.7 0.7 0.7
p rgb 0.7 0.7 0.7
mxyz 6.8000 -0.5000 0.0000
box 0.1000 3.0000 3.0000
c 0.7 0.7 0.7
p rgb 0.7 0.7 0.7
p chiller
mxyz 8.0000 0.5000 0.0500
box 1.5000 1.5000 1.8000
c 0 0.4 1
p rgb 0 0.4 1
mxyz 8.7500 1.2500 1.8500
d 1.2
z 0.1000
c 0 0.8 0
p rgb 0 0.8 0
p dispenser_canopy_and_supports
mxyz 4.0000 -14.0000 4.5000
box 12.0000 7.0000 0.5000
c 0.7 0.7 0.7
p rgb 0.7 0.7 0.7
mxyz 6.0000 -11.5000 0.0000
d 0.3
z 4.5000
gxyz 0.0000 2.0000 -4.5000
z 4.5000
gxyz 8.0000 0.0000 -4.5000
z 4.5000
gxyz 0.0000 -2.0000 -4.5000
z 4.5000
p dispenser
c 0 0.4 1
p rgb 0 0.4 1
mxyz 10.0000 -11.7500 0.0000
d 0.15
z 1.2000
gxyz 3.5000 0.0000 -1.2000
z 1.2000

```

```
gxyz -6.2500 0.7500 -1.2000
box 1.5000 0.7500 3.0000
gx 3.7500
box 1.5000 0.7500 3.0000
gxyz -2.7500 -0.7500 0.0000
z 1.2000
gxyz -1.7500 0.0000 -1.2000
z 1.2000
gxyz 5.1500 0.0000 -1.2000
z 1.2000
p tube_trailer
c 0.95 0.95 0.95
p rgb 0.95 0.95 0.95
mxyz -13.0000 -8.0000 0.7500
box 2.3000 12.0000 0.3500
c 0.1 0.1 0.1
p rgb 0.1 0.1 0.1
mxyz -13.0000 -7.6000 0.3750
d 0.74
x 0.4000
gxyz -0.4000 0.9000 0.0000
x 0.4000
gxyz 1.5000 -0.9000 0.0000
x 0.4000
gxyz -0.4000 0.9000 0.0000
x 0.4000
gxyz -2.3000 4.4000 0.0000
x 0.4000
gxyz -0.4000 0.9000 0.0000
x 0.4000
gxyz 1.5000 -0.9000 0.0000
x 0.4000
gxyz -0.4000 0.9000 0.0000
x 0.4000
gxyz -2.3000 3.7500 0.0000
x 0.4000
gx 1.5000
x 0.4000
c 0.95 0.95 0.95
p rgb 0.95 0.95 0.95
mxyz -10.7000 0.8500 1.0950
box -2.3000 3.1500 1.0000
gz 1.0000
box -2.3000 2.3000 0.5000
```

```

gz 0.5000
box -2.3000 1.0000 0.5000
c 0.8 0.8 0.85
p rgb 0.8 0.8 0.85
mxyz -12.5500 -7.9500 1.4450
d 0.7
y 7.0000
gxyz 0.0000 -7.0000 0.7000
y 7.0000
gxyz 0.0000 -7.0000 0.7000
y 7.0000
gxyz 0.7000 -7.0000 -1.4000
y 7.0000
gxyz 0.0000 -7.0000 0.7000
y 7.0000
gxyz 0.0000 -7.0000 0.7000
y 7.0000
gxyz 0.7000 -7.0000 -1.4000
y 7.0000
gxyz 0.0000 -7.0000 0.7000
y 7.0000
gxyz 0.0000 -7.0000 0.7000
y 7.0000
c 0.3 0.3 0.3
p rgb 0.3 0.3 0.3
p tube_trailer_enclosure
mxyz -4.0000 4.0000 0.0000
box -3.0000 0.1000 4.0000
poro 0.0000 0.5000 0.5000 0.0000
gx -3.0000
box -2.0000 0.1000 4.0000
poro 0.0000 0.0000 0.0000 0.0000
gx -2.0000
box 0.1000 -13.0000 4.0000
gxyz 5.0000 -13.0000 0.0000
box -5.0000 0.1000 4.0000
p gas_bottle_rack_3
c 0.7 0.7 0.7
p rgb 0.7 0.7 0.7
mxyz -6.1000 3.3000 0.0000
box 2.0000 0.6500 2.0000
c 0.7 0.7 0.7
p rgb 0.7 0.7 0.7
p car

```

