

**NIST Technical Note 2163**

# **P-Flash – A Machine Learning based Flashover Prediction Model to Enable Smart Firefighting for Compartment Fires**

Wai Cheong Tam  
Jun Wang  
Richard Peacock  
Paul Reneke  
Eugene Yujun Fu  
Thomas Cleary

This publication is available free of charge from:  
<https://doi.org/10.6028/NIST.TN.2163>

**NIST**  
**National Institute of  
Standards and Technology**  
U.S. Department of Commerce

## NIST Technical Note 2163

# P-Flash – A Machine Learning based Flashover Prediction Model to Enable Smart Firefighting for Compartment Fires

Wai Cheong Tam  
Richard Peacock  
Paul Reneke  
Thomas Cleary

*Fire Research Division, National Institute of Standards and Technology*

Jun Wang  
Eugene Yujun Fu  
*Department of Computing, The Hong Kong Polytechnic University*

This publication is available free of charge from:  
<https://doi.org/10.6028/NIST.TN.2163>

June 2021



U.S. Department of Commerce  
Gina M. Raimondo, Secretary

National Institute of Standards and Technology  
*James K. Olthoff, Performing the Non-Exclusive Functions and Duties of the Under Secretary of Commerce  
for Standards and Technology & Director, National Institute of Standards and Technology*

Certain commercial entities, equipment, or materials may be identified in this document in order to describe an experimental procedure or concept adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.

**National Institute of Standards and Technology Technical Note 2163**  
**Natl. Inst. Stand. Technol. Tech. Note 2163, 52 pages (June 2021)**  
**CODEN: NTNOEF**

**This publication is available free of charge from:**  
**<https://doi.org/10.6028/NIST.TN.2163>**

[Intentionally Left Blank]

## **Abstract**

This report provides additional technical details to an article entitled *P-Flash – A Machine Learning-based Model for Flashover Prediction using Recovered Temperature Data*.

Research was conducted to examine the use of Support Vector Regression (SVR) to build a model to forecast the potential occurrence of flashover in a single-floor, multi-room compartment fire. Synthetic temperature data for heat detectors in different rooms were generated, 1000 simulation cases are considered, and a total of 8 million data points are utilized for model development. An operating temperature limitation is placed on heat detectors where they fail at a fixed exposure temperature of 150 °C and no longer provide data to more closely follow actual performance.

The forecast model, P-Flash (Prediction model for Flashover occurrence), is developed to use an array of heat detector temperature data, including in adjacent spaces, to recover temperature data from the room of fire origin and predict potential for flashover. Two special treatments, sequence segmentation and learning from fitting, are proposed to overcome the temperature limitation of heat detectors in real-life fire scenarios and to enhance prediction capabilities to determine if the flashover condition is met even with situations where there is no temperature data from all detectors. Experimental evaluation shows that P-Flash offers reliable prediction. The model performance is approximately 83 % and 81 %, respectively, for current and future flashover occurrence, considering heat detector failure at 150 °C. Results demonstrate that P-Flash, a new data-driven model, has potential to provide fire fighters real-time, trustworthy, and actionable information to enhance situational awareness, operational effectiveness, and safety for firefighting.

## **Key words**

Machine learning; flashover prediction; fire modeling; heat detector; smart firefighting.

[Intentionally Left Blank]

## Table of Contents

<b>1. Introduction .....</b>	<b>1</b>
<b>2. Data Generation.....</b>	<b>2</b>
2.1. Numerical Setup .....	2
2.2. CData Settings .....	4
2.3. Data Profiles .....	5
<b>3. Model Development of P-Flash .....</b>	<b>6</b>
3.1. Sequence Segmentation .....	6
3.2. Feature Extraction .....	8
3.3. Training and Testing .....	<b>Error! Bookmark not defined.</b> 0
3.3.1. Regression Models.....	<b>Error! Bookmark not defined.</b> 1
3.3.2. Learning from Fitting .....	<b>Error! Bookmark not defined.</b> 2
<b>4. Results and Discussion .....</b>	<b>13</b>
<b>5. Conclusions and Outlook .....</b>	<b>15</b>
<b>References .....</b>	<b>15</b>
<b>Appendix A: CData Input File .....</b>	<b>17</b>
<b>Appendix B: P-Flash Source Codes.....</b>	<b>171</b>
<b>Appendix B.1: Main .....</b>	<b>172</b>
<b>Appendix B.2: Read Data .....</b>	<b>175</b>
<b>Appendix B.3: Data Pre-processing and Feature Extraction.....</b>	<b>177</b>
<b>Appendix B.4: Training and Testing .....</b>	<b>43</b>
<b>Appendix B.5: Fitting.....</b>	<b>47</b>
<b>Appendix C: P-Flash Limitations.....</b>	<b>51</b>

[Intentionally Left Blank]



## List of Tables

Table 1. Summary of thermal properties and geometric configurations. ....	3
Table 2. MRND that specifies random number generator for peak HRR and time to peak HRR. ....	4
Table 3. Summary of extracted features. ....	9
Table 4. Performance summary for P-Flash. ....	<b>Error! Bookmark not defined.</b> 4
Table 5. Overall model accuracy with early and late prediction for potential occurrence of flashover.....	<b>Error! Bookmark not defined.</b> 5
Table B1. P-Flash performance again new test set for current prediction.....	52

## List of Figures

Fig. 1. Schematic of the single-story three compartments with a fire in action. ....	3
Fig. 2. Scatter plot for peak HRR vs time to peak. ....	4
Figs. 3. Room 1, Corridor, and Room 2 temperature profiles for a) a fast growth fire with low peak HRR case and b) a medium growth fire with high peak HRR case. ....	5
Fig. 4. Mean detector temperature profiles and its deviation in different compartments. ....	6
Fig. 5. Machine learning pipeline for P-Flash (from raw data to feature extraction). ....	7
Fig. 6. The overview of model architecture for P-Flash. ....	<b>Error! Bookmark not defined.</b> 0

## No table of figures entries found.

Figs. 8. Comparison between ground truth and predictions obtained from P-Flash with and without LFF. ....	<b>Error! Bookmark not defined.</b> 3
---	---------------------------------------

## No table of figures entries found.

Fig. B2. Comparison between ground truth and predictions (current) obtained from P-Flash for Case 3. ....	52
---	----

[Intentionally Left Blank]

## 1. Introduction

Over the five-year period from 2013 – 2017, the fire departments in the United States responded to an average of 500,000 structure fires annually [1]. These fires resulted in approximately 2,500 civilian fire deaths, 14,000 civilian fire injuries, and more than \$10 billion dollars in direct property losses. In addition, more than 31,000 firefighters were injured, and approximately 360 of them were killed on the fire ground [2]. Statistics show that rapid fire development caused by extreme fire behaviors such as flashover is identified as one for the major causes of fatal injury for fire fighters during structural firefighting. Although flashover conditions (i.e., hot layer gas temperature approximately 600 °C and/or average heat flux at the floor level reaching 20 kW/m<sup>2</sup>) are well known in the fire research community, this kind of detailed information about the interior thermal conditions is usually unavailable. It is rather difficult for fire fighters to understand the potential fire hazards inside the compartment. In a structural fire, rollover [3] is one possible indicator. Visually, it can be seen as flames rolling across the ceiling. When rollover phenomenon is observed, a potential flashover is likely to occur. However, this extreme fire indicator is not easy to recognize, and it could take many years of experience to build up the necessary proficiency. Therefore, if fire fighters do not have such a high level of situational awareness, the flashover threat presents itself as an unpredictable life-threatening hazard.

Several research efforts have been conducted to develop data-driven models that can estimate the heat release rate (HRR) based on information obtained from sensors in real-time. Davis and Forney [4] developed an inverse fire model based on empirical correlations. Provided the estimated HRR, the location of the fire and the fire size could be obtained using an inverse modeling technique. Yet, the model is only suitable for one-room compartments. Based on a generic algorithm, Neviackas and Trouvé [5] obtained a generalized HRR which can be used to determine flashover conditions in multi-room geometries. Overholt and Ezekoye [6] also developed an inverse model using a predictor-corrected method. Based on smoke layer temperature measurements, the prediction accuracy of the model was shown to be within 60 s. However, a challenging problem exists in which all models [4-6] rely on complete measurement data sets acquired using laboratory equipment. In practical situations, sensors such as heat and/or smoke detectors will stop functioning at a certain elevated temperature [7]. If the required temperature/smoke data is missing, the estimated HRR obtained from these models will become highly uncertain and presumably, the prediction of flashover occurrence based on the estimated HRR will be unreliable.

Unlike the previous attempts [4-6], the temperature limitation for sensors, such as heat detectors, are considered in this present work with the objective to develop a machine learning model, P-Flash (Prediction model for Flashover occurrence) that can predict the flashover occurrence even with missing temperature data due to malfunctioning heat detectors. In the next section, the synthetic data being used to develop the model will first be described. Then, the model development of P-Flash will be presented. In order to demonstrate the prediction capability of P-Flash, two study cases are included, and Section 4 provides results and discussion. Section 5 provides additional model testing to highlight the current model limitation. Finally, some concluding remarks on P-Flash and future work are presented in Section 6.

## 2. Data Generation

Scarcity of real-world data from building sensors during fire events is one of the challenges for the use of the machine learning (ML) paradigm. The data problem has been raised in different literature, such as [7]. For the fire safety community, it can be noted that acquiring the desired sensor data is not trivial because 1) fire events do not happen frequently, 2) time series data associated with fire events in building environments are not available to the public data warehouse [8], and 3) physically conducting full-scale fire experiments in buildings such as [9] is extremely costly and time-consuming. Moreover, no prior research work has been carried out to provide guidance on the data requirements for ML applications. With that, there may be a high probability that the obtained experimental data is not usable. When the conventional ML paradigm demands a large amount of training data, the CFAST Fire Data Generator (CData) [10] is utilized to generate synthetic data to facilitate the use of ML paradigms for prediction of fire hazards in buildings.

In general, CData is a computational tool with its front-end written in Python<sup>1</sup>. The code was developed to generate time series data for typical devices/sensors (i.e., heat detector, smoke detector, and other targets) in any user-specified fire environments within a building structure. CFAST [11] is used as the simulation engine in CData for two reasons. First, the fire simulation program is mathematically verified and is validated with experimental data [12]. The verification and validation (V&V) process is an active and continuous effort at the National Institute of Standards and Technology (NIST) to ensure the fidelity of the code. Second, CFAST is numerically efficient. Using the Fire Research Division computer cluster at NIST, more than ten thousand simulation cases with various geometric and fire configurations specified in this study can be completed in a single day. This advantage provides the flexibility and capability to conduct parametric studies for obtaining the most relevant and high quality synthetic data set for researching the use of ML paradigms. It should be noted that the authors<sup>2</sup> understood the inherent model assumptions being made in CFAST. When the feasibility of using synthetic data for the development of ML models is warranted, a more sophisticated fire simulation program, such as the Fire Dynamic Simulator [13], and/or even full-scale experimental data can be utilized in future studies. These higher fidelity data would allow the ML model to account for other realistic conditions, such as the effect of hot gas movement to the detectors, to improve the model performance.

### 2.1. Numerical Setup

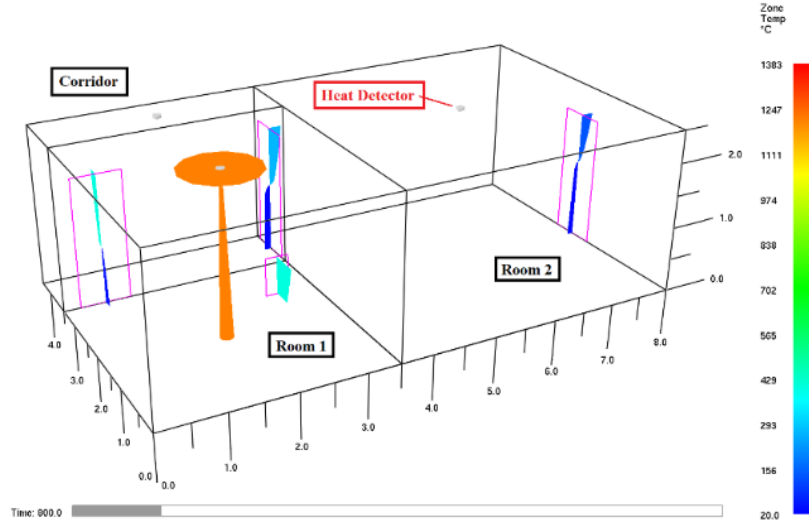
Consider a single-story building with three compartments as shown in Fig. 1. The dimensions of Room 1 are 3.5 m x 3.5 m, and the dimensions of Room 2 and Corridor are 4.5 m x 4.5 m and 3.5 m x 1 m, respectively. The ceiling height is 2.5 m, and it is identical for all compartments. For simplicity, the material of all walls, ceilings, and floors is gypsum wallboard. As seen in Fig. 1, there are 4 openings: 1) a window in Room 1, 2) a door between Room 1 and Corridor, 3) a door between Corridor and Room 2, and 4) an exit-door in Room 2. The openings are fully opened. There is one heat detector in every compartment, and they are all located at the center of each compartment about 4.5 cm away from the ceiling. The

---

<sup>1</sup> Certain commercial equipment, instruments, or materials are identified in this paper in order to specify the procedures adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose.

<sup>2</sup> Richard Peacock and Paul Reneke at NIST are both the principal investigators for continuous development and maintenance of CFAST.

response time index for the heat detector is  $35 \text{ (m}\cdot\text{s)}^{0.5}$ . The outdoor conditions are typical with the temperature at  $20^\circ\text{C}$  and atmospheric pressure of  $101 \text{ kPa}$ . Table 1 provides the summary of the thermal properties of the gypsum wallboard and the geometric configurations of the openings.



