ST-MELD: Spatiotemporal Model-Estimated Lava Distribution

Jason Ives

University of Guelph
Dr. John Lindsay

April 13, 2026

1 Introduction

Active volcanos can be found throughout the world, often near significant human populations. Research by Small and Naumann (2001) suggests that almost 9% of the global population lives within 100km of an active volcano. The unanticipated eruption of these volcanos has the potential to cause millions of dollars in damage (World Bank, 2022), and result in significant loss of life (Doocy et al., 2013). Fortunately, monitoring and overall prediction of volcanic behavior has advanced quickly in recent years (U.S. Geological Survey, 2026a), and mature volcano hazard monitoring programs (Province of British Columbia, 2026; U.S. Geological Survey, 2026b) are now in place.

However these programs focus on generalized eruption prediction and public safety. Eruption impact management and mitigation is generally left to be overseen at the local level. One aspect of planning for volcano-based contingencies is understanding which localized areas are at risk due to lava flow. **Spatiotemporal Model-Estimated Lava Distribution**, or **ST-MELD**, is designed to help officials and individuals in volcano adjacent locations anticipate localized lava flow risk and make informed decisions while planning their communities and creating preparedness plans.

ST-MELD is a deterministic, empirically driven modeling system that estimates the behavior of lava flows based on lava characteristics and volcanic topography. Key lava characteristics such as composition, temperature, and thickness are combined with topographical data captured from a Digital Elevation Model (DEM) to estimate lava flow rates and lava distribution across a region. Different lava distribution rasters can be created based on a wide-variety of conditions, allowing for comparative analysis and impact range estimation.

2 Related Work

Central to the design of of ST-MELD is the Jeffreys equation (Jeffreys, 1925), which describes average viscosity of a flow over an inclined plane. This equation was adapted to estimate lava flow rates by Nichols (1939). Together these equations significantly improved lava viscosity modeling and increased accuracy in lava flow estimation. Other work in the field of lava flow mechanics has recognized the behavior of most flowing lava as that of a Bingham fluid (Dragoni et al., 1986) and further refined the lava flow rate formula to include a yield strength factor, which is not accounted for in ST-MELD. Instead, ST-MELD focuses on implementation of the Nichols-adapted Jeffreys equation, deriving estimates for density from work by Tenzer et al. (2011) documenting rock density values in New Zealand, oxide composition estimates from the lecture on chemical composition of igneous rocks by Finn

(nd)[1], and viscosity estimates from the work of Giordano et al. (2008) on modeling lava viscosity using the Vogel-Fulcher-Tammann (VFT) equation. The resulting model components are combined in a Python based model using Whitebox Workflows (Lindsay, 2026) for key raster transformations.

3 Data

3.1 Input Data

ST-MELD utilizes a variety of user input data sources, allowing for model customization and exploration. Users can input several model parameters (Table 1) based on data they have available, or as part of an exploratory analysis. These parameters include selecting the lava material type - basalt, andesite, or rhyolite; the lava thickness; the lava temperature; and the model duration.

	Default Value	**Possible Values**	**Out of Range Entry Allowed?**	**Units**
Lava Material	Basalt	Basalt, Andesite, Rhyolite	No	n/a
Lava Thickness	1	1 - 100	Yes	Metres
Lava Temperature	1000	600 - 1300	Yes	°C
Model Duration	60	1 - 1440	Yes	Minutes

Table 1: Fields for capturing user input data.

3.2 Embedded Data

ST-MELD also utilizes a reference data layer (Appendix D) that is recalled and processed based on user selection. By selecting the lava type material, the user triggers the loading of an average density value Tenzer et al. (2011), and a set of oxide percentage values Finn (nd) for that material from a comma-separated value (csv) file. These values are used in calculating the Jeffreys equation value.

4 Methodology

ST-MELD is written in Python 3.13.2, using VSCode. The programming is divided into 3 files:

[1]Le Maitre (1976) is referenced as original source, but source data tables could not be found

4.1 st_meld.py

This is a small initialization file for the model interface (Appendix A). It exists primarily to provide a layer of abstraction between user interface definition and the program execution.

4.2 st_meld_model.py

This file (Appendix B) defines the `LavaFlowModel` class, which encapsulates the core modeling functionality.

4.2.1 The Jeffreys Equation

The primary engine of ST-MELD is the Jeffreys equation for flow viscosity (Jeffreys, 1925), as adapted for lava flow rates by Nichols (1939).

$$\bar{u} = \frac{\rho g t^2}{3\eta} * \sin(\alpha)$$

where:

$\bar{u}$	Average velocity	(m/s)
ρ	Melt density	(kg/m^3)
g	Gravitational constant	(m/s^2)
t	Flow thickness	(m)
η	Flow viscosity	(Pa·s)
α	Slope angle	(degrees)

The ST-MELD implementation of the Jeffreys equation results in a raster of the same extent as the user-provided DEM raster, containing lava flow rate values for each cell.

Melt Density Melt density values, meausred in kg/m^3, are based on work by Tenzer et al. (2011), mapping rock density values across New Zealand. It is important to note that the use of rock-based density values rather than lava-based values means the density values used in ST-MELD likely overestimate the true lava density by approximately 10% (Moore, 2001), due to thermal expansion effects at high temperatures. These static density values are stored in a csv file and recalled based on the user-selected lava material type.

Gravitational Constant The gravitational constant used in the Jerfreys equation is an established scientific constant, meausred in m/s^2. It's value is hard-coded.

Flow Thickness Flow thickness, mesaured in metres, is a user-provided parameter with a default value of 1 metre. A range of 1-100 metres is soft-enforced within the user interface controls, but this limit can be bypassed with manual numeric entry.