```
c 0.1 0.3 0.7
p rbg 0.1 0.3 0.7
mxyz 8.8000 -13.5000 0.7000
box 3.0000 -1.8000 1.2000
box -1.2000 -1.8000 0.5000
c 0.05 0.05 0.05
p rbg 0.05 0.05 0.05
mxyz 8.5000 -13.8000 0.3500
y 0.3000
gx 2.5000
y -0.3000
gy -1.5000
y 0.3000
gx -2.5000
y -0.3000
c 0.85 0.85 0.9
p rbg 0.85 0.85 0.9
p low_storage
mxyz -5.5000 -7.0000 1.2000
d 1.5
y 8.0000
gx -1.6000
y -8.0000
gxyz -0.7500 1.0000 -0.7500
box 1.5000 0.4000 -0.4500
gy 5.6000
box 1.5000 0.4000 -0.4500
gx 1.6000
box 1.5000 0.4000 -0.4500
gy -5.6000
box 1.5000 0.4000 -0.4500
gxyz -1.7000 7.0000 -0.4500
box 3.5000 0.1000 2.1500
c 0.4 0.4 0.4
p rbg 0.4 0.4 0.4
mxyz -15.0000 -16.0000 0.0000
box 32.0000 23.0000 -0.1000
```

C.2 Natural ventilation

```

p DOOZER
cp 0 0.4 1 comp_shed
mxyz -3.6000 1.1000 0.0000
box 2.2000 -0.1 2.0000
box 0.1 -5.2 2.
mxyz -1.4 -4.1 0
box -2.2 0.1 2.
box -0.1 5.2 2.
mxyz -2.5 -3.7 0.5
cp 1 0.4 0 comp_roof
mxyz -3.7 -1.5 2.4
wh 5.4 0.1
xz 1.2 -0.3
gx -0.02
xz 1.2 0.3
cp 0.7 0.7 0.7 compressor_inside
mxyz -2.5 -3.7 0.5
d 1
y 4
c 0.3 0.3 0.3
p rbg 0.3 0.3 0.3
p compressor_enclosure
mxyz -4.0000 4.0000 0.0000
box 15.0000 0.1000 4.0000
box 0.1000 -13.0000 4.0000
gx 14.9000
box 0.1000 -4.5000 4.0000
poro 0.0000 0.5000 0.5000 0.0000
gy -4.5000
box 0.1000 -2.0000 4.0000
poro 0.0000 0.0000 0.0000 0.0000
gy -2.0000
box -10.0000 0.1000 4.0000
gx -10.0000
box 0.1000 -3.5000 4.0000
gy -3.5000
box -4.9000 0.1000 4.0000
c 0.7 0.7 0.7
p rbg 0.7 0.7 0.7
p storage_and_firewalls
mxyz -0.3000 -0.5000 0.0000

```

```

box 0.1000 3.0000 3.0000
c 0.8 0.8 0.85
p rgb 0.8 0.8 0.85
mxyz -0.2000 0.0000 0.4500
d 0.5
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
gxyz -7.0000 0.7000 -2.1000
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
gxyz -7.0000 0.7000 -2.1000
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
gxyz -7.0000 0.7000 -2.1000
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
gxyz -7.0000 0.0000 0.7000
x 7.0000
c 0.7 0.7 0.7
p rgb 0.7 0.7 0.7
mxyz 6.8000 -0.5000 0.0000
box 0.1000 3.0000 3.0000
c 0.7 0.7 0.7
p rgb 0.7 0.7 0.7
p chiller
mxyz 8.0000 0.5000 0.0500

```