**Fig 1.** Schematic of the single-story three compartments with a fire in action.

**Table 1.** Summary of thermal properties and geometric configurations.

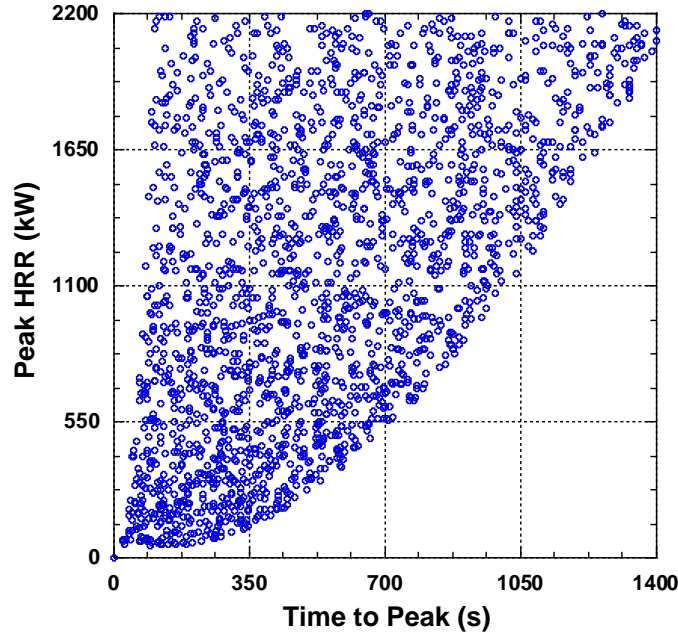
	Conductivity (W/[m·K])	Specific heat (J/[kg·K])	Density (kg/m <sup>3</sup> )	Thickness (m)	Emissivity (-)
<b>Gypsum</b>	0.276	1.017	752	0.0159	0.94

	From	To	Width	Length	Distance away from ceiling
	(-)	(-)	(m)	(m)	(m)
<b>Window</b>	Room 1	Exterior	0.3	0.5	0.5
<b>Door 1</b>	Room 1	Corridor	0.75	2	0.5
<b>Door 2</b>	Corridor	Room 2	0.75	2	0.5
<b>Exit door</b>	Room 2	Exterior	0.75	2	0.5

Given the experimental setup, a t-squared fire is placed at the center in Room 1. In this study, simple three-stage t-squared fires are considered. It has a growing stage, a plateau, and a decay stage. Basically, a fire will grow at a rate proportional to the time raised to the second power. When the fire reaches its peak, it will sustain for some time (denoted as plateau). It then dies down (denoted as fire decay) and is extinguished. Based on references provided in [14,15], a range of fires are selected to describe the fire growing stage. Figure 2 shows the scatter plot of peak heat release rate (HRR) and time to peak for the 1000 cases. It can be seen that the peak HRR and the time to peak ranges from approximately  $50 \text{ kW}$  to  $2200 \text{ kW}$  and from  $50 \text{ s}$  to

1400 s, respectively. The selected range of peak HRR and time to peak cover various burning items from an office trash can with a slow fire growth rate to an upholstered furniture fire with an ultra-fast fire growth rate. In terms of duration for plateau and fire decay, they are assumed to be constant. The duration for plateau and fire decay are set to be 2000 s and 1500 s, respectively.



**Fig. 2.** Scatter plot for peak HRR vs time to peak.

## 2.2. CData Settings

This subsection demonstrates how CData is utilized to configure fire cases as mentioned in Sec 2.1. Appendix A shows the complete CData input file being used in this study. In general, the CData input file has similar namelists to that of CFAST input files except that CData has additional namelists to facilitate data sampling. Based on the descriptions provided in the previous section, there are two varying conditions for fires: i) the peak HRR and ii) the time to peak HRR. In order to sample the desired fire conditions, a namelist, MRND, is used. As shown in the appendix, two lines of code are involved to sample fire conditions uniformly across the domain of interest. The corresponding information is summarized in the following table:

**Table 2.** MRND that specifies random number generator for peak HRR and time to peak HRR.

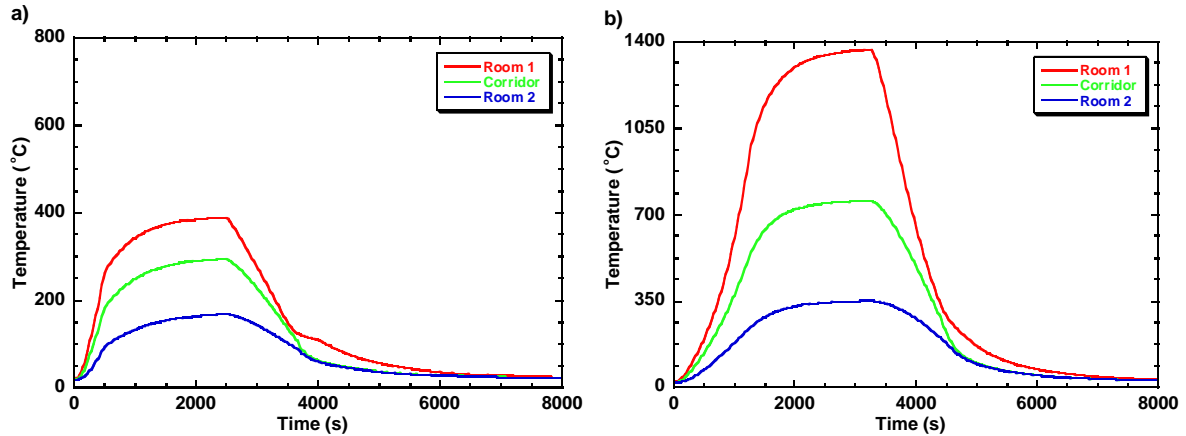
<b>MRND</b>				
<b>ID</b>	<b>Distribution Type</b>	<b>Value Type</b>	<b>Minimum</b>	<b>Maximum</b>
Peak HRR Generator	Uniform	Real	50 000	2 200 000
End of Growth Time Generator	Uniform	Real	75	1400

There are five parameters in MRND: 1) ID, 2) distribution type, 3) value type, 4) minimum, and 5) maximum. The first parameter defines the unique name of the desired generator. The second parameter specifies the required distribution function which would be used for sampling. In the current version of CData [10], it supports 8 different well-defined distribution functions, such as uniform, triangle, normal, truncate normal, log normal, truncated log normal, beta, and linear. Since a set of uniformly distributed fire cases are needed to facilitate the model training, the uniform distribution is utilized. For value type, real number is chosen because the bounding conditions are expected to be defined by numerical values. Lastly, minimum and maximum specify the lower bound and the upper bound of the fire conditions, respectively.

The random number generators, Peak HRR Generator and End of Growth Time Generator, specify the fire growing stage. In order to construct the three-stage t-squared fires, another namelist, MFIR, is utilized to combine a number of MRND namelists to generate the namelist that CFAST would recognize. Specifically, the plateau and the fire decay are specified using Peak HRR (2<sup>nd</sup> value) and Plateau End Time, and End of Fire HRR and Fire End Time, respectively. Since the total number of 1000 cases<sup>3</sup> are required, the parameter NUBMBER\_OF\_CASES from MHDR namelist is set to be 1000.

### 2.3. Data Profiles

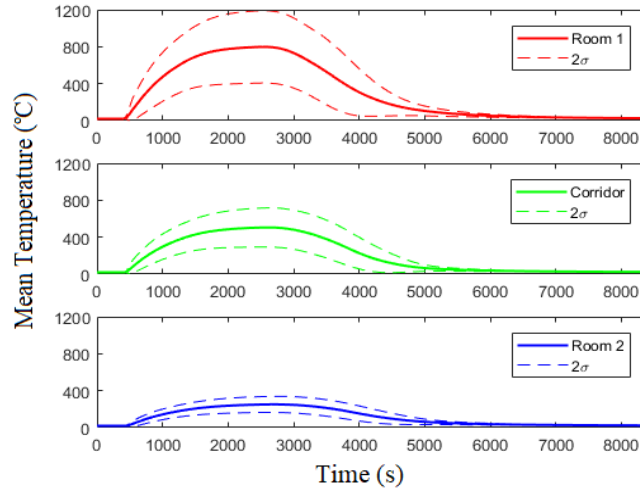
For all simulation runs, a fire is started in Room 1. Subsequently, the upper layer gas temperature rises, and the layer thickness increases. Some hot gases leave the building structure, and some flow through the door. Air mixing between Room 1 and Corridor occurs. Due to the mixing, the upper gas layer temperature in Corridor also increases. Similar mass transfer and heat transfer processes take place between Corridor and Room 2, and the Room 2 upper gas layer temperature gradually rises. Figs. 3 show Room 1, Corridor, and Room 2 temperature profiles for two selecting cases: a) a fast growth fire with low peak HRR case and b) a medium growth fire with high peak HRR case. The total simulation time for each simulation run is 8400 s, and the temperature output interval is 20 s.



**Fig. 3.** Room 1, Corridor, and Room 2 temperature profiles for a) a fast growth fire with low peak HRR case and b) a medium growth fire with high peak HRR case.

<sup>3</sup> The selected number of simulation runs was determined based on a parametric study. Five sets (100, 500, 1000, 2000, and 5000 cases) of data were considered. Using the experimental setup mentioned in Section 2.1, the model performance for the prediction of flashover achieves convergence when the number of cases reaches 1000 cases. The full dataset including all input files associated with the 1000 cases can be found at <https://doi.org/10.18434/M32258>.

Figure 4 shows the mean temperature profiles for the three detectors as a function of time. As shown in the figure, the temperature profiles in Corridor and Room 2 are lower than that of Room 1, and the dashed lines represent two times the standard deviation of detector temperature profiles. Although this study uses only temperature data, in principle other time series data such as smoke concentration obtained from smoke detectors, can also be used for the model development. Moreover, building structures with different compartments (in terms of quantity, orientation, and door connection) and fires involving various fire growth behavior can also be considered in the data generation so that a more generalized ML model can be developed for actual use. This research effort is under way, and the findings will be reported in future publications.



**Fig. 4.** Mean detector temperature profiles and its deviation in different compartments.

### 3. Model Development of P-Flash

Given a set of data, two additional steps including data preprocessing and model training are required for the development of P-Flash. Fig. 4 depicts the processes associated with the data preprocessing and the corresponding main codes are provided in Appendix B1.

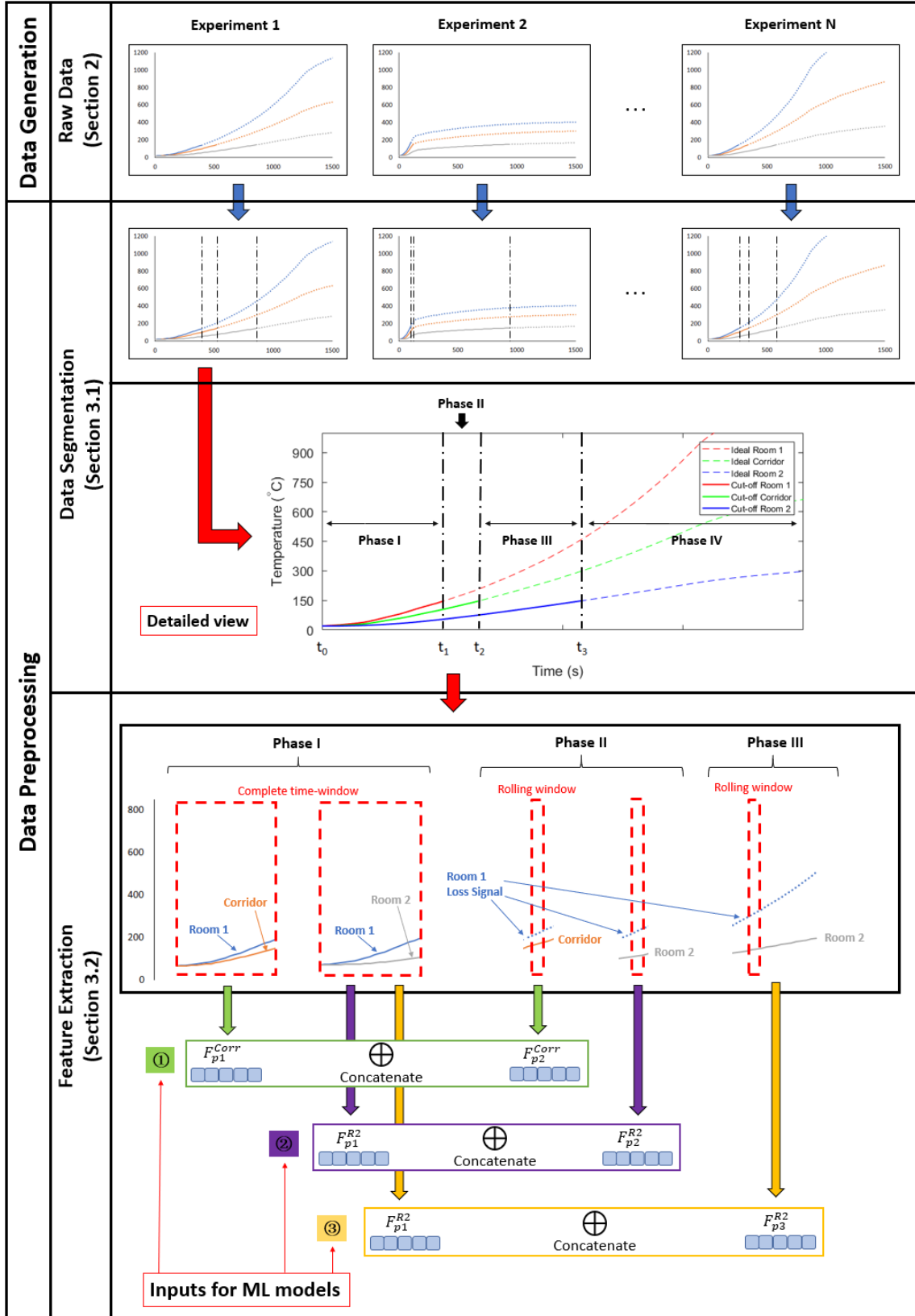
#### 3.1. Sequence Segmentation

Loss of detector temperature signal is one of the primary difficulties for the development of an accurate machine learning-based flashover prediction model. For actual fire scenarios, heat detectors cannot survive at elevated temperature [7] and would fail at temperatures well below the estimated flashover temperature ( $\sim 600^\circ\text{C}$ ). It is well known that developing a ML model based on unphysical data significantly jeopardizes the model performance. With the malfunctioning detectors, the temperature can be unphysical, and a special treatment is needed to preprocess the data such that unphysical data can be eliminated.

Knowing that detectors<sup>4</sup> stop functioning at elevated temperature (here assumed as  $150^\circ\text{C}$ ), the detailed view shown in Fig. 5 presents the temperature profiles from ideal detectors (dashed

<sup>4</sup> In reference [16], there are 7 classes for heat detector. Each class has different maximum operating temperature range at the ceiling. The selected cut-off temperature ( $150^\circ\text{C}$ ) is based on the *extra high* class heat detector.