Flow Viscosity Viscosity is a key factor in lava flow rate determination (Zeinalova et al., 2025). To establish a strong viscosity estimate for use in the Jeffreys equation, ST-MELD utilizes the Vogel-Fulcher-Tammann (VFT) equation as proposed by Giordano et al. (2008).

$$\log \eta = A + \frac{B}{T - C}$$

Where A, B, and C are melt composition oxide coefficients, and T is lava temperature.

A In the VFT equation, the A coefficient is a pre-exponential factor representing a minimum viscosity at infinite temperature. Giordano et al. (2008) assume this value to be constant and set it at -4.55. ST-MELD follows suit.

B The B coefficient of the VFT equation represents the viscosity rate of change as temperature changes. In otherwords, a low B value means the viscosity will change rapidly as temperature changes. A high B value indicates the viscosity changes slowly as temperature changes. The constituent coefficients for the calculation of the B value represent weight factors for a series of constituent oxide components. The formula for combining these oxides is:

$$B = \sum_{i=1}^{7} [b_i M_i] + \sum_{j=1}^{3} [b_1 j (M1_1 j * M2_1 j)]$$

Where i, j are weight reference keys (Appendix E) and the M values represent the various mol% oxide levels comprising the lava material. These oxide levels are recalled from a csv file basd on the user-selected lava material type.

C The C coefficient of the VFT equation captures the temperature at which the lava viscosity becomes infinite. As the actual temperature approaches the C temperature, the denominator in the VFT equation shrinks, drastically increasing the viscosity. As with the B coefficient, the C coefficient is calculated using weighted constituent oxide component factors. The formula for the C coefficient is:

$$C = \sum_{i=1}^{6} [c_i N_1] + [c_1 1 (N1_1 1 * N2_1 1)]$$

Where i is the weight reference key (Appendix F) and the N values represent the various mol% oxide levels comprising the lava material. These oxide levels are also recalled from a csv file based on the user-selected lava material type.

Temperature For the ST-MELD architecture, temperature is a user-provided value. It has a default value of 1000°C, and a soft-enforced range of 600°C to 1300°C. Values outside of that range can be applied with manual numeric entry. Before use in the VFT equation, the user-provided tempertaure is converted to degrees Kelvin.

Slope Angle The slope angle is calculated for each cell in the user-provided primary DEM. ST-MELD uses Whitebox Workflows (Lindsay, 2026) to execute a series of raster transformations, resulting in a raster containing the sine of the slope for each cell, which is used in the flow rate calculations.

- **Depression Filling:** The first step in transforming the user-provided DEM is depression filling, using the Wang and Liu algorithm (Wang and Liu, 2006). This is implemented using the `fill_depressions_wang_and_liu` function in Whitebox Workflows. This process fills in small depressions in the DEM, ensuring minor depressions don't unrealistically impact the lava flow rates and distribution.
- **Slope Calculation:** Using the Whitebox Workflows `slope` function, the slope is calculated for each cell in the depression filled raster. These slope results are converted to radian units, for compatibility with the sine slope calculation.
- **Sine Slope Calculation:** The slope raster is used to calculate the sine slope raster using the NumPy `sin` function. This raster is used in the $\sin(\alpha)$ portion of the Jeffreys equation.

4.2.2 Lava Distribution

Lava distribution is modeled by iteratively progressing the leading edge of the lava flow as it moves away from the lava source. A list of leading edge cells is maintained, and their remaining traversal time is tracked. Each iteration, the cell with the least remaining traversal time becomes the active cell, and the accumulated flow time tracker of the model is updated to reflect that the cell has been fully traversed. All other traversal times in the leading edge traversal time list are also updated to reflect their traversal progress during that same time period. The completed cell is removed from the leading edge and traversal time lists, and it's downhill neighbor as determined by aspect is added to the leading edge and traversal

time lists. If there remains a difference between the accumulated traversal time and the user-provided target time, the iterative process continues with the next cycle. The lava distribution modeling stops when the list of leading edge cells is empty, or the accumulated traversal time meets or exceeds the user-provided target time.

The lava distribution raster is iteratively developed by updating the cell value of each member of the leading edge list to 1 as it is added to the list. When the lava distribution modeling process is complete, the final lava distribution raster is multiplied by the flow rate raster to replace the 1 in cells marked as having lava with the flow rate, leaving the remaining cells with a 0 value.

Aspect and Leading Edge Membership Aspect data is calculated using the Whitebox Workflows `aspect` function, which transforms the depression filled raster. Within the processing loop, the aspect data for the active cell is categorized to it's closest cardinal or intercardinal direction in degrees. This directional signal is further transformed to a relative cell location to determine which cell will join the leading edge of the lava flow next.

Traversal Time Before lava distribution modeling begins, a raster of traversal times is calculated using the flow rate raster. Traversal time for each cell is calculated by averaging the x and y cell sizes, and multiplying that value by the reciprocal of the cell-level flow rate; effectively dividing the distance lava needs to travel to cross the cell by the flow rate within that cell to find the time it will take to traverse the cell.

$$t_{cell} = \left(\frac{\Delta x + \Delta y}{2}\right) \cdot \frac{1}{Q}$$

4.2.3 Lava Source

The lava source dictates the origin of the lava within the model. Users can provide a binary mask raster to identify a designated region of the DEM as the lava source, or ST-MELD can define the buffered maximum elevation of the user-provided DEM as the lava source.

Highest Point The highest point in the DEM is detected by reading the `configs.maximum` value from the filled dem raster. When the lava distribution raster is subsequently initialized, the highest point and a buffer of surrounding cells are designated as a lava-containing cells. Those cells are then processed to determine the leading edge, starting the lava distribution iteration process.

Mask Raster As an alternative to the highest point lava source, users can provide a mask raster to explicitly define a lava source. The mask raster should be derived from the primary input raster, and use the same coordinate system. If a mask raster is provided by the user, it takes the place of the highest point + buffer raster in the workflow, and is subsequently processed to determine leading edge cells and start the distribution iteration process.