```
box 1.5000 1.5000 1.8000
c 0 0.4 1
p rgb 0 0.4 1
mxyz 8.7500 1.2500 1.8500
d 1.2
z 0.1000
c 0 0.8 0
p rgb 0 0.8 0
p dispenser_canopy_and_supports
mxyz 4.0000 -14.0000 4.5000
box 12.0000 7.0000 0.5000
c 0.7 0.7 0.7
p rgb 0.7 0.7 0.7
mxyz 6.0000 -11.5000 0.0000
d 0.3
z 4.5000
gxyz 0.0000 2.0000 -4.5000
z 4.5000
gxyz 8.0000 0.0000 -4.5000
z 4.5000
gxyz 0.0000 -2.0000 -4.5000
z 4.5000
p dispenser
c 0 0.4 1
p rgb 0 0.4 1
mxyz 10.0000 -11.7500 0.0000
d 0.15
z 1.2000
gxyz 3.5000 0.0000 -1.2000
z 1.2000
gxyz -6.2500 0.7500 -1.2000
box 1.5000 0.7500 3.0000
gx 3.7500
box 1.5000 0.7500 3.0000
gxyz -2.7500 -0.7500 0.0000
z 1.2000
gxyz -1.7500 0.0000 -1.2000
z 1.2000
gxyz 5.1500 0.0000 -1.2000
z 1.2000
p tube_trailer
c 0.95 0.95 0.95
p rgb 0.95 0.95 0.95
mxyz -13.0000 -8.0000 0.7500
```

```

box  2.3000  12.0000  0.3500
c  0.1  0.1  0.1
p rgb 0.1  0.1  0.1
mxyz -13.0000 -7.6000  0.3750
d 0.74
x  0.4000
gxyz -0.4000  0.9000  0.0000
x  0.4000
gxyz  1.5000 -0.9000  0.0000
x  0.4000
gxyz -0.4000  0.9000  0.0000
x  0.4000
gxyz -2.3000  4.4000  0.0000
x  0.4000
gxyz -0.4000  0.9000  0.0000
x  0.4000
gxyz  1.5000 -0.9000  0.0000
x  0.4000
gxyz -0.4000  0.9000  0.0000
x  0.4000
gxyz -2.3000  3.7500  0.0000
x  0.4000
gx  1.5000
x  0.4000
c 0.95 0.95 0.95
p rgb 0.95 0.95 0.95
mxyz -10.7000  0.8500  1.0950
box -2.3000  3.1500  1.0000
gz  1.0000
box -2.3000  2.3000  0.5000
gz  0.5000
box -2.3000  1.0000  0.5000
c 0.8 0.8 0.85
p rgb 0.8 0.8 0.85
mxyz -12.5500 -7.9500  1.4450
d 0.7
y  7.0000
gxyz  0.0000 -7.0000  0.7000
y  7.0000
gxyz  0.0000 -7.0000  0.7000
y  7.0000
gxyz  0.7000 -7.0000 -1.4000
y  7.0000
gxyz  0.0000 -7.0000  0.7000

```



```

y 7.0000
gxyz 0.0000 -7.0000 0.7000
y 7.0000
gxyz 0.7000 -7.0000 -1.4000
y 7.0000
gxyz 0.0000 -7.0000 0.7000
y 7.0000
gxyz 0.0000 -7.0000 0.7000
y 7.0000
c 0.3 0.3 0.3
p rbg 0.3 0.3 0.3
p tube_trailer_enclosure
mxyz -4.0000 4.0000 0.0000
box -3.0000 0.1000 4.0000
poro 0.0000 0.5000 0.5000 0.0000
gx -3.0000
box -2.0000 0.1000 4.0000
poro 0.0000 0.0000 0.0000 0.0000
gx -2.0000
box 0.1000 -13.0000 4.0000
gxyz 5.0000 -13.0000 0.0000
box -5.0000 0.1000 4.0000
p gas_bottle_rack_3
c 0.7 0.7 0.7
p rbg 0.7 0.7 0.7
mxyz -6.1000 3.3000 0.0000
box 2.0000 0.6500 2.0000
c 0.7 0.7 0.7
p rbg 0.7 0.7 0.7
p car
c 0.1 0.3 0.7
p rbg 0.1 0.3 0.7
mxyz 8.8000 -13.5000 0.7000
box 3.0000 -1.8000 1.2000
box -1.2000 -1.8000 0.5000
c 0.05 0.05 0.05
p rbg 0.05 0.05 0.05
mxyz 8.5000 -13.8000 0.3500
y 0.3000
gx 2.5000
y -0.3000
gy -1.5000
y 0.3000
gx -2.5000

```