**Fig. 5.** Machine learning pipeline for P-Flash (from raw data to feature extraction).

lines) and those with a cut-off temperature at 150°C (solid lines) for a simulation run with a fast-growth fire originating in Room 1. As shown in the plot, the available data for the detector in Room 1 is limited. At  $t_l$ , the temperature signal from Room 1 is lost. For simplicity, when the temperature signal is lost, the temperature is artificially turned into a constant in this study (i.e., a value of zero). In general, it is rather difficult for any models, even ML models, to provide any reliable flashover prediction with these limited temperature data (i.e., up to only 150°C). However, it is seen that temperature signals from other compartments do exist. Given this observation, it is believed that the use of the available temperature data from other compartments helps to “recover” the detector temperature in Room 1 which can be used to determine the flashover condition. In order to facilitate this process, a sequence segmentation is applied to the temperature data set.

Using the sequence segmentation, a new data structure is laid out. As shown in same plot, there are 3 vertical lines, dividing the temperature profiles into 4 phases. Each of the phases contains different available temperature signals. For example, signals from all detectors are present in Phase I ( $t_0 - t_1$ ). In Phase II ( $t_1 - t_2$ ), signals from only Corridor and Room 2 are available. In Phase III ( $t_2 - t_3$ ), the last available signals are from Room 2. No temperature signals exist in Phase IV, and additional treatment is needed to facilitate the prediction of flashover conditions. Three benefits are found from using the segmented data: 1) the unphysical information due to any malfunctioning detectors is eliminated, 2) the ML model can take full advantage of the available data associated with a specified phase, and 3) the new data structure provides the basis for the model development of P-Flash. It is worth noting that only temperature data less than or equal to 150°C are used for model development.

### 3.2. Feature Extraction

Feature extraction [17] is an essential ML task to facilitate the development of a model. In this process, the raw data (i.e., discrete temperature data which is uncorrelated in time) is transformed into a data set with a reduced number of variables which contains more discriminative information. An example of discriminative information can be the rate of change of temperature which relates the temperature increase over a period of time. It can be understood that a large rate of change in temperature indicates a higher chance of having more rapid fire growth which would possibly lead to a flashover if sufficient oxygen is available and the fire continues to grow. This higher level information facilitates the learning process for a ML model which helps develop a more accurate model.

The feature extraction section depicted in Fig. 5 shows the feature vectors<sup>5</sup>,  $F$ , being extracted from the detector temperature profiles in different phases, and there are five different feature vectors:  $F_{p1}^{Corr}$ ,  $F_{p1}^{R2}$ ,  $F_{p2}^{Corr}$ ,  $F_{p2}^{R2}$ , and  $F_{p3}^{R2}$ . In terms of notation, the superscript denotes the extracted features corresponding to the compartment, and the subscript denotes the extracted features associated with a specific phase. For general practice, the construction of features and the required number of feature vectors are based on three factors: 1) the structure of the data (refer to Section 3.1), 2) how often the prediction is needed, and 3) the architecture of the ML model.

In Phase I, since no prediction is required, the features are extracted based on a complete time-window with the intention of encoding the relationships among the temperatures associated

---

<sup>5</sup> A feature vector contains a number of different features.

with Room 1 ( $T^{R1}$ ), Corridor ( $T^{Corr}$ ), and Room 2 ( $T^{R2}$ ). In this study, two types of features are obtained, and they are temperature-based features and trend-based features. Table 3 provides a list of extracted features. In ML, the temperature based-features provide the overall statistics of the temperature data and the trend based-features provide the overall temperature behavior with respect of time. For example, the temperature-based feature, *mean of  $T^X(t_0:t_1)$* , can be understood as the average temperature in between  $t_0$  and  $t_1$ . For the trend-based feature, the  $dT^X/dt$  represents the first derivative of temperature which describes the rate of change of temperature over a period of time. The superscript  $X$  describes three different compartments: Room 1 (R1), Corridor (Corr), and Room 2 (R2). It should be noted that the differential time ( $dt$ ) being used to obtain the first derivative of the temperature is different than the length of the complete time-window. Since the overall behavior of the temperature profile is relatively smooth, the differential time is taken to be one time-step (20 s). As shown in the table, six different features are being extracted in Phase I, and they are added to form the feature vector.

**Table 3.** Summary of extracted features.

	Phase I	Phase II	Phase III
<b>Temperature-based features</b>	Mean and Max. of $T^X(t_0:t_1)$	Mean and Max. of $T^Y(t_i:t_{i+rolling\_window})$	Mean and Max. of $T^Z(t_i:t_{i+rolling\_window})$
<b>Trend-based features</b>	Min., Mean, and Max. of $dT^X/dt$	Min., Mean, and Max. of $dT^Y/dt$	Min., Mean, and Max. of $dT^Z/dt$
	Index of Max. of $dT^X/dt$ divided by length of fixed window	Index of Max. of $dT^Y/dt$ divided by length of rolling window	Index of Max. of $dT^Z/dt$ divided by length of rolling window

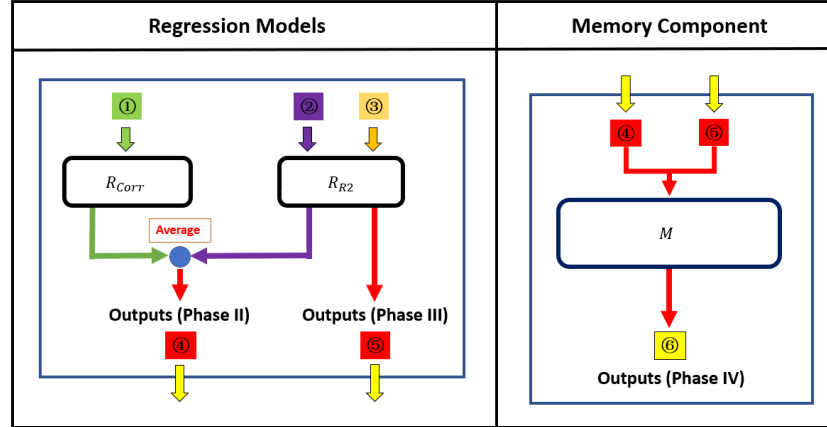
$X$  represents R1, Corr, and R2.  $Y$  represents Corr and R2 and  $Z$  represents R2. The abbreviation Min. and Max. denotes minimum and maximum, respectively.

In Phase II and III, although similar process is being carried out, the extracted features are obtained based on a rolling window [17]. Basically, the rolling window contains a sub-data set. After a feature extraction is executed, the window shifts onwards for one time-step. A numerical experiment is conducted to determine the optimal size of the rolling window. For real-time detection, the window size is taken to be six time-steps. In general, the use of rolling windows helps to provide extracted features containing more localized information. For Phase IV, since no temperature data is available, no features are being extracted. It is worth noting that the symbol,  $\oplus$ , as shown in Fig. 4 represents concatenation of two vectors. When the feature extraction process is complete, three feature vectors: ① =  $F_{p1}^{Corr} \oplus F_{p2}^{Corr}$ , ② =  $F_{p1}^{R2} \oplus F_{p2}^{R2}$ , and ③ =  $F_{p1}^{R2} \oplus F_{p3}^{R2}$ , are obtained, and they are used to train/develop the models for P-Flash. In the next section, the descriptions of model training are presented. It should also

be noted that feature selection, such as use of collinearity check and variable importance, can be made to select the features that contribute the most to the predictions. The corresponding source codes for data pre-processing and feature extraction are provided in Appendix B3.

### 3.3. Training and Testing

Figure 6 depicts the overview of the model architecture for P-Flash. P-Flash consists of two regression models ( $R_{Corr}$  and  $R_{R2}$ ) and a memory component ( $M$ ). The primary difference between the two models is that  $R_{Corr}$  is trained based on feature vector ①, and  $R_{R2}$  is trained based on feature vectors ② and ③. In theory, a single regression model might work. However, the training process for such a model involving more information is numerically more difficult and overfitting<sup>6</sup> might occur and this is attributed to the fact that all the temperature behaviors from three different sensors will have to be learned by only one regression model. Since using either of the approaches (two regression models or single regression model) will provide relatively the same prediction, the two regression model approach is utilized for training efficiency. The memory component,  $M$ , is a hybrid module: it performs as a storage to contain outputs from  $R_{Corr}$  and  $R_{R2}$  and provides temperature prediction of Room 1 based on the historical information. This model architecture provides robust and flexible prediction capabilities to adapt to more complex cases with a larger number and different types of detectors.



**Fig. 6.** The overview of model architecture for P-Flash.

Due to the fact that the model first sees temperature data of three compartments for Phase I and Phase II, both regression models,  $R_{Corr}$  and  $R_{R2}$ , are executed simultaneously and two separate temperature predictions for Room 1 in Phase II are obtained. In order to compensate for the numeric difference, averaging is conducted, and the temperature prediction is stored in the memory component. Since only the temperature in Room 2 exists in Phase III, only  $R_{R2}$  is executed and the temperature of Room 1 in Phase III is obtained. Similarly, the output is stored in the memory component. The ML algorithm being used for training and the details of model testing are provided in the next subsection.

<sup>6</sup> Overfitting is a modeling error that occurs when a function is too closely fit to a limited set of data points. For example, rather than learning the overall trend inherent to the data set, the model attempts to memorize the noise from the data.

### 3.3.1. Regression Models

Support vector regression (SVR) [18] is used to develop the two regression models ( $R_{corr}$  and  $R_{R2}$ ). Fundamentally, SVR finds a decision boundary, known as a hyperplane, to correlate data instances and maximizes the constrained margin such that the distance between the data instances is optimal to achieve greatest model generalizability. For example, given a training dataset  $T = \{(X_1, y_1), (X_2, y_2), \dots, (X_n, y_n)\}$  which can be linearly separated, the hyperplane denoted as  $p$  can be written as:

$$w \cdot X + b = 0 \quad (1)$$

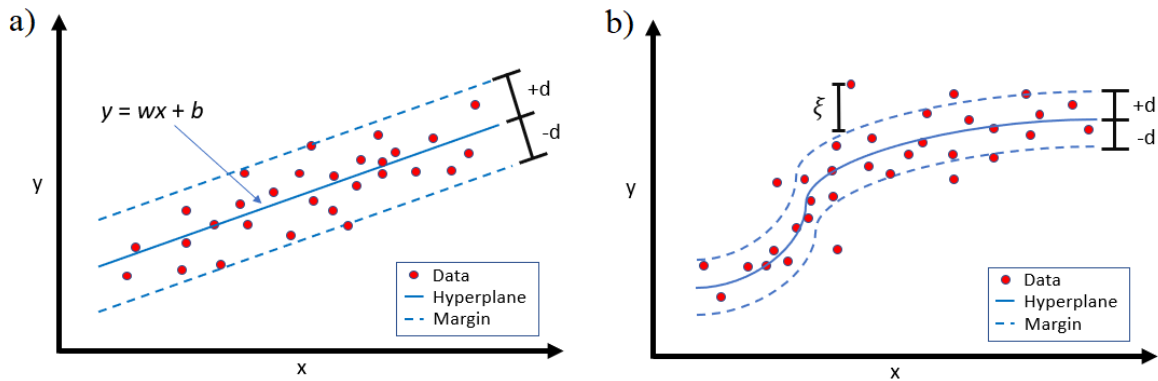
where  $X_n$  is the sample of  $n^{th}$  instance and  $y_n$  is the target/ground truth. In this study,  $X_n$  will be the three feature vectors and  $y_n$  will be the Room 1 temperature.  $w$  and  $b$  are the weight and the bias of the hyperplane, respectively. Based on the definition provided in [19], the distance between the instances for different classes is:

$$d = \min_{i=1,2,\dots,n} y_i \left( \frac{w}{\|w\|} \cdot X_i + \frac{b}{\|w\|} \right) \quad (2)$$

where  $\|w\|$  is norm of  $w$ . For SVR, the distance is known as margin. Therefore, SVR determines the hyperplane with the largest margin by solving the optimization problem:

$$\arg \max_{w,b} \left( \min_{i=1,2,\dots,n} y_i \left( \frac{w}{\|w\|} \cdot X_i + \frac{b}{\|w\|} \right) \right) \quad (3)$$

Figure 7a provides an example case with data in linear behavior. The hyperplane is obtained to best correlate the data and the margin is determined to include each data point. As shown in the figure, this form of SVR is similar to the best of fit with a simple linear regression.



**Figs. 7.** Example of a regression model from a) a linear SVR and b) non-linear SVR.

For real-life applications, fire data are often more complex, and they are not linearly separable (See Fig. 7b). In order to overcome the numerical difficulty, there are two options. The first option is called the “kernel trick” [19], and there are four commonly used nonlinear kernel

functions: 1) polynomial kernel, 2) Gaussian kernel, 3) radial basis function (RBF), and 4) sigmoid kernel. The use of a kernel function allows the transformation of data into a higher dimensional space such that the instances  $X_n$  for different classes separated by a hyperplane (i.e., nonlinear) exists. The second option is to introduce a regularization/slack variable. With the implementation for the regularization variable,  $C$ , a small proportion of the data are ignored (see  $\xi$  in Fig. 7b). Although there is trade-off for the use of these options, it generally provides a more generalized model and helps to avoid over-fitting. This implies the situation where the model only memorizes the data without obtaining any useful patterns and relationships for the data behavior. If over-fitting occurs, the model performance will be very poor.

In this study, a 5-fold cross validation method [8] is utilized to facilitate the training and testing process. In principle, the entire dataset from 1000 simulation runs is randomly divided into five subsets, and each subset/fold contains 200 sessions. In general, one fold of data is being used as testing data, and the remaining four folds are being used as training data. This process is carried out iteratively for five times until all five different folds of data are being used as the testing set. The trained regression models provide Room 1 temperature predictions in Phase II to Phase III. Utilizing grid search [8], the optimal configurations for SVR are  $C = 1000$  and  $\text{Gamma} = 0.05$  with RBF kernel. The corresponding source codes for the development of the two regression models are provided in Appendix B4.