4.3 st_meld_ui.py

This file defines the ST-MELD Graphical User Interface (GUI) (Figure 1), primarily as the `LavaFlowUi` class, along with a set of non-class utility functions (Appendix C).

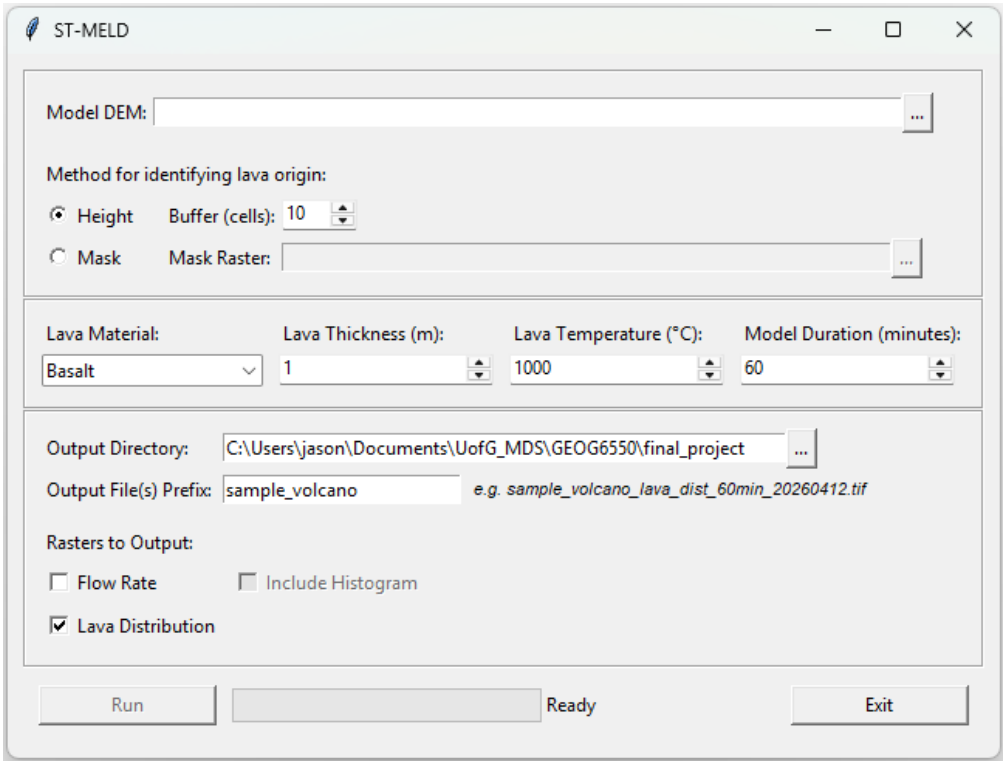

Figure 1: The ST-MELD user interface. The top frame is for configuring model inputs. The center frame is for model parameterization. The bottom frame is for configuring model outputs.

The interface is divided into 3 sections:

Inputs The top frame of the GUI is for configuring model inputs.

- **Model DEM:** The path and filename of the primary DEM, representing the topography of interest. The user can manually enter a file path, or use the the file selector button to navigate to and select the file, but a value must be provided for the model

to run. While ST-MELD is designed to model lava flow over volcanic topography, the model can process a potential lava distribution on any DEM raster with appropriate formatting and metadata.

- **Method for identifying lava origin:** Specification of how the model should define the lava origin. The user can choose **Height** to let the model auto-detect the highest point, or **Mask** to use a user-supplied mask raster. If the height method is chosen, the user can specify a buffer size (in cells - default: 10) around the highest point that will define the lava source. If the mask raster method is used, the user should provide a path and filename for a binary mask raster that marks out the source of the lava flow, with the same dimensions and PCS as the primary input DEM. This can be done by either manually entering a file path or by using the file selector button to navigate to and select the file.

Parameters The center frame of the GUI is for confiuring user specified model parameters, as described in Section 4.2.3.

Outputs The bottom frame of the GUI is for configuring model outputs.

- **Output Directory:** The path of the directory (default: current working directory) for result file output. All output files will be stored in the selected directory. The user can manually enter a directory path, or use the the file selector button to navigate to and select the directory, but a value must be provided for the model to run.
- **Output File(s) Prefix:** A descriptive prefix to be appended to the output files. This text will become the leading text of all output file names, as demonstrated in the accompanying example text. Output filenames will also show the file type, model duration, and date.
- **Rasters to Output:** Options for selecting which rasters to output. Users can choose to output a lava distribution raster, a flow rate raster, or both. If the user outputs a flow rate raster, they can also include an HTML-based histogram of flow rates generated using the Whitebox Workflows `raster_histogram` function.

Below the outputs frame of the GUI is a small execution section, including run and exit buttons, and a dynamic status bar.

5 Results

As described Section 4.3, ST-MELD outputs up to 3 separate files.

5.1 Lava Distribution Raster

The lava distribution raster (Figure 2) represents the estimated lava flow based on the user-provided DEM and lava source, lava characteristics, and modelling timeframe. It maintains the same extent and coordinate system as the user-provided DEM, and can be used as an overlay layer within GIS software.