```
y -0.3000
c 0.85 0.85 0.9
p rbg 0.85 0.85 0.9
p low_storage
mxyz -5.5000 -7.0000 1.2000
d 1.5
y 8.0000
gx -1.6000
y -8.0000
gxyz -0.7500 1.0000 -0.7500
box 1.5000 0.4000 -0.4500
gy 5.6000
box 1.5000 0.4000 -0.4500
gx 1.6000
box 1.5000 0.4000 -0.4500
gy -5.6000
box 1.5000 0.4000 -0.4500
gxyz -1.7000 7.0000 -0.4500
box 3.5000 0.1000 2.1500
c 0.4 0.4 0.4
p rbg 0.4 0.4 0.4
mxyz -15.0000 -16.0000 0.0000
box 32.0000 23.0000 -0.1000
```

Appendix D

List of simulations

The list provided here refers to the combinations of variables that were considered in this project. Each of the simulations was run twice: ignited and unignited.

		ventilation	leakPosition	p	folder	diam	case	direction
natural ventilation	430 bar	naturalVent	(-2.5, -3.75, 1.2)	430	1area	0.0012	case1	(0,-1,0)
		naturalVent	(-2.5, -3.75, 1.2)	430	1area	0.0012	case2	(0,0,+1)
		naturalVent	(-2.5, -3.75, 1.2)	430	1area	0.0012	case3	(0,0,-1)
		naturalVent	(-2.5, -3.75, 1.2)	430	10area	0.0038	case1	(0,-1,0)
		naturalVent	(-2.5, -3.75, 1.2)	430	10area	0.0038	case2	(0,0,+1)
		naturalVent	(-2.5, -3.75, 1.2)	430	10area	0.0038	case3	(0,0,-1)
		naturalVent	(-2.5, -3.75, 1.2)	430	fullBore	0.0120	case1	(0,-1,0)
		naturalVent	(-2.5, -3.75, 1.2)	430	fullBore	0.0120	case2	(0,0,+1)
		naturalVent	(-2.5, -3.75, 1.2)	430	fullBore	0.0120	case3	(0,0,-1)
	630 bar	naturalVent	(-2.5, -1.7, 1.0)	630	1area	0.0010	case1	(+1,0,0)
		naturalVent	(-2.5, -1.7, 1.0)	630	1area	0.0010	case2	(-1,0,0)
		naturalVent	(-2.5, -1.7, 1.0)	630	1area	0.0010	case3	(0,-1,0)
		naturalVent	(-2.5, -1.7, 1.0)	630	10area	0.0032	case1	(+1,0,0)
		naturalVent	(-2.5, -1.7, 1.0)	630	10area	0.0032	case2	(-1,0,0)
		naturalVent	(-2.5, -1.7, 1.0)	630	10area	0.0032	case3	(0,-1,0)
		naturalVent	(-2.5, -1.7, 1.0)	630	fullBore	0.0100	case1	(+1,0,0)
		naturalVent	(-2.5, -1.7, 1.0)	630	fullBore	0.0100	case2	(-1,0,0)
		naturalVent	(-2.5, -1.7, 1.0)	630	fullBore	0.0100	case3	(0,-1,0)
	900 bar	naturalVent	(-2.5, 0.4, 0.7)	900	1area	0.0010	case1	(-1,0,0)
		naturalVent	(-2.5, 0.4, 0.7)	900	1area	0.0010	case2	(0,1,0)
		naturalVent	(-2.5, 0.4, 0.7)	900	1area	0.0010	case3	(0,-1,0)