### 3.3.2. Learning from Fitting

In Phase IV, since all detectors are lost, no inputs are available, and therefore no reliable predictions can be made from the regression models. In order to overcome this physical limit, learning from fitting is implemented to facilitate the extrapolation of the temperature in Room 1 using the historical data (i.e., the available temperature in Phase I and predicted temperature obtained in Phase II and III). Given the current set of data, there can be two possible scenarios in Phase IV: 1) a scenario where the predicted temperature of Room 1 is sufficiently long enough to observe a logarithmic temperature increase or 2) the fire is so large (in terms of peak HRR with short time to peak) that the temperature rise appears to be an exponential function. For that, two mathematical expressions, a sigmoidal binding function and a 5<sup>th</sup> order polynomial, are considered. The sigmoidal binding function is used for the first scenario:

$$p_i = (b\sqrt{t_i})/(\sqrt{t_i} + a) + c \quad (4)$$

whereas the high order polynomial is used for the second scenario:

$$p_i = d_5 t_i^5 + d_4 t_i^4 + d_3 t_i^3 + d_2 t_i^2 + d_1 t_i + d_0 \quad (5)$$

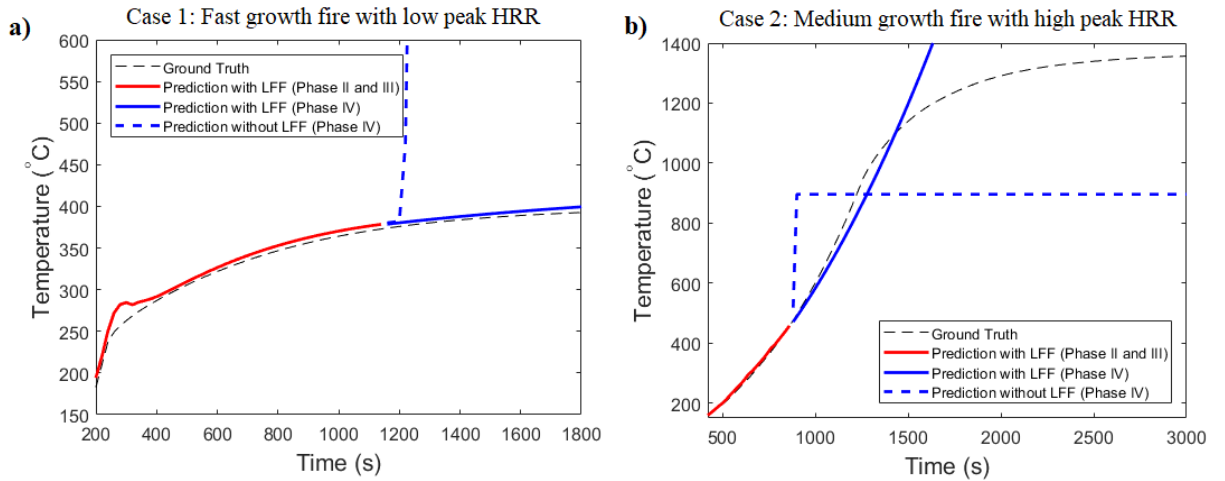
where  $p_i$  is the prediction and  $t_i$  is the time associated with index  $i$ . Optimization is carried out to obtain  $a$ ,  $b$ ,  $c$ , and  $d$  to produce a best fit to generalize the Room 1 temperature data in Phase I to III. Given the best fit, Room 1 temperature in Phase IV can be extrapolated. The corresponding source codes for carrying out the fitting are provided in Appendix B5.

#### 4. Results and Discussion

Figures 8 show the temperature predictions obtained from P-Flash for two selected cases: 1) a fast growth fire with low peak HRR case and 2) a medium growth fire with high peak HRR case. There are three sets of curves in each figure: i) ground truth/Room 1 temperature, ii) prediction with learning from fitting (LFF), and iii) prediction without LFF. For each prediction curve, it can be composed of up to two lines: a) red line represents the Room 1 temperature predictions associated with Phase II and III and b) blue line is for predictions in Phase IV. Since no prediction is needed for Phase I, comparison is omitted.

In Fig. 8a, it can be seen that P-Flash provides accurate temperature predictions of Room 1 in all phases, and the benefit of using LFF is noticeable. After approximately 1150 s, when all detectors are lost, P-Flash is still capable to provide predictions with similar trend and magnitude. For P-Flash without LFF, the prediction relies on the regression models, and it can be shown that the temperature prediction is unrealistic (i.e., showing a temperature increase to as high as 910 °C). This observation demonstrates that unphysical inputs will lead to unphysical outputs. Also, it is worth noting that the discrepancy observed at around 250 s is probably due to the change of temperature increase in the available detector temperature. Physically, it is the pivot point of its 1<sup>st</sup> derivative where the rate of change of temperature changes from positive sign to negative sign. Additional effort is under way to reduce such fluctuation.

In Fig. 8b, it can be seen that the temperature of Room 1 being recovered from Phase II and III is still growing exponentially. In the current version, P-Flash does not have additional information to predict the temperature decays. However, it is capable to project the temperature increase in which the determination of flashover (i.e., temperature approaching 600 °C) in Room 1 can be made. As shown in the figure, the results obtained based on P-Flash without LFF are over-estimated.



**Fig. 8.** Comparison between ground truth and predictions obtained from P-Flash with and without LFF.

In order to evaluate the model performance over the 1000 different cases, the mean absolute error (MAE) is being determined, and it is defined as:

$$MAE = \sum_{i=1}^m \frac{1}{m} \sum_{j=1}^{n_i} \frac{1}{n_i} |p_{i,j} - y_{i,j}| \quad (6)$$

where  $p$  is the prediction and  $y$  is the ground truth. The variables  $m$  and  $n$  represent the number of simulation runs and the number of prediction points for each phase associated to each case, respectively. The number of simulation runs is 1000 in this study, and since the extrapolation of a 5<sup>th</sup> order polynomial increases dramatically, the number of prediction points for each case in Phase IV is determined based on the flashover temperature condition. That means the comparison is omitted if the ground truth is larger than 600°C. Table 4 shows the MAE associated with different phases. It should be noted that the above results are denoted as “current prediction” and this is due to the fact that the prediction at time  $t$  is based on information obtained in time  $t$ .

Given the prediction and the ground truth, an additional assessment can be carried out to examine the overall model accuracy in terms of flashover occurrence prediction. The flashover occurrence is true when the temperature is larger than 600°C. The overall accuracy is determined as the ratio of correct prediction within 20 s of the time of flashover to the total number of flashover occurrence in 1000 cases. Two example cases can be found in Figs. 8. In case 1, since the ground truth does not meet the potential flashover occurrence criteria (i.e., ~ 600°C), no flashover is observed. As compared to prediction from P-Flash, the recovered Room 1 temperature does not meet the potential flashover occurrence, false to flashover is also observed. For that, the prediction from P-Flash is determined to be correct. In case 2, a potential flashover occurrence is observed at about 1050 s. However, the recovered Room 1 temperature based on P-Flash does not reach 600°C at that time stamp. For that, P-Flash fails to predict the potential occurrence of flashover and this is a miss prediction. In order to discriminate the miss prediction, it is further divided into two categories: i) early prediction and ii) late prediction where the early prediction and the late prediction indicate that the Room 1 temperature recovered based on P-Flash reaches flashover occurrence criteria about more than 20 s prior to or more than 20 s after the flashover condition is met based on the ground true, respectively. Table 5 shows the overall model accuracy for the prediction of flashover occurrence to be approximately 83 %. The early prediction and the late prediction are shown to be 8 % and 9 %, respectively.

Since the current model is developed simple fire and vent opening conditions, Appendix C presents additional evaluation to reveal the current limitation of P-Flash and to provide guidelines about data requirement for the development of a more robust flashover prediction model in multi-compartment buildings.

**Table 4.** Performance summary for P-Flash.

	<b>Phase II</b>	<b>Phase III</b>	<b>Phase IV</b>
	<b>MAE (°C)</b>	<b>MAE (°C)</b>	<b>MAE (°C)</b>
<b>Current Prediction</b>	11	13	30
<b>Future Prediction</b>	13	16	37



**Table 5.** Overall model accuracy with early and late prediction for potential occurrence of flashover.

	Accuracy	Early	Late
	%	%	%
<b>Current Prediction</b>	83	8	9
<b>Future Prediction</b>	81	15	4

In actual firefighting, it is best if fire fighters can obtain the condition of the room of the fire origin ahead of time because they can optimize their rescue strategies and fire fighting tactics. For that, it is interesting to examine how well P-Flash can forecast temperature in advance (i.e., 150 s). In this scenario, the prediction at time  $t+150$  s is based on information obtained in time  $t$ . Since the temperature information for all compartments tends to have a monotonic increasing behavior, the temperature relationship at current time,  $t$ , can correlate well with flashover occurrence in future time,  $t+150$  s. As shown in Table 3 and Table 5, the MAE associated with this kind of scenario (denoted as future prediction) only increases slightly and the overall accuracy of P-Flash is relatively the same. These observations are expected as the temperature increase behaviors are well captured by the regression models and the fittings. However, the early prediction for the future prediction cases has noticeable increase and this is due to the fact that the temperature information being used generally has large temperature increase rate at time,  $t$ , as compared to time,  $t+150$  s.

## 5. Conclusions and Outlook

The development of P-Flash is presented. The realistic treatment of modeled sensor data that is not continuously available, but is subject to data loss due to thermal failure, though a challenge, was shown to be overcome successfully by the SVR modeling techniques and P-Flash is capable of recovering required detector temperatures for the determination of flashover conditions in the room of fire origin. P-Flash is under further development to handle more realistic conditions and these conditions include realistic multi-compartment building structures, fire located in any compartments, experimentally validated fire growth behavior of burning items, arbitrary vent opening conditions for windows and doors, and sensor limits. In order to facilitate data-driven fire fighting, collaborative works are required to develop smart fire protection systems and/or information transmission infrastructure. In the near future, P-Flash or a similar forecasting model could provide fire fighters with trustworthy and actionable information about fire scenes under the cognomen smart firefighting.

## Acknowledgments

The authors wish to thank Dr. Michael Huang for helpful discussion.

## References

- [1] Ahrens, M. (2017). Trends and patterns of US fire loss. National Fire Protection Association: Quincy, MA, USA.

- [2] Fahy, R. F., LeBlanc, P. R., & Molis, J. L. (2009). Firefighter Fatalities in the US-2008. National Fire Protection Association, Quincy, MA, USA.
- [3] Stowell, F. M. & Murnane, L. (2013). Essentials of Fire Fighting and Fire Department Operation. 6th Edition. International Fire Service Training Association.
- [4] Richards R., Munk B., Plumb O. (1997) Fire detection, location and heat release rate through inverse problem solution. Part I: theory. Fire Safety J 28(4):323–350.
- [5] Neviackas A., Trouvé A. (2007) Sensor-driven inverse zone modeling of enclosure fire dynamics. In: SFPE Professional Development Conference and Exposition. Las Vegas, NV.
- [6] Overholt, K. J., & Ezekoye, O. A. (2012). Characterizing heat release rates using an inverse fire modeling technique. Fire Technology, 48(4), 893-909.
- [7] Pomeroy, A. T. (2010). Analysis of the effects of temperature and velocity on the response time index of heat detectors. University of Maryland. (MS Dissertation).
- [8] Lichman, M., 2013. UCI machine learning repository.
- [9] Madrzykowski, D., & Weinschenk, C. (2019). Impact of fixed ventilation on fire damage patterns in full-scale structures. National Criminal Justice Reference Service, 252831.
- [10] Reneke, R., Peacock, R., Gilbert, S., and Cleary, T. CFAST - Consolidated Fire and Smoke Transport (Version 7) Volume 5: CFAST Fire Data Generator (CData). NIST Technical Note. (In review).
- [11] Peacock, R. D., Reneke, P. A. and Forney, G. P. (2017). CFAST–Consolidated Model of Fire Growth and Smoke Transport (Version 7): User’s Guide. NIST Technical Note 1889v2.
- [12] Peacock, R. D. CFAST–Consolidated Fire and Smoke Transport (Version 7) Volume 4: Configuration Management. NIST TN 1889v1.
- [13] McGrattan, K., Hostikka, S., McDermott, R., Floyd, J., Weinschenk, C., & Overholt, K. (2013). Fire dynamics simulator user’s guide. NIST special publication, 1019 (6).
- [14] Babrauskas, V. (2016). Heat release rates. In SFPE handbook of fire protection engineering (pp. 799-904). Springer, New York, NY.
- [15] Kim, H. J., & Lilley, D. G. (2002). Heat release rates of burning items in fires. Journal of propulsion and power, 18(4), 866-870.
- [16] NFPA 72 National Fire Alarm Code 2002 Edition. NFPA, Quincy, MA
- [17] Liu, H., & Motoda, H. (Eds.). (1998). Feature extraction, construction and selection: A data mining perspective (Vol. 453). Springer Science & Business Media.
- [18] Tam, W. C., Fu, E. Y., Mensch, A., Hamins, A., You, C., Ngai, G., & va Leong, H. (2020). Prevention of cooktop ignition using detection and multi-step machine learning algorithms. Fire Safety Journal, 103043.
- [19] Yang, C.C. & Shieh, M.D. (2010). A support vector regression based prediction model of affective responses for product form design. Computers & Industrial Engineering, 59(4), 682-689.
- [20] Vapnik, V. (1998). The support vector method of function estimation. In Nonlinear Modeling (pp. 55-85). Springer, Boston, MA.