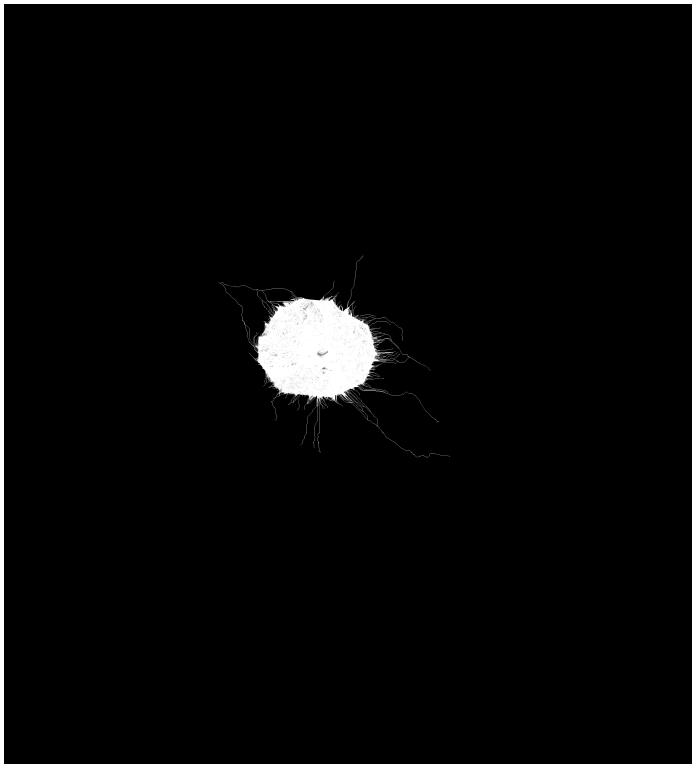

Figure 2: An unprocessed ST-MELD lava distribution raster. This example shows source-masked lava distribution, modeled using a DEM of Mt. Baker, Washington. Lava characteristics include andesite lava type, 1 metre thickness, 1000 °C temperature, and 100 minutes modeling time.

5.2 Lava Flow Rate Raster and Histogram

The Jeffreys equation flow rate raster (Figure 3) is used to generate the lava distribution raster, but can also be output by the model and used for standalone analysis. The raster maintains the same extent and coordinate system as the user-provided DEM. Flow rates are encoded for each active cell.

This raster can be accompanied by an HTML histogram of flow rate values, generated using using the Whitebox Workflows `raster_histogram` function. This is an interactive histogram that provides x and y bin values on hover, and can alternate between probability distribution and cumulative distribution modes, in addition to the primary histogram mode. The example histogram shows flow rate values ranging from 0 to 3.99 m/s.

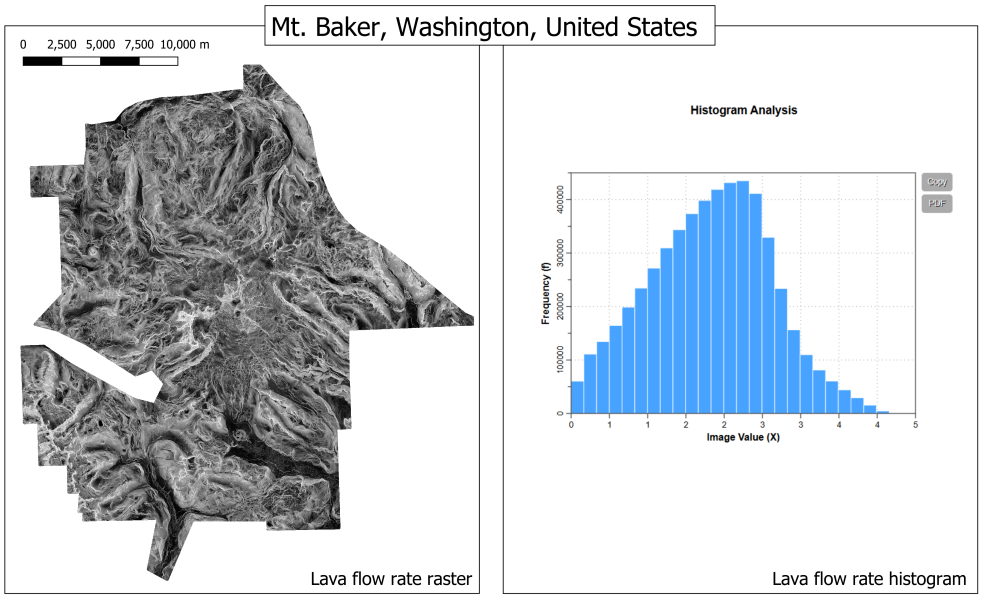

Figure 3: A combined display of the Jeffreys equation flow rate raster and it's accompanying HTML histogram. These are two separate files when output by ST-MELD. Modeled using a DEM of Mt. Baker, Washington. Lava characteristics include andesite lava type, 1 metre thickness, and 1000 °C temperature. Processed using QGIS.

5.3 Validation

Validation of the lava flow rate and distribution functionality focused on controlled intra-model comparative experiments. These experiments were designed to confirm that systematic changes to various model parameters affect the flow rate and resulting lava distribution in ways that align with the mathematical foundations of the Jeffreys equation and established lava flow mechanics.

- **Lava type:** Basalt lava flows can typically advance at a rate of 10 km/h or more on steep slopes, while andesite flows typically advance at a rate of a few km/h. Rhyolite, slower still, often moves less than a few metres in an hour (U.S. Geological Survey, 2023). While there is no assumption of otherwise equal lava characteristics in this data, the relationship between flow rate and lava type nonetheless aligns well with

ST-MELD results, as seen in Figure 4. The difference in flow rate between lava types is broadly attributable to differences in density and viscosity, with increases in density leading to a higher flow rate, and with increases in viscosity leading to a lower flow rate. In fact lava type distinctions in ST-MELD can be seen as something of a proxy for shared variations in density and viscosity.