		naturalVent	(-2.5, 0.4, 0.7)	900	10area	0.0025	case1	(-1,0,0)
		naturalVent	(-2.5, 0.4, 0.7)	900	10area	0.0025	case2	(0,1,0)
		naturalVent	(-2.5, 0.4, 0.7)	900	10area	0.0025	case3	(0,-1,0)
		naturalVent	(-2.5, 0.4, 0.7)	900	fullBore	0.0080	case1	(-1,0,0)
		naturalVent	(-2.5, 0.4, 0.7)	900	fullBore	0.0080	case2	(0,1,0)
		naturalVent	(-2.5, 0.4, 0.7)	900	fullBore	0.0080	case3	(0,-1,0)
forced ventilation	430 bar	forcedVent	(-2.5, -3.75, 1.2)	430	1area	0.0012	case1	(0,-1,0)
		forcedVent	(-2.5, -3.75, 1.2)	430	1area	0.0012	case2	(0,0,+1)
		forcedVent	(-2.5, -3.75, 1.2)	430	1area	0.0012	case3	(0,0,-1)
		forcedVent	(-2.5, -3.75, 1.2)	430	10area	0.0038	case1	(0,-1,0)
		forcedVent	(-2.5, -3.75, 1.2)	430	10area	0.0038	case2	(0,0,+1)
		forcedVent	(-2.5, -3.75, 1.2)	430	10area	0.0038	case3	(0,0,-1)
		forcedVent	(-2.5, -3.75, 1.2)	430	fullBore	0.0120	case1	(0,-1,0)
		forcedVent	(-2.5, -3.75, 1.2)	430	fullBore	0.0120	case2	(0,0,+1)
		forcedVent	(-2.5, -3.75, 1.2)	430	fullBore	0.0120	case3	(0,0,-1)
	630 bar	forcedVent	(-2.5, -1.7, 1.0)	630	1area	0.0010	case1	(+1,0,0)
		forcedVent	(-2.5, -1.7, 1.0)	630	1area	0.0010	case2	(-1,0,0)
		forcedVent	(-2.5, -1.7, 1.0)	630	1area	0.0010	case3	(0,-1,0)
		forcedVent	(-2.5, -1.7, 1.0)	630	10area	0.0032	case1	(+1,0,0)
		forcedVent	(-2.5, -1.7, 1.0)	630	10area	0.0032	case2	(-1,0,0)
		forcedVent	(-2.5, -1.7, 1.0)	630	10area	0.0032	case3	(0,-1,0)
		forcedVent	(-2.5, -1.7, 1.0)	630	fullBore	0.0100	case1	(+1,0,0)
		forcedVent	(-2.5, -1.7, 1.0)	630	fullBore	0.0100	case2	(-1,0,0)
		forcedVent	(-2.5, -1.7, 1.0)	630	fullBore	0.0100	case3	(0,-1,0)
	900 bar	forcedVent	(-2.5, 0.4, 0.7)	900	1area	0.0010	case1	(-1,0,0)
		forcedVent	(-2.5, 0.4, 0.7)	900	1area	0.0010	case2	(0,1,0)
		forcedVent	(-2.5, 0.4, 0.7)	900	1area	0.0010	case3	(0,-1,0)
		forcedVent	(-2.5, 0.4, 0.7)	900	10area	0.0025	case1	(-1,0,0)
		forcedVent	(-2.5, 0.4, 0.7)	900	10area	0.0025	case2	(0,1,0)
		forcedVent	(-2.5, 0.4, 0.7)	900	10area	0.0025	case3	(0,-1,0)
		forcedVent	(-2.5, 0.4, 0.7)	900	fullBore	0.0080	case1	(-1,0,0)
		forcedVent	(-2.5, 0.4, 0.7)	900	fullBore	0.0080	case2	(0,1,0)
		forcedVent	(-2.5, 0.4, 0.7)	900	fullBore	0.0080	case3	(0,-1,0)