## Appendix A: CData Input File

&HEAD VERSION = 7600, TITLE = 'FSJ\_3\_compartments' /

!! CData Sampling Namelist

&MHDR NUMBER\_OF\_CASES = 1000 WRITE\_SEEDS = .TRUE. /

&MRND ID = 'End of Growth Time Generator', DISTRIBUTION\_TYPE = 'UNIFORM' VALUE\_TYPE = 'REAL' MINIMUM = 75 MAXIMUM = 1400 /

&MRND ID = 'Peak HRR Generator', DISTRIBUTION\_TYPE = 'UNIFORM' VALUE\_TYPE = 'REAL' MINIMUM = 50000 MAXIMUM = 2200000 /

&MRND ID = 'Plateau End Time' DISTRIBUTION\_TYPE = 'CONSTANT' VALUE\_TYPE = 'REAL' REAL\_CONSTANT\_VALUE = 2000 /

&MRND ID = 'Fire End Time' DISTRIBUTION\_TYPE = 'CONSTANT' VALUE\_TYPE = 'REAL' REAL\_CONSTANT\_VALUE = 1500 /

&MRND ID = 'End of Fire HRR' DISTRIBUTION\_TYPE = 'CONSTANT' VALUE\_TYPE = 'REAL' REAL\_CONSTANT\_VALUE = 0 /

&MFIR ID = 'Fire\_generator' FIRE\_ID = 'Fire' FIRE\_TIME\_GENERATORS = 'End of Growth Time Generator'

'Plateau End Time' 'Fire End Time' FIRE\_HRR\_GENERATORS = 'Peak HRR Generator' 'Peak HRR Generator'

'End of Fire HRR' NUMBER\_OF\_GROWTH\_POINTS = 20. /

!! CFAST Namelist

!! Scenario Configuration

&TIME SIMULATION = 8400 PRINT = 20 SMOKEVIEW = 20 SPREADSHEET = 20 /

&INIT PRESSURE = 101325 RELATIVE\_HUMIDITY = 50 INTERIOR\_TEMPERATURE = 20 EXTERIOR\_TEMPERATURE = 20 /

!! Material Properties

&MATL ID = 'GYPSUM' MATERIAL = 'GYPSUM Sam',

CONDUCTIVITY = 0.276 DENSITY = 752 SPECIFIC HEAT = 1.01699993896484, THICKNESS = 0.0159 EMISSIVITY = 0.94 /

!! Compartments

&COMP ID = 'Comp 1'

DEPTH = 4.5 HEIGHT = 2.5 WIDTH = 4.5

CEILING\_MATL\_ID = 'GYPSUM' CEILING\_THICKNESS = 0.15 WALL\_MATL\_ID = 'GYPSUM' WALL\_THICKNESS = 0.15 FLOOR\_MATL\_ID = 'GYPSUM' FLOOR\_THICKNESS = 0.15

ORIGIN = 0, 0, 0 GRID = 50, 50, 50 LEAK AREA\_RATIO = 3.7777777777778E-06, 2.5679012345679E-06 /

&COMP ID = 'Comp 2'

DEPTH = 3.5 HEIGHT = 2.5 WIDTH = 1

CEILING\_MATL\_ID = 'GYPSUM' CEILING\_THICKNESS = 0.15 WALL\_MATL\_ID = 'GYPSUM' WALL\_THICKNESS = 0.15 FLOOR\_MATL\_ID = 'GYPSUM' FLOOR\_THICKNESS = 0.15

ORIGIN = 3.5, 4.5, 0 GRID = 50, 50, 50 LEAK AREA\_RATIO = 7.5555555555556E-06, 1.48571428571429E-05 /

&COMP ID = 'Comp 3'

DEPTH = 3.5 HEIGHT = 2.5 WIDTH = 3.5

CEILING\_MATL\_ID = 'GYPSUM' CEILING\_THICKNESS = 0.15 WALL\_MATL\_ID = 'GYPSUM' WALL\_THICKNESS = 0.15 FLOOR\_MATL\_ID = 'GYPSUM' FLOOR\_THICKNESS = 0.15

ORIGIN = 0, 4.5, 0 GRID = 50, 50, 50 LEAK AREA\_RATIO = 4.85714285714286E-06, 4.24489795918367E-06 /

!! Wall Vents

&VENT TYPE = 'WALL' ID = 'Window' COMP\_IDS = 'Comp 3' 'OUTSIDE', BOTTOM = 1.5 HEIGHT = 0.5, WIDTH = 0.3

FACE = 'LEFT' OFFSET = 1.65 /

&VENT TYPE = 'WALL' ID = 'WallVent2-3' COMP\_IDS = 'Comp 2', 'Comp 3', BOTTOM = 0 HEIGHT = 2, WIDTH = 0.75

CRITERION = 'TIME' T = 0, 1 F = 1, 1 FACE = 'LEFT' OFFSET = 2.5 /

&VENT TYPE = 'WALL' ID = 'WallVent1-2' COMP\_IDS = 'Comp 1', 'Comp 2', BOTTOM = 0 HEIGHT = 2, WIDTH = 0.75

CRITERION = 'TIME' T = 0, 1 F = 1, 1 FACE = 'REAR' OFFSET = 3.625 /

&VENT TYPE = 'WALL' ID = 'Door' COMP\_IDS = 'Comp 1' 'OUTSIDE', BOTTOM = 0 HEIGHT = 2, WIDTH = 0.75  
CRITERION = 'TIME' T = 0, 1 F = 1, 1 FACE = 'FRONT' OFFSET = 1.875 /

!! Fires

&FIRE ID = 'Fire' COMP\_ID = 'Comp 3', FIRE\_ID = 'Random\_Fire' LOCATION = 1.75, 1.75 /  
&CHEM ID = 'Random\_Fire' CARBON = 3 CHLORINE = 0 HYDROGEN = 7 NITROGEN = 1 OXYGEN = 2 HEAT\_OF\_COMBUSTION = 26000 RADIATIVE\_FRACTION = 0.35 /  
&TABL ID = 'Random\_Fire' LABELS = 'TIME', 'HRR', 'HEIGHT', 'AREA', 'CO\_YIELD', 'SOOT\_YIELD', 'HCN\_YIELD', 'HCL\_YIELD', 'TRACE\_YIELD' /  
&TABL ID = 'Random\_Fire', DATA = 0, 0, 0, 0.001, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 22.8551770939605, 21.9006517182919, 0, 0.000134431445407394, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 45.710354187921, 87.6026068731677, 0, 0.000407519937727851, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 68.5655312818815, 197.105865464627, 0, 0.000779641369607396, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 91.420708375842, 350.410427492671, 0, 0.00123536944159478, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 114.275885469803, 547.516292957298, 0, 0.00176543964574354, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 137.131062563763, 788.423461858509, 0, 0.00236343067970157, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 159.986239657724, 1073.1319341963, 0, 0.00302452885528663, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 182.841416751684, 1401.64170997068, 0, 0.00374493985677181, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 205.696593845645, 1773.95278918164, 0, 0.00452156609163308, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 228.551770939605, 2190.06517182919, 0, 0.00535181223645546, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 2228.5517709396, 2190.06517182919, 0, 0.00535181223645546, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 2378.5517709396, 1773.95278918164, 0, 0.00452156609163308, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 2528.5517709396, 1401.64170997068, 0, 0.00374493985677181, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 2678.5517709396, 1073.1319341963, 0, 0.00302452885528663, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 2828.5517709396, 788.423461858509, 0, 0.00236343067970157, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 2978.5517709396, 547.516292957298, 0, 0.00176543964574354, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 3128.5517709396, 350.410427492671, 0, 0.00123536944159478, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 3278.5517709396, 197.105865464627, 0, 0.000779641369607396, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 3428.5517709396, 87.6026068731676, 0, 0.000407519937727851, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 3578.5517709396, 21.9006517182919, 0, 0.000134431445407394, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 3728.5517709396, 0, 0, 0, 0.031, 0.13, 0, 0, 0 /  
&TABL ID = 'Random\_Fire', DATA = 3738.5517709396, 0, 0, 0.001, 0.031, 0.13, 0, 0, 0 /

!! Devices

&DEVC ID = 'HD1\_1\_1' COMP\_ID = 'Comp 3' LOCATION = 0.55, 1.75, 2.465 TYPE = 'HEAT\_DETECTOR' SETPOINT = 65, RTI = 35 /  
&DEVC ID = 'HD1\_1\_2' COMP\_ID = 'Comp 3' LOCATION = 0.55, 1.75, 2.365 TYPE = 'HEAT\_DETECTOR' SETPOINT = 65, RTI = 35 /  
&DEVC ID = 'HD1\_1\_3' COMP\_ID = 'Comp 3' LOCATION = 0.55, 1.75, 2.265 TYPE = 'HEAT\_DETECTOR' SETPOINT = 65, RTI = 35 /  
&DEVC ID = 'HD1\_1\_4' COMP\_ID = 'Comp 3' LOCATION = 0.55, 1.75, 2.165 TYPE = 'HEAT\_DETECTOR' SETPOINT = 65, RTI = 35 /  
&DEVC ID = 'HD1\_1\_5' COMP\_ID = 'Comp 3' LOCATION = 0.55, 1.75, 2.065 TYPE = 'HEAT\_DETECTOR' SETPOINT = 65, RTI = 35 /  
&DEVC ID = 'HD1\_1\_6' COMP\_ID = 'Comp 3' LOCATION = 0.55, 1.75, 1.965 TYPE = 'HEAT\_DETECTOR' SETPOINT = 65, RTI = 35 /  
&DEVC ID = 'HD1\_1\_7' COMP\_ID = 'Comp 3' LOCATION = 0.55, 1.75, 1.865 TYPE = 'HEAT\_DETECTOR' SETPOINT = 65, RTI = 35 /  
&DEVC ID = 'HD1\_2\_1' COMP\_ID = 'Comp 3' LOCATION = 1.15, 1.75, 2.465 TYPE = 'HEAT\_DETECTOR' SETPOINT = 65, RTI = 35 /  
&DEVC ID = 'HD1\_2\_2' COMP\_ID = 'Comp 3' LOCATION = 1.15, 1.75, 2.365 TYPE = 'HEAT\_DETECTOR' SETPOINT = 65, RTI = 35 /  
&DEVC ID = 'HD1\_2\_3' COMP\_ID = 'Comp 3' LOCATION = 1.15, 1.75, 2.265 TYPE = 'HEAT\_DETECTOR' SETPOINT = 65, RTI = 35 /  
&DEVC ID = 'HD1\_2\_4' COMP\_ID = 'Comp 3' LOCATION = 1.15, 1.75, 2.165 TYPE = 'HEAT\_DETECTOR' SETPOINT = 65, RTI = 35 /  
&DEVC ID = 'HD1\_2\_5' COMP\_ID = 'Comp 3' LOCATION = 1.15, 1.75, 2.065 TYPE = 'HEAT\_DETECTOR' SETPOINT = 65, RTI = 35 /  
&DEVC ID = 'HD1\_2\_6' COMP\_ID = 'Comp 3' LOCATION = 1.15, 1.75, 1.965 TYPE = 'HEAT\_DETECTOR' SETPOINT = 65, RTI = 35 /  
&DEVC ID = 'HD1\_2\_7' COMP\_ID = 'Comp 3' LOCATION = 1.15, 1.75, 1.865 TYPE = 'HEAT\_DETECTOR' SETPOINT = 65, RTI = 35 /  
&DEVC ID = 'HD1\_3\_1' COMP\_ID = 'Comp 3' LOCATION = 1.75, 1.75, 2.465 TYPE = 'HEAT\_DETECTOR' SETPOINT = 65, RTI = 35 /  
&DEVC ID = 'HD1\_3\_2' COMP\_ID = 'Comp 3' LOCATION = 1.75, 1.75, 2.365 TYPE = 'HEAT\_DETECTOR' SETPOINT = 65, RTI = 35 /

[illegible]

[Intentionally Left Blank]

## **Appendix B: P-Flash Source Codes**

Appendix B consists of 5 parts: B1 to B5. Appendix B1 provides the main codes for P-Flash. Appendix B2 presents codes that read the data and assign them to an appropriate format. Appendix B3 provides codes for data pre-processing and feature extraction. Appendix B4 provides codes for model training and testing. Appendix B5 presents codes for the memory component and fitting. The logic flow of the codes strictly follows the model descriptions provided in the main text.

To execute the codes, the reader only needs the original data which can be downloaded from <https://doi.org/10.18434/M32258> and combines the codes in a single file.

## Appendix B1: Main

```
# Load libraries
import numpy as np
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error
from scipy.optimize import curve_fit
# Load scripts
import CsvReader
import Utility

# Main codes
# Input files
DataFile = 'C:\\Users\\xxx\\IdeaProjects\\3compartment\\Inputs\\case1000.csv'
# Output locations
resultDir = 'C:\\Users\\xxx\\IdeaProjects\\3compartment\\Results\\'
# Cross-validation setting
CV = 5

# 2 major steps are done here: data pro-processing and feature extraction
# Look at Appendix B2 and B3
SP = SignalPreprocessing(DataFile, resultDir)
# Get only allInstances
allInstances = SP.allInstances

# Construct appropriate training and testing sets for "current" prediction or "future" prediction
learningTestingDict, TestInstancesGroup = Instance.Instances2GroupLists_multiRoomV3(Instances = allInstances, cv=CV, typeLabel= 'current') # or Future

# Start to carry out training and testing for the 5 different subsets (CV = 5)
# Another 2 major steps are done here: modeling development and fitting
# Look at Appendix B4 and B5
for i in range(0, CV):
    Xs_train_fast = learningTestingDict['fast_train'][0][i]
    y_train_fast = learningTestingDict['fast_train'][1][i]
    Xs_test_fast = learningTestingDict['fast_test'][0][i]
    y_test_fast = learningTestingDict['fast_test'][1][i]
```



```

Xs_train_slow = learningTestingDict['slow_train'][0][i]
y_train_slow = learningTestingDict['slow_train'][1][i]
Xs_test_slow = learningTestingDict['slow_test'][0][i]
y_test_slow = learningTestingDict['slow_test'][1][i]

TestInstances = TestInstancesGroup[i]

# Normalization
scaler_fast = StandardScaler()
Xs_train_fast = scaler_fast.fit_transform(Xs_train_fast)
Xs_test_fast = scaler_fast.transform(Xs_test_fast)
scaler_slow = StandardScaler()
Xs_train_slow = scaler_slow.fit_transform(Xs_train_slow)
Xs_test_slow = scaler_slow.transform(Xs_test_slow)

# Initialize model
FTR = FireTemperatureRegression()

# 2 SVR models: r1 and r2
# r1 exists in Phase 2
# r2 exists in Phase 2 and 3
FTR.fit(
    Xs_r1 = Xs_train_fast,
    y_r1 = y_train_fast,
    Xs_r2 = Xs_train_slow,
    y_r2 = y_train_slow
)

for i_test in range(0, len(TestInstances)):
    testI = TestInstances[i_test]
    if len(testI[2]) > 0:
        currentID = testI[-1]
        # Carrying out fitting
        predict_values, real_values, p2, r2, p3, r3, p23, r23, p4, r4, fit_type = FTR.predict_instancesV2(
            fvs_fast = testI[0],
            fvs_slow = testI[1],
            reals = testI[2],
            types = testI[3],
            ts = testI[5],

```

```

    scale_fast = scaler_fast,
    scale_slow = scaler_slow
)

# Determine overall MAE in Phase 2 to Phase 4
MAE, num = FTR.getMAE(predict_values, real_values)
# Determine MAE in Phase 2 only
try:
    MAE_p2, _ = FTR.getMAE(p2, r2)
except ValueError:
    MAE_p2 = 0
# Determine MAE in Phase 3 only
MAE_p3, _ = FTR.getMAE(p3, r3)
# Determine overall MAE in Phase 2 and 3
MAE_p23, _ = FTR.getMAE(p23, r23)
# Determine MAE in Phase 4 only
MAE_p4, _ = FTR.getMAE(p4, r4)
# End of Main codes