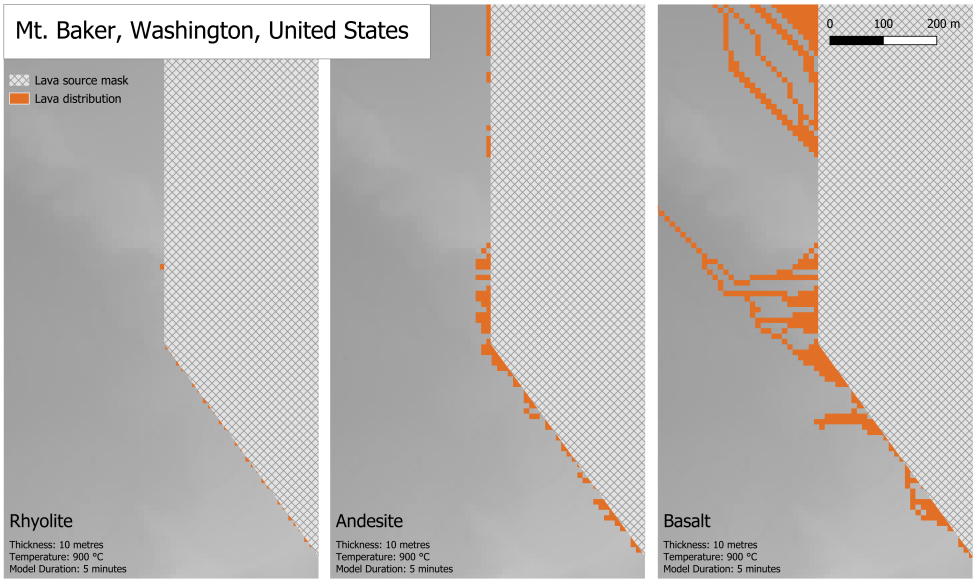

Figure 4: A comparison of lava distribution across 3 different lava materials; rhyolite, andesite, and basalt. This confirms the relative flow rate levels align with documented lava flow mechanics. Processed using QGIS. All other factors are held equal.

- **Thickness:** In the Jeffreys equation, the factor for lava thickness is a multiplier within the numerator. This suggests that, all other factors being equal, the thicker a lava flow is, the higher it's flow rate will be, resulting in a more widespread distribution. The results of a comparative analysis of lava thickness within ST-MELD (Figure 5) confirm this behavior.

- **Temperature:** Within the ST-MELD modeling architecture, temperature resides in the denominator of the VFT equation for viscosity. As such, increases in temperature will serve to drive viscosity down, which in-turn will increase the resulting Jeffreys equation flow rate. A comparative analysis of flow rate as lava temperature varies (Figure 6) reflects this relationship, and shows the importance of viscosity in lava flow rate (National Park Service, 2023), changing considerably over 100 °C intervals.

- **Duration:** Modelling duration is an important factor for risk analysis and exploratory use-cases. While elementary, it is still important to confirm that lava flow distribution

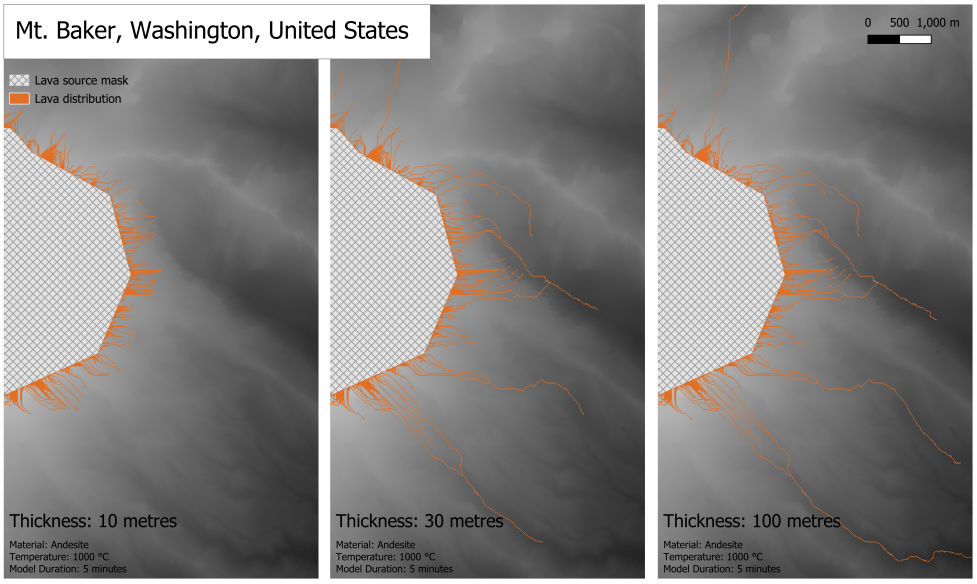

Figure 5: A comparison of three levels of lava thickness, showing that as thickness increases, flow rate increases. This confirms thickness mechanics within ST-MELD align well with the Jeffreys equation. Processed using QGIS. All other factors are held equal.