```

## Appendix B2: Read Data

# Step 1 to get data

```
class SignalPreprocessing(object):
```

```
    # Getting raw data
```

```
    def __init__(self, signal_fn, resultDir):
```

```
        readingModule = CsvReader.CsvReader(signal_fn)
```

```
        Data = readingModule.getData(
```

```
            indexs = [0,1,2,3,4,5],
```

```
            hasHeader = 1,
```

```
            needHandleNegativeOneIndex = [],
```

```
            flag = None
```

```
        )
```

```
        self.ids_list = Data[0]
```

```
        self.times_list = Data[1]
```

```
        self.t1s_list = Data[2]
```

```
        self.t2s_list = Data[3]
```

```
        self.t3s_list = Data[4]
```

```
        self.tps_list = Data[5]
```

```
        self._buildInstances()
```

```
def _buildInstances(self):
```

```
    self.allInstances = []
```

```
    temp_ids = []
```

```
    idxRanges = []
```

```
    lastID = self.ids_list[0]
```

```
    startIdx = 0
```

```
    for i in range(0, len(self.ids_list)):
```

```
        print(i)
```

```
        currentID = self.ids_list[i]
```

```
        if currentID != lastID:
```

```
            endIdx = i
```

```
            idxRanges.append([startIdx, endIdx])
```

```
            temp_ids.append(lastID)
```

```
            # Refresh
```

```
            lastID = currentID
```

```
            startIdx = i
```

```

# Append last one
idxRanges.append([startIdx, len(self.ids_list)])
temp_ids.append(lastID)
for i in range(0, len(idxRanges)):
    print(i)
    currentRng = idxRanges[i]
    newInstance = Instance(
        temp_ids[i],
        self.times_list[currentRng[0]: currentRng[1]],
        self.t1s_list[currentRng[0]: currentRng[1]],
        self.t2s_list[currentRng[0]: currentRng[1]],
        self.t3s_list[currentRng[0]: currentRng[1]],
        self.tps_list[currentRng[0]: currentRng[1]])
    self.allInstances.append(newInstance)
# End of step 1

```

## Appendix B3: Data Pre-Processing and Feature Extraction

# Step 2 to do data pre-processing and feature extraction

```
class Instance(object):
```

```
    @staticmethod
```

```
    def Instances2GroupLists_multiRoomV3(Instances, cv, typeLabel):
```

```
        Xs_train_group_fastRoom = []
```

```
        Xs_test_group_fastRoom = []
```

```
        y_train_group_fastRoom = []
```

```
        y_test_group_fastRoom = []
```

```
        Xs_train_group_slowRoom = []
```

```
        Xs_test_group_slowRoom = []
```

```
        y_train_group_slowRoom = []
```

```
        y_test_group_slowRoom = []
```

```
        TestInstances_group = []
```

```
totalLength = len(Instances)
```

```
step = totalLength / float(cv)
```

```
chunkIdxs = np.arange(0, totalLength, step).tolist()
```

```
for i in range(0, len(chunkIdxs)):
```

```
    startIdx = chunkIdxs[i]
```

```
    if i + 1 < len(chunkIdxs):
```

```
        endIdx = chunkIdxs[i + 1]
```

```
    else:
```

```
        endIdx = len(Instances)
```

```
    # Training and Testing subsets
```

```
    # Build Xs and Ys
```

```
    Xs_train_fast = []
```

```
    Xs_test_fast = []
```

```
    Xs_train_slow = []
```

```
    Xs_test_slow = []
```

```
    # -----
```

```
    y_train_fast = []
```

```
    y_test_fast = []
```

```
    y_train_slow = []
```

```
    y_test_slow = []
```

```
    # -----
```

```
    TestInstance = []
```

```
    # -----
```

```

for i_instance in range(0, len(Instances)):
    currentInstance = Instances[i_instance]
    if i_instance >= startIdx and i_instance < endIdx:
        Xs_test_slow.extend(currentInstance.instances_r1_fvs)
        Xs_test_fast.extend(currentInstance.instances_r2_fvs)
        if typeLabel == 'current':
            currentTestInstance = [
                currentInstance.test_fvs_fast_current,
                currentInstance.test_fvs_slow_current,
                currentInstance.test_Labels_current,
                currentInstance.test_types_current,
                currentInstance.times_refine[currentInstance.thresholdPoint_t1],
                currentInstance.test_t_current,
                currentInstance.id
            ]
            TestInstance.append(currentTestInstance)
        else:
            currentTestInstance = [
                currentInstance.test_fvs_fast_future,
                currentInstance.test_fvs_slow_future,
                currentInstance.test_Labels_future,
                currentInstance.test_types_future,
                currentInstance.times_refine[currentInstance.thresholdPoint_t1],
                currentInstance.test_t_future,
                currentInstance.id
            ]
            TestInstance.append(currentTestInstance)
    else:
        Xs_train_slow.extend(currentInstance.instances_r1_fvs)
        Xs_train_fast.extend(currentInstance.instances_r2_fvs)
        if typeLabel == 'current':
            y_train_slow.extend(currentInstance.regression_r1_lb0s)
            y_train_fast.extend(currentInstance.regression_r2_lb0s)
        else:
            y_train_slow.extend(currentInstance.regression_r1_lbins)
            y_train_fast.extend(currentInstance.regression_r2_lbins)

Xs_train_group_fastRoom.append(Xs_train_fast)
Xs_test_group_fastRoom.append(Xs_test_fast)

```

```

    y_train_group_fastRoom.append(y_train_fast)
    y_test_group_fastRoom.append(y_test_fast)
    Xs_train_group_slowRoom.append(Xs_train_slow)
    Xs_test_group_slowRoom.append(Xs_test_slow)
    y_train_group_slowRoom.append(y_train_slow)
    y_test_group_slowRoom.append(y_test_slow)
    TestInstances_group.append(TestInstance)
return {
    'fast_train': [Xs_train_group_fastRoom, y_train_group_fastRoom],
    'fast_test': [Xs_test_group_fastRoom, y_test_group_fastRoom],
    'slow_train': [Xs_train_group_slowRoom, y_train_group_slowRoom],
    'slow_test': [Xs_test_group_slowRoom, y_test_group_slowRoom]
}, TestInstances_group

# Carrying out data pre-processing
def __init__(self, id, times, t1s, t2s, t3s, ps, threshold=150, predictTempThreshold=600):
    # Read values
    self.id = id
    self.times = times
    self.t1s = t1s
    self.t2s = t2s
    self.t3s = t3s
    self.ps = ps
    self.threshold = threshold
    self.predictTempThreshold = predictTempThreshold
    # Process starts here
    self._preprocess()
    self._determine_labels()
    self._applyTemperatureThreshold()
    self._extract_features_flash()
    self._extractRegressionFeaturesAndLabels(3, 6)
    self._buildPhaseTimeRanges()
    self._buildTestingInstance(3, 6)
    print('d')

def _preprocess(self):
    flag = (self.ps[0] == self.ps[1])
    while flag:
        self.times = self.times[1:]

```

```

self.t1s = self.t1s[1:]
self.t2s = self.t2s[1:]
self.t3s = self.t3s[1:]
self.ps = self.ps[1:]
flag = (self.ps[0] == self.ps[1])
# Get front data value (for refine use)
self.front_p = float(self.ps[0])
self.front_t1 = float(self.t1s[0])
self.front_t2 = float(self.t2s[0])
self.front_t3 = float(self.t3s[0])
# Convert to float
self.times = [float(ele) - float(self.times[0]) for ele in self.times]
self.t1s = [float(ele) - float(self.t1s[0]) for ele in self.t1s]
self.t2s = [float(ele) - float(self.t2s[0]) for ele in self.t2s]
self.t3s = [float(ele) - float(self.t3s[0]) for ele in self.t3s]
self.ps = [float(ele) - float(self.ps[0]) for ele in self.ps]
# Recover original signal
self.ps_ori = [e+self.front_p for e in self.ps]
self.t1s_ori = [e+self.front_t1 for e in self.t1s]
self.t2s_ori = [e+self.front_t2 for e in self.t2s]
self.t3s_ori = [e+self.front_t3 for e in self.t3s]
# Cut off right hand part
flag = (self.ps[-1] < self.ps[-2])
while flag:
    self.times = self.times[:-1]
    self.t1s = self.t1s[:-1]
    self.t2s = self.t2s[:-1]
    self.t3s = self.t3s[:-1]
    self.ps = self.ps[:-1]
    flag = (self.ps[-1] < self.ps[-2])
# Generate other basic features
self.times_diff = self.times[1:]
self.diff_ps = np.diff(np.array(self.ps)).tolist()
self.diff_t1s = np.diff(np.array(self.t1s)).tolist()
self.diff_t2s = np.diff(np.array(self.t2s)).tolist()
self.diff_t3s = np.diff(np.array(self.t3s)).tolist()
# Smoothing
self.diff_ps = Utility.move_average(self.diff_ps, 4)
self.diff_t1s = Utility.move_average(self.diff_t1s, 4)

```



```

self.diff_t2s = Utility.move_average(self.diff_t2s, 4)
self.diff_t3s = Utility.move_average(self.diff_t3s, 4)
# Padding
self.times_gap = []
self.t1s_gap = []
self.t2s_gap = []
self.t3s_gap = []
for i in range(0, len(self.times)):
    self.times_gap.append(self.times[i])
    self.t1s_gap.append(self.ps_ori[i] - self.t1s_ori[i])
    self.t2s_gap.append(self.ps_ori[i] - self.t2s_ori[i])
    self.t3s_gap.append(self.ps_ori[i] - self.t3s_ori[i])
print('d')

# Assign labels
def _determine_labels(self):
    if max(self.ps) >= self.predictTempThreshold:
        self.isFlash = True
    else:
        self.isFlash = False

# Apply temperature threshold
def _applyTemperatureThreshold(self):
    self.times_refine = []
    self.ps_refine = []
    self.t1s_refine = []
    self.t2s_refine = []
    self.t3s_refine = []
    self.thresholdPoint_p = None
    self.thresholdPoint_t1 = None
    self.thresholdPoint_t2 = None
    self.thresholdPoint_t3 = None

    for i in range(0, len(self.times)):
        self.times_refine.append(self.times[i])
        # Apply threshold
        current_p = self.ps[i] + self.front_p
        current_t1 = self.t1s[i] + self.front_t1
        current_t2 = self.t2s[i] + self.front_t2

```

```

current_t3 = self.t3s[i] + self.front_t3
if current_p < self.threshold:
    self.ps_refine.append(current_p)
else:
    self.ps_refine.append(-1)
    if self.thresholdPoint_p == None:
        self.thresholdPoint_p = i
if current_t1 < self.threshold:
    self.t1s_refine.append(current_t1)
else:
    self.t1s_refine.append(-1)
    if self.thresholdPoint_t1 == None:
        self.thresholdPoint_t1 = i
if current_t2 < self.threshold:
    self.t2s_refine.append(current_t2)
else:
    self.t2s_refine.append(-1)
    if self.thresholdPoint_t2 == None:
        self.thresholdPoint_t2 = i
if current_t3 < self.threshold:
    self.t3s_refine.append(current_t3)
else:
    self.t3s_refine.append(-1)
    if self.thresholdPoint_t3 == None:
        self.thresholdPoint_t3 = i

```

# Construct different phases

```

def _buildPhaseTimeRanges(self):
    if self.thresholdPoint_p == None:
        self.thresholdPoint_p = len(self.times_refine)-1
    if self.thresholdPoint_t1 == None:
        self.thresholdPoint_t1 = len(self.times_refine)-1
    if self.thresholdPoint_t2 == None:
        self.thresholdPoint_t2 = len(self.times_refine)-1
    self.phase1Range = [self.times_refine[self.thresholdPoint_p], self.times_refine[self.thresholdPoint_t2]]
    self.phase2Range = [self.times_refine[self.thresholdPoint_t2], self.times_refine[self.thresholdPoint_t1]]
    self.phase3Range = [self.times_refine[self.thresholdPoint_t1], 999999] # just make sure it is large enough

```

# Subroutine of \_extract\_features\_flash

```

@staticmethod
def _FEATURE_single_TS(times, temps, startIndex, endIndex):
    # compute delta time
    deltaT = times[endIndex] - times[startIndex]
    # f1
    deltaTemp = temps[endIndex] - temps[startIndex]
    # f2
    avgDeltaTempRate = float(deltaTemp) / float(deltaT)
    # compute instant change
    instantValueChanges = []
    for i in range(startIndex+1, endIndex+1):
        dt_instant = times[i] - times[i-1]
        dv_instant = temps[i] - temps[i-1]
        speed_instant = float(dv_instant) / float(dt_instant)
        instantValueChanges.append(speed_instant)
    # f3 min instant speed
    min_instantValue = min(instantValueChanges)
    # f4 max instant speed
    max_instantValue = max(instantValueChanges)
    # f5 avg instant speed
    avg_instantValue = np.array(instantValueChanges).mean()
    # f6 largest index position
    maxIndex = instantValueChanges.index(max(instantValueChanges))
    position_ratio = float(maxIndex) / float(len(instantValueChanges))
    # f7 trend
    # build points
    X = np.arange(len(instantValueChanges))
    Y = np.array(instantValueChanges)
    A = np.vstack([X, np.ones(len(X))]).T
    m_all, _ = np.linalg.lstsq(A, Y)[0]
    X = np.arange(len(instantValueChanges))
    Y = np.array(instantValueChanges)
    A = np.vstack([X, np.ones(len(X))]).T
    m_all, _ = np.linalg.lstsq(A, Y)[0]
    # first part and second part
    middleIndex = int(len(instantValueChanges)/2)+1
    # first part
    X = np.arange(len(instantValueChanges[0:middleIndex]))
    Y = np.array(instantValueChanges[0:middleIndex])

```

```

A = np.vstack([X, np.ones(len(X))]).T
m_front, _ = np.linalg.lstsq(A, Y)[0]
# back part
X = np.arange(len(instantValueChanges[middleIndex:]))
if len(X.tolist()) == 0:
    m_back = m_front
else:
    Y = np.array(instantValueChanges[middleIndex:])
    A = np.vstack([X, np.ones(len(X))]).T
    m_back, _ = np.linalg.lstsq(A, Y)[0]
# fv
fv = [deltaT, deltaTemp, avgDeltaTempRate, min_instantValue, max_instantValue, avg_instantValue, position_ratio, m_all, m_front, m_back]
return fv

# Obtain features for Room 1 in Phase 1
def _extract_features_flash(self):
    if self.thresholdPoint_p != None:
        self.fv_full = []
        fv_p_temp = Instance._FEATURE_single_TS(times=self.times_refine, temps=self.ps_refine, startIndex=0, endIndex=self.thresholdPoint_p-1)
        fv_p_diff = Instance._FEATURE_single_TS(times=self.times_diff, temps=self.diff_ps, startIndex=0, endIndex=self.thresholdPoint_p-1)
        self.fv_full.extend(fv_p_temp)
        self.fv_full.extend(fv_p_diff)
    else:
        self.fv_full = None

# Obtain features for 1) Corridor in Phase 1 to 2 and 2) Room 2 in Phase 1 to 3
def _extractRegressionFeaturesAndLabels(self, n_space, windowLength = 6):
    self.isRegression_valid = False
    self.r1_window_end_ts = []
    self.instances_r1_fvs = []
    self.regression_r1_lb0s = []
    self.regression_r1_lbns = []
    self.r2_window_end_ts = []
    self.instances_r2_fvs = []
    self.regression_r2_lb0s = []
    self.regression_r2_lbns = []
    # check
    self.isRegression_valid = True

```

```

if self.thresholdPoint_p == None:
    self.isRegression_valid = False
    self.fv_fix_slow = None
    self.fv_fix_fast = None
else:
    if self.thresholdPoint_t1 == None:
        self.thresholdPoint_t1 = len(self.times_refine)-1
    if self.thresholdPoint_t2 == None:
        self.thresholdPoint_t2 = len(self.times_refine)-1
    # Get basic
    idxRange_fixing = [0, self.thresholdPoint_p-1]
    idxRange_changing_r1 = [self.thresholdPoint_p, self.thresholdPoint_t1-1]
    idxRange_changing_r2 = [self.thresholdPoint_p, self.thresholdPoint_t2-1]
    # Phase 1
    fv_fix_r1, self.mean_r1_value, self.mean_r1_change = Instance._FEATURE_regression(self.times_refine, self.ps_refine, self.t1s, idxRange_fixing[0],
idxRange_fixing[1])
    fv_fix_r2, self.mean_r2_value, self.mean_r2_change = Instance._FEATURE_regression(self.times_refine, self.ps_refine, self.t2s, idxRange_fixing[0],
idxRange_fixing[1])
    self.fv_fix_slow = fv_fix_r1
    self.fv_fix_fast = fv_fix_r2
    # Phase 2 and 3
    # r1 first ---
    # Apply moving window
    r1_mv_idxRanges = Instance._movingWindow(
        frontIndex=0,
        endIndex=idxRange_changing_r1[1],
        lengthWindow=windowLength,
        stepIndex=1
    )
    for i in range(0, len(r1_mv_idxRanges)):
        currentRange = r1_mv_idxRanges[i]
        changing_fv = Instance._FEATURE_regression(self.times_refine, self.ps_refine, self.t1s_refine, currentRange[0], currentRange[1], isFix=False,
deltaT=self.times_refine[currentRange[-1]]-self.times_refine[self.thresholdPoint_p],mean_t_value=self.mean_r1_value, mean_t_change=self.mean_r1_change)
        fv_full = []
        fv_full.extend(changing_fv)
        fv_full.extend(fv_fix_r1)
        # Get label
        currentLabel0 = self.ps_ori[currentRange[-1]]
        currentLabeln = self.ps_ori[min(len(self.ps_ori)-1, currentRange[-1]+n_space)]

```

```

        # Append
        self.r1_window_end_ts.append(self.times_refine[currentRange[-1]])
        self.instances_r1_fvs.append(fv_full)
        self.regression_r1_lb0s.append(currentLabel0)
        self.regression_r1_lbns.append(currentLabeln)
    # Then r2 ---
    # Apply moving window
    r2_mv_idxRanges = Instance._movingWindow(
        frontIndex=0,
        endIndex=idxRange_changing_r2[1],
        lengthWindow=windowLength,
        stepIndex=1
    )
    for i in range(0, len(r2_mv_idxRanges)):
        currentRange = r2_mv_idxRanges[i]
        changing_fv = Instance._FEATURE_regression(self.times_refine, self.ps_refine, self.ts_refine, currentRange[0], currentRange[1], isFix=False,
        deltaT=self.times_refine[currentRange[-1]]-self.times_refine[self.thresholdPoint_p], mean_t_value=self.mean_r1_value, mean_t_change=self.mean_r1_change)
        fv_full = []
        fv_full.extend(changing_fv)
        fv_full.extend(fv_fix_r2)
        # Get label
        currentLabel0 = self.ps_ori[currentRange[-1]]
        currentLabeln = self.ps_ori[min(len(self.ps_ori) - 1, currentRange[-1] + n_space)]
        # Append
        self.r2_window_end_ts.append(self.times_refine[currentRange[-1]])
        self.instances_r2_fvs.append(fv_full)
        self.regression_r2_lb0s.append(currentLabel0)
        self.regression_r2_lbns.append(currentLabeln)

# Subroutine of _extractRegressionFeaturesAndLabels
@staticmethod
def _FEATURE_regression(times, ps, ts, startIndex, endIndex, isFix = True, deltaT = None, mean_t_value = None, mean_t_change=None):
    # ps is the fire room
    # ts is other room
    mean_ps_v = np.array(ps[startIndex:endIndex+1]).mean()
    mean_ts_v = np.array(ts[startIndex:endIndex+1]).mean()
    instantValueChanges_ps = []
    for i in range(startIndex + 1, endIndex + 1):
        dt_instant = times[i] - times[i - 1]

```

```

        dv_instant = ps[i] - ps[i - 1]
        speed_instant = float(dv_instant) / float(dt_instant)
        instantValueChanges_ps.append(speed_instant)
    instantValueChanges_ts = []
    for i in range(startIndex + 1, endIndex + 1):
        dt_instant = times[i] - times[i - 1]
        dv_instant = ts[i] - ts[i - 1]
        speed_instant = float(dv_instant) / float(dt_instant)
        instantValueChanges_ts.append(speed_instant)
    meanPs = np.array(instantValueChanges_ps).mean()
    maxPs = np.array(instantValueChanges_ps).max()
    minPs = np.array(instantValueChanges_ps).min()
    meanTs = np.array(instantValueChanges_ts).mean()
    maxTs = np.array(instantValueChanges_ts).max()
    minTs = np.array(instantValueChanges_ts).min()
    # Compute ratio
    maxIndax = instantValueChanges_ps.index(max(instantValueChanges_ps))
    position_ratio_p = float(maxIndax) / float(len(instantValueChanges_ps))
    maxIndax = instantValueChanges_ts.index(max(instantValueChanges_ts))
    position_ratio_t = float(maxIndax) / float(len(instantValueChanges_ts))
    if isFix == False:
        fv = [mean_ts_v, meanTs, maxTs, minTs, position_ratio_t, deltaT]
        return fv
    else:
        fv = [mean_ps_v, mean_ts_v, meanPs, maxPs, minPs, meanTs, maxTs, minTs, position_ratio_p, position_ratio_t]
        mean_ts_value = mean_ts_v
        mean_ts_change = meanTs
        return fv, mean_ts_value, mean_ts_change

# Subroutine of _extractRegressionFeaturesAndLabels
@staticmethod
def _movingWindow(frontIndex, endIndex, lengthWindow, stepIndex):
    eachWindowIndexRanges = []
    # paras
    totalLength = endIndex - frontIndex + 1
    i = frontIndex
    while i < endIndex - lengthWindow:
        # iterate window
        currentWindowStartIdx = i

```

```

        currentWindowEndIdx = i + lengthWindow
        eachWindowIndexRanges.append([currentWindowStartIdx, currentWindowEndIdx])
        # refresh
        i = i+stepIndex
    return eachWindowIndexRanges

# Subroutine of _extractRegressionFeaturesAndLabels
@staticmethod
def _isInIndexRange(idx, rng):
    # if idx >= rng[0] and idx<=rng[1]:
    #     return True
    if idx<=rng[1]:
        return True
    else:
        return False

# Combine instances
def _buildTestingInstance(self, n_space, lengthWindow):
    if self.id == '3':
        print('d')
    if self.isRegression_valid == False:
        self.test_t_current = []
        self.test_Labels_current = []
        self.test_t_future = []
        self.test_Labels_future = []
        self.test_types_future = [] # 'multi' or 'single'
        self.test_types_current = []
        self.test_fvs_slow_future = [] # 1
        self.test_fvs_fast_future = [] # 2
        self.test_fvs_slow_current = [] # 1
        self.test_fvs_fast_current = [] # 2
        # logInfo ---
        self.test_logInfo_historyTemp_slow = []
        self.test_logInfo_windowTemp_slow = []
        self.test_logInfo_deltaTs = []
    else:
        idxRange_changing_slow = [self.thresholdPoint_p, self.thresholdPoint_t1 - 1]
        idxRange_changing_fast = [self.thresholdPoint_p, self.thresholdPoint_t2 - 1]
        # Build test fvs and types (one room or two room) and labels

```



```

self.test_t_current = []
self.test_Labels_current = []
self.test_t_future = []
self.test_Labels_future = []
self.test_types_future = []
self.test_types_current = []
self.test_fvs_slow_future = [] # 1
self.test_fvs_fast_future = [] # 2
self.test_fvs_slow_current = [] # 1
self.test_fvs_fast_current = [] # 2
# logInfo ---
self.test_logInfo_historyTemp_slow = []
self.test_logInfo_windowTemp_slow = []
self.test_logInfo_deltaTs = []
# Build -- future
for i in range(self.thresholdPoint_p, self.thresholdPoint_t1):
    # Determine window
    currentPredictIndex = i
    currentWindowEndIndex = max(0, currentPredictIndex-n_space)
    currentWindowStartIndex = max(0, currentWindowEndIndex - lengthWindow +1)
    if currentWindowEndIndex - currentWindowStartIndex+1 <= (1/2)* lengthWindow:
        continue
    self.test_Labels_future.append(self.ps_ori[currentPredictIndex])
    self.test_t_future.append(self.times_refine[currentPredictIndex])
    # Determine type
    flag_slow = Instance._isInIndexRange(currentWindowEndIndex, idxRange_changing_slow)
    flag_fast = Instance._isInIndexRange(currentWindowEndIndex, idxRange_changing_fast)
    if flag_slow == True and flag_fast == True:
        self.test_types_future.append('multi')
    else:
        # It should not go here
        self.test_types_future.append('single')
    # ----- test fv -----
    if flag_slow == True:
        # Build fv
        changing_fv = Instance._FEATURE_regression(self.times_refine, self.ps_refine, self.t1s_refine,
                                                    currentWindowStartIndex, currentWindowEndIndex, isFix=False,
                                                    deltaT=self.times_refine[currentWindowEndIndex] -self.times_refine[self.thresholdPoint_p],
mean_t_value=self.mean_r1_value, mean_t_change=self.mean_r1_change)

```

```

fv_full = []
fv_full.extend(changing_fv)
fv_full.extend(self.fv_fix_slow)
# Append -- test fv
self.test_fvs_slow_future.append(fv_full)
# Append -- history temp for slow room
self.test_logInfo_historyTemp_slow.append(
    self.t1s_refine[0:currentWindowEndIndex+1]
)
self.test_logInfo_windowTemp_slow.append(
    self.t1s_refine[currentWindowStartIndex:currentWindowEndIndex+1]
)
self.test_logInfo_deltaTs.append(
    self.times_refine[currentWindowEndIndex+1] - self.times_refine[self.thresholdPoint_p]
)
else:
    self.test_fvs_slow_future.append(None)
    self.test_logInfo_historyTemp_slow.append(None)
    self.test_logInfo_deltaTs.append(None)
if flag_fast == True:
    # Build fv
    changing_fv = Instance._FEATURE_regression(self.times_refine, self.ps_refine, self.t2s_refine,
                                              currentWindowStartIndex, currentWindowEndIndex, isFix=False,
                                              deltaT=self.times_refine[currentWindowEndIndex] - self.times_refine[self.thresholdPoint_p],
mean_t_value=self.mean_r2_value, mean_t_change=self.mean_r2_change)
    fv_full = []
    fv_full.extend(changing_fv)
    fv_full.extend(self.fv_fix_fast)
    # Append
    self.test_fvs_fast_future.append(fv_full)
else:
    self.test_fvs_fast_future.append(None)

for i in range(self.thresholdPoint_t1, len(self.times_refine)):
    self.test_t_future.append(self.times_refine[i])
    self.test_Labels_future.append(self.ps_ori[i])
    self.test_types_future.append('trend')
# Build -- current
for i in range(self.thresholdPoint_p, self.thresholdPoint_t1):

```

```

# Determine window
currentPredictIndex = i
currentWindowEndIndex = max(currentPredictIndex, 0)
currentWindowStartIndex = max(0, currentWindowEndIndex - lengthWindow + 1)
if currentWindowEndIndex - currentWindowStartIndex + 1 <= (1 / 2) * lengthWindow:
    continue
self.test_Labels_current.append(self.ps_ori[currentPredictIndex])
self.test_t_current.append(self.times_refine[currentPredictIndex])
# Determine type
flag_slow = Instance._isInIndexRange(currentWindowEndIndex, idxRange_changing_slow)
flag_fast = Instance._isInIndexRange(currentWindowEndIndex, idxRange_changing_fast)
if flag_slow == True and flag_fast == True:
    self.test_types_current.append('multi')
else:
    # It should not go here
    self.test_types_current.append('single')
# ----- test fv -----
if flag_slow == True:
    # Build fv
    changing_fv = Instance._FEATURE_regression(self.times_refine, self.ps_refine, self.t1s_refine,
                                              currentWindowStartIndex, currentWindowEndIndex, isFix=False,
                                              deltaT=self.times_refine[currentWindowEndIndex] - self.times_refine[self.thresholdPoint_p],
mean_t_value=self.mean_r1_value, mean_t_change=self.mean_r1_change)
    f = Utility.list_contain_nan(changing_fv)
    if f == True:
        print('a')

    fv_full = []
    fv_full.extend(changing_fv)
    fv_full.extend(self.fv_fix_slow)
    # Append
    self.test_fvs_slow_current.append(fv_full)
else:
    self.test_fvs_slow_current.append(None)
if flag_fast == True:
    # Build fv
    changing_fv = Instance._FEATURE_regression(self.times_refine, self.ps_refine, self.t2s_refine,
                                              currentWindowStartIndex, currentWindowEndIndex, isFix=False,
                                              deltaT=self.times_refine[currentWindowEndIndex] - self.times_refine[self.thresholdPoint_p],

```

```

mean_t_value=self.mean_r2_value, mean_t_change=self.mean_r2_change)
    fv_full = []
    fv_full.extend(changing_fv)
    fv_full.extend(self.fv_fix_fast)
    # Append
    self.test_fvs_fast_current.append(fv_full)
else:
    self.test_fvs_fast_current.append(None)
for i in range(self.thresholdPoint_t1, len(self.times_refine)):
    self.test_t_current.append(self.times_refine[i])
    self.test_Labels_current.append(self.ps_ori[i])
    self.test_types_current.append('trend')

    print('d')
# End of step 2