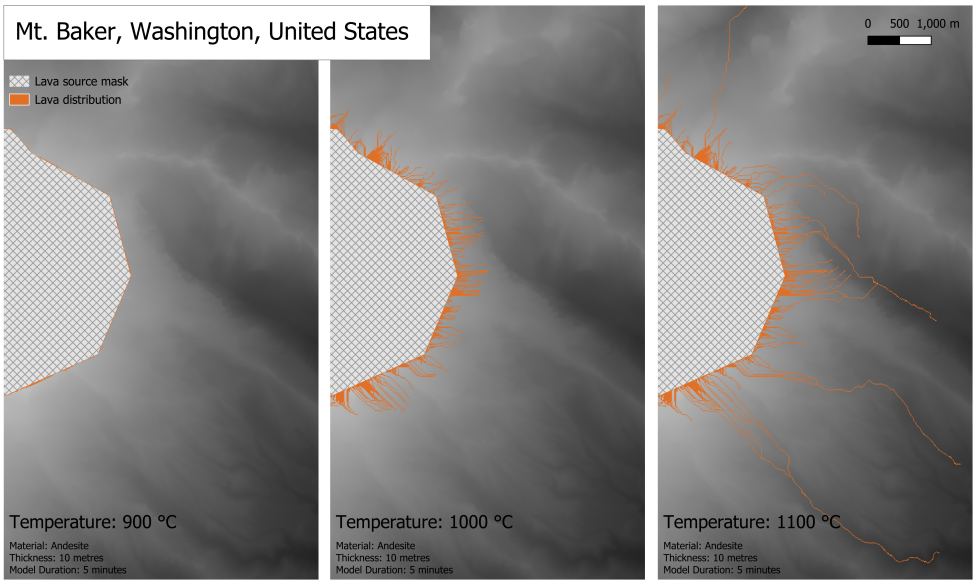

Figure 6: A comparison of three levels of lava temperature, showing that as temperature increases, flow rate increases. This confirms temperature mechanics within ST-MELD align well with the Jeffreys equation. Processed using QGIS. All other factors are held equal.

increases over time. Comparative analysis of flow rate over different modeling windows (Figure 7) shows that ST-MELD performs as expected, increasing flow distribution as the modeling duration increases.

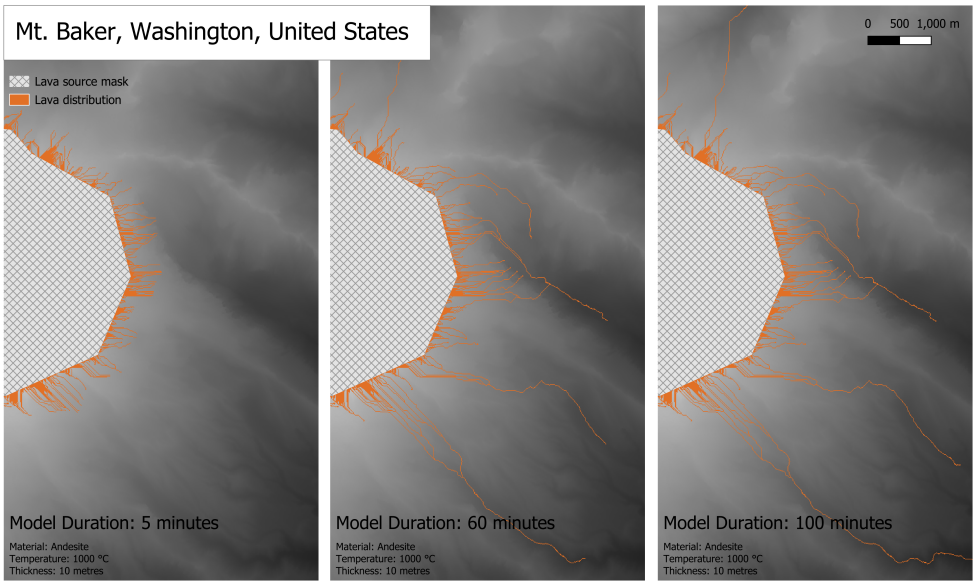

Figure 7: A comparison of three levels of modeling duration, confirming that as modeling duration is extended, lava distribution increases. Processed using QGIS. All other factors are held equal.

6 Discussion and Limitations

Comparative analysis of ST-MELD modeling outputs across parameters confirms a generalized alignment with lava flow mechanics. Additionally, review of a sample of histogram data for andesite at 1 metre thickness and 1000 °C shows a flow rate range of $0m/s$ to $3.99m/s$; an order of magnitude appropriate range of values based on reference estimates (U.S. Geological Survey, 2023). This shows that ST-MELD is succesfully estimating Jeffreys equation lava flow rates and translating those rates into lava distribution maps.

ST-MELD has a number of key limitations that also need to be considered.

- **General estimation** While estimation is a definitional part of any model, ST-MELD could achieve better accuracy through reduced general estimation. Responsiveness could be significantly improved for lava density in particular. Lava density currently is estimated with a single static value for each lava type, regardless of variation in temperature or thickness.

- **Accumulation** There are no accumulation or dispersion factors in the ST-MELD architecture. This means when two flows are joined during lava distribution, no change in lava thickness is registered. Each cell in the distribution raster is, once traversed, assumed to have the same amount of lava as every other cell at each subsequent timestep. This lack of accumulation also prevents the lava from filling existing low spots and continuing to flow, something actual metres-thick lava does regularly.

- **Yield strength** Yield strength is a key factor in modern lava distribution and flow rate models, and ST-MELD does not account for it. This limitation is tied to the lack of an accumulation factor, and in broad strokes means that the lava is always assumed to be flowing and bears no minimum shear stress required to break it's internal equilibrium and transition the lava to a flowing state.

- **Cooling** Lava in the ST-MELD ecosystem never cools, and maintains it's user-specified temperature throughout the entire modeling process. As evidenced during the validation phase, temperature has a strong effect on viscosity and thusly on lava flow rates. Even a few degrees of cooling would likely have a significant impact on lava distribution and should be considered in future models.

- **Effusion rate** Effusion rate, or the rate lava exits the source, is one of the most important factors in lava flow rate. ST-MELD does not account for any specific effusion rate, and instead assumes the effusion rate to be exactly what is necessary to advance a lava flow of the specified thickness at the calculated flow rate. Establishing an effusion factor to "push" the lava flow forward is a complex but important part of any modern lava flow model.

- **Cell traversal algorithm** The cell traversal algorithm makes certain unrealistic assumptions about the behavior of lava within the cell. Primarily, it assumes full traversal of the current cell before the lava moves to the next cell. This is unrealistic since lava will take the shortest path between it's inlet and outlet points within the cell, and leads to a likely overestimation in overall traversal times throughout the model.

Despite those limitations, ST-MELD shows solid performance in calculating Jeffreys equation-based lava flow rates and translating those rates into a distribution of lava flows.

7 Conclusion

ST-MELD strives to be a flexible, accessible, and powerful modeling tool for estimating lava flow rates and lava distributions. With as little as a single DEM, a user can do an exploratory

analysis of lava distribution under a variety of conditions and extract a complete lava flow rate raster for further analysis. This information can help community members and officials in volcano-adjacent areas improve eruption preparedness and make risk-aware decisions about the future of their community.

By combining the Jeffereys equation for liquid flow rates and the Vogel-Fulcher-Tammann (VFT) equation for lava viscosity, ST-MELD creates a strong foundation for modeling lava behavior. The tool then builds upon that foundation by calculating raster-based values for slope, aspect, and lava flow rate. Finally, the lava flow rates are used to calculate raster-based traversal time data, and eventually to calculate lava distribution over time. This scaffolding develops ST-MELD into an efficient, deterministic, spatio-temporal model of lava behavior.

References

Doocy, S., A. Daniels, S. Dooling, and Y. Gorokhovich (2013). The human impact of volcanoes: a historical review of events 1900-2009 and systematic literature review. *PLoS Currents 5*.

Dragoni, M., M. Bonafede, and E. Boschi (1986). Downslope flow models of a bingham liquid: Implications for lava flows. *Journal of Volcanology and Geothermal Research 30*(3), 305–325.

Finn, G. C. (n.d.). Igneous rocks 2. Lecture notes, ERSC 3P21: Igneous and Metamorphic Petrology, Brock University. Accessed March 29, 2026.

Giordano, D., J. K. Russell, and D. B. Dingwell (2008). Viscosity of magmatic liquids: A model. *Earth and Planetary Science Letters 271*(1-4), 123–134.

Jeffreys, H. (1925). The flow of water in an inclined channel of rectangular section. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 49*(293), 793–807.

Le Maitre, R. W. (1976, 11). The chemical variability of some common igneous rocks. *Journal of Petrology 17*(4), 589–598.

Lindsay, J. B. (2026). Whitebox workflows.

Moore, J. G. (2001). Density of basalt core from hilo drill hole, hawaii. *Journal of Volcanology and Geothermal Research 112*(1-4), 221–230.

National Park Service (2023). Lava flow forms. Accessed: April 12, 2026.

Nichols, R. L. (1939). Viscosity of lava. *The Journal of Geology 47*(3), 290–302.

Province of British Columbia (2026). Volcanoes — ClimateReadyBC. Accessed: 2026-04-02.

Small, C. and T. Naumann (2001). The global distribution of human population and recent volcanism. *Global Environmental Change Part B: Environmental Hazards 3*(3), 93–109.

Tenzer, R., P. Sirguey, M. Rattenbury, and J. Nicolson (2011). A digital rock density map of New Zealand. *Computers & Geosciences 37*(8), 1181–1191.

U.S. Geological Survey (2023). Lava flows destroy everything in their path. Accessed: April 12, 2026.

U.S. Geological Survey (2026a). Comprehensive monitoring provides timely warnings of volcano reawakening. Accessed: 2026-04-02.

U.S. Geological Survey (2026b). Volcano hazards program. Accessed: 2026-04-02.

Wang, L. and H. Liu (2006). An efficient method for identifying and filling surface depressions in digital elevation models for hydrologic analysis and modelling. *International Journal of Geographical Information Science 20*(2), 193–213.

World Bank (2022, February). Tonga volcanic eruption and tsunami: World bank global rapid post-disaster damage estimation (grade) report. Technical report, The World Bank, Washington, DC. Accessed: 2026-04-02.

Zeinalova, N., G. Bilotta, A. Ismail-Zadeh, A. Cappello, G. Ganci, and F. Schilling (2025). Influence of rheological parameters on lava flow morphology inferred from numerical modelling. *International Journal of Earth Sciences 114*(4), 697–714.

A Python Code: st_meld.py

```
import st_meld_ui as ui

def main():
  ui.run()

main()
```