```

## Appendix B4: Training and Testing

```
# Step 3 to do training and testing
class FireTemperatureRegression(object):
    # Initialize parameters
    def __init__(self):
        self.regressionModule_room1 = None
        self.regressionModule_room2 = None
        # Time range
        self.Phase1TimeRange = None
        self.Phase2TimeRange = None
        self.Phase3TimeRange = None

    # Training and Testing the model
    def fit(self, Xs_r1, y_r1, Xs_r2, y_r2, isGrid = True):
        # r1
        gsc = GridSearchCV(
            estimator=SVR(kernel='rbf'),
            param_grid={
                'C': [0.1, 1, 100, 1000],
                'epsilon': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10],
                'gamma': [0.0001, 0.001, 0.005, 0.1, 1, 3, 5]
            },
            cv=5, scoring='neg_mean_squared_error', verbose=0, n_jobs=-1)
        grid_result = gsc.fit(Xs_r1, y_r1)
        best_params = grid_result.best_params_
        print('Room 1---C: ' + str(best_params["C"]))
        print('Room 1---epsilon: ' + str(best_params["epsilon"]))
        print('Room 1---gamma: ' + str(best_params["gamma"]))
        self.regressionModule_room1 = SVR(kernel='rbf', C=best_params["C"], epsilon=best_params["epsilon"], gamma=best_params["gamma"],
            coef0=0.1, shrinking=True,
            tol=0.001, cache_size=200, verbose=False, max_iter=-1)
        self.regressionModule_room1.fit(Xs_r1, y_r1)
        # r2
        gsc = GridSearchCV(
            estimator=SVR(kernel='rbf'),
            param_grid={
                'C': [0.1, 1, 100, 1000],
                'epsilon': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10],
```

```

        'gamma': [0.0001, 0.001, 0.005, 0.1, 1, 3, 5]
    },
    cv=5, scoring='neg_mean_squared_error', verbose=0, n_jobs=-1)
grid_result = gsc.fit(Xs_r2, y_r2)
best_params = grid_result.best_params_
print('Room 2---C: ' + str(best_params["C"]))
print('Room 2---epsilon: ' + str(best_params["epsilon"]))
print('Room 2---gamma: ' + str(best_params["gamma"]))
self.regressionModule_room2 = SVR(kernel='rbf', C=best_params["C"], epsilon=best_params["epsilon"],
                                gamma=best_params["gamma"],
                                coef0=0.1, shrinking=True,
                                tol=0.001, cache_size=200, verbose=False, max_iter=-1)
self.regressionModule_room2.fit(Xs_r2, y_r2)

# Padding and smoothing
@staticmethod
def MA(y, N=4):
    y_padded = np.pad(y, (N // 2, N - 1 - N // 2), mode='edge')
    y_smooth = np.convolve(y_padded, np.ones((N,)) / N, mode='valid')
    return y_smooth

# Subroutine of predict_instancesV2
# To get fitting type (binding or polynomial)
@staticmethod
def predict_instance_trend(memory_ts, memory_temps, predict_ts):
    FTP_trend = FireTemperaturePredictionForSameRoom()
    FTP_trend.fit(memory_ts, FireTemperatureRegression.MA(memory_temps))
    predict_temps = FTP_trend.predict(predict_ts)
    return predict_temps, FTP_trend.type

# Fitting
def predict_instancesV2(self, fvs_fast, fvs_slow, reals, types, ts, scale_fast, scale_slow):
    p2 = []
    r2 = []
    p3 = []
    r3 = []
    p23 = []
    r23 = []
    p4 = []

```

```

r4 = []

predict_values = []
i_trend = types.index('trend')
for i in range(0, i_trend):
    currentType = types[i]
    fv_fast = fvs_fast[i]
    fv_slow = fvs_slow[i]
    if fv_fast != None:
        fv_fast = scale_fast.transform([fv_fast])[0]
    if fv_slow != None:
        fv_slow = scale_slow.transform([fv_slow])[0]
    if currentType == 'multi':
        pred = self.predict_instanceV2(
            fv_fast = fv_fast,
            fv_slow = fv_slow,
            type = 1
        )
        p2.append(pred)
        r2.append(reals[i])
        p23.append(pred)
        r23.append(reals[i])
    elif currentType == 'single':
        pred = self.predict_instanceV2(
            fv_fast=fv_fast,
            fv_slow=fv_slow,
            type=2
        )
        p3.append(pred)
        r3.append(reals[i])
        p23.append(pred)
        r23.append(reals[i])
    predict_values.append(pred)
predict_ts = ts[i_trend:]
predict_temps, fit_type = FireTemperatureRegression.predict_instance_trend(ts[i_trend:], predict_values, predict_ts)
predict_values.extend(predict_temps)
p4 = predict_temps
r4 = reals[i_trend:]
return predict_values, reals, p2, r2, p3, r3, p23, r23, p4, r4, fit_type

```

```

# To produce final results for Phase 2 and 3
def predict_instanceV2(self, fv_fast, fv_slow, type):
    if type == 1:
        temperature_r1 = self.regressionModule_room1.predict([fv_fast])[0]
        temperature_r2 = self.regressionModule_room2.predict([fv_slow])[0]
        return (temperature_r1 + temperature_r2) / 2.0
    elif type == 2:
        temperature_r2 = self.regressionModule_room2.predict([fv_slow])[0]
        return temperature_r2

# Calculate MAE
@staticmethod
def getMAE(preds, reals):
    mae = mean_absolute_error(preds, reals)
    return mae, len(preds)
# End of step 3

```



## Appendix B5: Fitting

```
# Step 4 to do fitting
class FireTemperaturePredictionForSameRoom(object):
    # Initialize the inner parameter
    def __init__(self):
        self.timeToFit = None
        self.temperatureToFit = None

    # Carrying fitting
    def fit(self, times, temperatures):
        self.timeToFit = times
        self.temperatureToFit = temperatures
        self.type = FireTemperaturePredictionForSameRoom._determine_fit_type(self.temperatureToFit)
        # Type = 0, binding curve
        if self.type == 0:
            sigma = np.ones(len(self.timeToFit))
            sigma[[-2,-1]] = 0.001

            self.param = curve_fit(FireTemperaturePredictionForSameRoom._binding, self.timeToFit, self.temperatureToFit,
                                   bounds=((0,0,0), (500,10000,500)), sigma=sigma, maxfev=np.inf)
        # Type = 1, 3rd order polynomial
        elif self.type == 1:
            diff = np.diff(np.array(self.temperatureToFit)).tolist()
            idx = diff.index(max(diff))

            sigma = np.ones(len(self.timeToFit))
            sigma[[0, idx-1]] = 0.001
            self.param = curve_fit(FireTemperaturePredictionForSameRoom._poly3, self.timeToFit, self.temperatureToFit,
                                   bounds=((0,-1,-1,-1000), (1,1,1,8000)),sigma=sigma, maxfev=np.inf)
        # 5th order polynomial
        else:
            diff = np.diff(np.array(self.temperatureToFit)).tolist()
            idx = diff.index(max(diff))
            sigma = np.ones(len(self.timeToFit))
            sigma[[0, idx-1]] = 0.001
            self.param = curve_fit(
                FireTemperaturePredictionForSameRoom._poly5, self.timeToFit, self.temperatureToFit,
                bounds=((0,-1,-1,-1,-1,-1), (1,1,1,1,1,1)),maxfev=np.inf)
```

```

# Gather values
def predict(self, ts_predict):
    if self.type == 0:
        values_a = FireTemperaturePredictionForSameRoom._binding(np.array(ts_predict), *self.param[0])
    elif self.type == 1:
        values_a = FireTemperaturePredictionForSameRoom._poly3(np.array(ts_predict), *self.param[0])
    else:
        values_a = FireTemperaturePredictionForSameRoom._poly5(np.array(ts_predict), *self.param[0])
    values = values_a.tolist()
    return values

# Fitting options
@staticmethod
def _poly5(x,a,b,c,d,e,f):
    return a*x**5 + b*x**4 +c*x**3 +d*x**2 +e*x +f
@staticmethod
def _binding(x, kd, bmax, e):
    return ((bmax * x**.5) / (x**.5 + kd))+e
@staticmethod
def _log(x, a, b):
    return a*(np.log(x) / np.log(5))+ b
@staticmethod
def _poly3(x,a,b,c,d):
    return a*x**3+b*x**2+c*x+d

# To determine fitting type
@staticmethod
def _determine_fit_type(values):
    # Type 0 => binding || 1 => poly
    diffs = np.diff(np.array(values)).tolist()
    diffs = FireTemperatureRegression.MA(diffs).tolist()
    diffs = FireTemperatureRegression.MA(diffs).tolist()
    maxV = max(diffs)
    maxIdx = diffs.index(maxV)
    maxIdxRatio = float(maxIdx) / float(len(diffs))
    if maxIdxRatio < 0.65:
        return 0
    else:

```

```
    return 1  
# End of step 4
```

[Intentionally Left Blank]

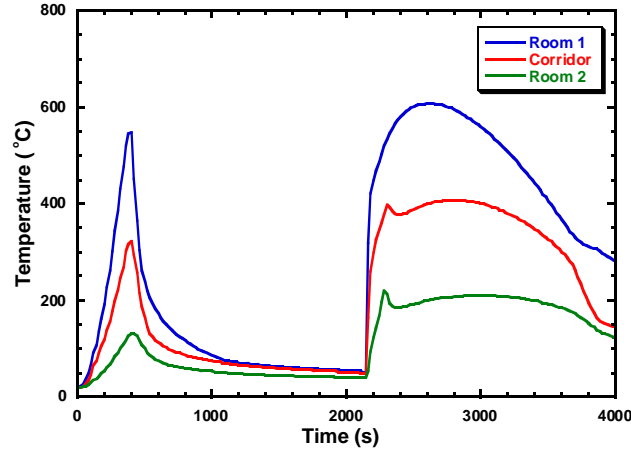
## Appendix C: P-Flash Limitations

In this section, P-Flash is tested against a new test set and it is believed that the following evaluation helps to reveal the current limitation of P-Flash and to provide guidelines about data requirement for the development of a more robust flashover prediction model in multi-compartment buildings.

The new test set accounts for three additional effects: 1) a different surface material, 2) arbitrary opening conditions of vents, and 3) a different fire growth behavior. For the new surface material concrete [12] is considered and the thermal conductivity, specific heat, density, thickness and surface emissivity is given as  $1.75 \text{ W/m}^2$ ,  $1 \text{ kJ/(kg}\cdot\text{C)}$ ,  $2200 \text{ kg/m}^3$ ,  $0.15 \text{ m}$ , and  $0.94$ , respectively. For vent conditions, all vents such as doors and windows are initially closed, but they can be opened at any time during a numerical experiment. Due to the arbitrary opening of the closed vents, a more complex fire growth behavior can also be introduced. For example, in Room 1 with two initially closed vents, a t-squared fire will begin to decay due to depletion of oxygen. When there is an opened door or window in Room 1, fresh air is entrained to the room. Given the added oxygen, a fire may continue to grow. Figure B1 shows the corresponding heat detector temperature profiles. This kind of event provide fires with double-peak growing behavior which is different from the t-squared fire. This example case is denoted as Case 3 and is discussed below. Similar to that described in Section 2, the remaining numerical setups are identical. In general, the simulation time for each numerical experiment is  $8400 \text{ s}$ , and the temperature output interval is  $20 \text{ s}$ . In total, there are 4000 different cases.

In order to provide insights on the influence of each of the additional effects, model performance against two limiting scenarios are first provided in Table B1. Scenario 1 consists of cases where only the effect of new surface material is accounted for where all vents are still open. For Scenario 2, besides having new surface material, cases with arbitrarily opening a Room 1 exterior window and the Room 2 exterior door are also considered. As shown in the table, the model performance for Scenario 1 remains similar to that of seen in Table 4. A physical interpretation for having relatively similar results is that since P-flash learns patterns associated with the higher level temperature information, such as the statistically-based rate of change in temperature for the heat detectors, and with the fact that the change of wall material does not lead to a significant change in temperature behavior, the model is capable of providing reliable predictions. However, model performance drops substantially for Scenario 2, especially in Phase IV where temperature signals from the heat detectors are no longer available, the MAE increases to more than  $150 \text{ }^\circ\text{C}$ . Figure B2 presents temperature comparison between ground-truth (black dash line) and P-Flash predictions (red line for Phase II and III and blue line for Phase IV). As shown in the figure, the temperature predictions in Phase II and III capture the relative trend as compared to the ground-truth and same behavior is observed even for the 2<sup>nd</sup> temperature rise appearing at around  $2150 \text{ s}$ . However, it can be seen that the prediction in Phase IV captures neither the trend nor magnitude of the ground-truth. The large discrepancy in Phase IV is largely due to the assumption imposed on the memory component (M) that the overall temperature behavior assumes either the sigmoidal binding function or the high order polynomial function based on the initial temperature rise. Table B1 shows the overall model performance of P-Flash. An increase to MAE in both Phase II and III is observed and the decrease in model performance is primarily due to the cases where there are arbitrarily

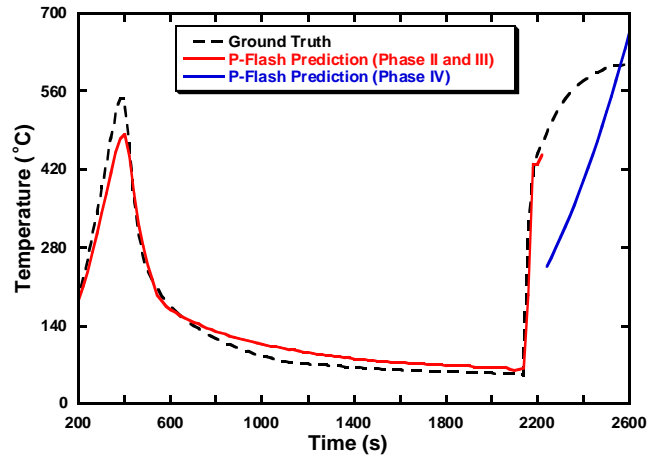
opening doors between Room 1 and Corridor, and Corridor and Room 2 (when doors are closed, the temperature from the corresponding heat detector in a particular compartment remains essentially at room temperature). In order to overcome the data complexity inherent in the new test set, additional data is needed for modeling training and additional treatments are required to facilitate the learning during the training process. This work is currently underway.



**Figure B1.** Temperature profiles for an example case (Case 3) demonstrating the effect of arbitrarily opening to vents.

**Table B1.** P-Flash performance again new test set for current prediction.

	Phase II	Phase III	Phase IV
	MAE (°C)	MAE (°C)	MAE (°C)
<b>Scenario 1</b>	10	14	33
<b>Scenario 2</b>	23	30	> 150
<b>Overall</b>	47	58	> 150



**Figure B2.** Comparison between ground truth and predictions (current) obtained from P-Flash for Case 3.