B Python Code: st_meld_model.py

```
import math
import webbrowser

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from whitebox_workflows import WbEnvironment


class LavaFlowModel:
  def __init__(self):
    print("*************************\nInitializing Lava Flow Model\
       nNOTE: DEM x,y units should be meters, otherwise results will
        be invalid\n*************************")
    ##MODEL VALUES AND DATA TO BE PASSED FROM UI
    self.dem_path = ""
    self.origin_flag = -1
    self.origin_buffer = -1
    self.origin_raster_path = ""
    self.lava_data = pd.DataFrame()
    self.material = ""
    self.thickness = -1
    self.t_celsius = -1
    self.timestep = -1
    self.output_path_flow_rate = ""
    self.output_path_lava_flow_rate_hist = ""
    self.output_path_lava_dist = ""
    self.output_flow_rate_flag = False
    self.output_flow_rate_hist_flag = False
    self.output_lava_dist_flag = False
    self.run_flag = False


    ##INTERNAL CLASS VARIABLES
```

```
    self.wbe = WbEnvironment()
    self.wbe.verbose = False
    #self.wbe.working_directory = r'data'

    ##STATIC VARIABLES
    self.g = 9.81 ##GRAVITATIONAL CONSTANT, M/S^2
    self.giordano_a = -4.55 ##GIORDANO ET AL. 2008 MODEL COEFFICIENT

  def _wght_pct_to_mole_pct_conversion(self, oxides):
    ##CONVERT WEIGHT PERCENT OXIDE COMPOSITION TO MOLE PERCENT FOR
       USE IN GIORDANO MODEL
    molar_masses = {
      'sio2': 60.08,
      'tio2': 79.87,
      'al2o3': 101.96,
      'feo': 71.85,
      'fe2o3': 159.69,
      'mno': 70.94,
      'mgo': 40.30,
      'cao': 56.08,
      'na2o': 61.98,
      'k2o': 94.20,
      'p2o5': 141.94
    }

    moles = {oxide: weight_pct / molar_masses[oxide] for oxide,
       weight_pct in oxides.items()}
    total_moles = sum(moles.values())
    mole_percents = {oxide: (mole / total_moles) * 100 for oxide,
       mole in moles.items()}
    return mole_percents

  def _calc_giordano_b(self):
    #USE GIORDANO ET AL. 2008 MODEL TO CALCULATE COEFFICIENT B FOR
       VFT VISCOSITY CALCULATION
    ##ASSUMING "DRY" MELTS, SETTING TO 0 FOR MODELING PURPOSES
    h2o = 0
    f2o = 0
```

```
    v = h2o + f2o
    feot = self.material_data.get('feo') + self.material_data.get('
        fe2o3')
    fm = feot + self.material_data.get('mno') + self.material_data.
        get('mgo')
    ta = self.material_data.get('tio2') + self.material_data.get('
        al2o3')
    nk = self.material_data.get('na2o') + self.material_data.get('
        k2o')

    b1 = (self.material_data.get('sio2') + self.material_data.get('
        tio2')) * 159.6
    b2 = self.material_data.get('al2o3') * -173.3
    b3 = (feot + self.material_data.get('mno') + self.material_data.
        get('p2o5')) * 72.1
    b4 = self.material_data.get('mgo') * 75.7
    b5 = self.material_data.get('cao') * -39.0
    b6 = (self.material_data.get('na2o') + v) * -84.1
    b7 = (v + np.log(1 + h2o)) * 141.5

    b11 = ((self.material_data.get('sio2') + self.material_data.get
        ('tio2')) * fm) * -2.43
    b12 = ((self.material_data.get('sio2') + ta + self.material_data
        .get('p2o5')) * (nk + h2o)) * -0.91
    b13 = (self.material_data.get('al2o3') * nk) * 17.6

    self.giordano_b = b1 + b2 + b3 + b4 + b5 + b6 + b7 + b11 + b12 +
         b13


def _calc_giordano_c(self):
    #USE GIORDANO ET AL. 2008 MODEL TO CALCULATE COEFFICIENT C FOR
        VFT VISCOSITY CALCULATION
    ##ASSUMING "DRY" MELTS, SETTING TO 0 FOR MODELING PURPOSES
    h2o = 0
    f2o = 0
```

```
    v = h2o + f2o
    feot = self.material_data.get('feo') + self.material_data.get('
        fe2o3')
    fm = feot + self.material_data.get('mno') + self.material_data.
        get('mgo')
    ta = self.material_data.get('tio2') + self.material_data.get('
        al2o3')
    nk = self.material_data.get('na2o') + self.material_data.get('
        k2o')

    c1 = self.material_data.get('sio2') * 2.75
    c2 = ta * 15.7
    c3 = fm * 8.3
    c4 = self.material_data.get('cao') * 10.2
    c5 = nk * -12.3
    c6 = np.log(1 + v) * -99.5
    c11 = ((self.material_data.get('al2o3') + fm + self.
        material_data.get('cao') - self.material_data.get('p2o5')) *
        (nk + v)) * 0.30

    self.giordano_c = c1 + c2 + c3 + c4 + c5 + c6 + c11

def _calc_vft(self):
    ##CALC VOGEL-FULCHER-TALMANN EQUATION FOR VISCOSITY
    if self.giordano_b is not None and self.giordano_c is not None:
        log_viscosity = self.giordano_a + (self.giordano_b / (self.
            t_kelvin - self.giordano_c))
        self.viscosity = 10 ** log_viscosity
    else:
        print("Error: Giordano coefficients not fully defined.")
    pass

def _convert_celsius_to_kelvin(self):
    try:
        self.t_kelvin = self.t_celsius + 273.15
    except Exception as e:
        print(f"Error converting temperature to Kelvin: {e}")
```

```
def _get_origin_raster(self):
  try:
    self.origin_raster = self.wbe.read_raster(self.
       origin_raster_path)
    print("Origin raster successfully extracted.")
  except Exception as e:
    print(f"Error extracting origin raster: {e}")

def _get_dem(self):
  try:
    self.dem = self.wbe.read_raster(self.dem_path)
    print("Elevation data successfully extracted from DEM.")
  except Exception as e:
    print(f"Error extracting elevation data: {e}")

def _dem_to_filled(self):
  ##DEPRESSION FILL TO ENSURE NO CELL HAS NO DOWNSTREAM FLOW
  try:
    self.dem_filled = self.wbe.fill_depressions_wang_and_liu(self.
       dem, flat_increment = 0.001) #PRE-PROCESSING TO REMOVE
       DEPRESSIONS
    print("Depression filling complete.")
  except Exception as e:
    print(f"Error during depression filling: {e}")

def _filled_to_slope(self):
  ##CONVERT DEM TO SLOPE USING WHITEBOX TOOLS
  try:
    self.slope = self.wbe.slope(self.dem_filled, units = "radians
       ") #CONVERT TO RADIANS FOR SIN SLOPE CALCULATION
    print("Slope data successfully calculated from DEM.")
  except Exception as e:
    print(f"Error calculating slope data: {e}")

def _slope_to_sin_slope(self):
  try:
    self.sin_slope = np.sin(self.slope)
    print("Trigonometric slope conversion complete.")
```

```
    except Exception as e:
      print(f"Error converting slope to sine values: {e}")

  def _filled_to_aspect(self):
    ##EXTRACT ASPECT DATA FROM DEM USING WHITEBOX
    try:
      self.aspect = self.wbe.aspect(self.dem_filled)
      print("Aspect data calculated.")
    except Exception as e:
      print(f"Error calculating aspect data: {e}")

  def _find_highest_point(self):
    ##FIND HIGHEST POINT IN DEM TO DEFINE LAVA START POINT
    self.dem_filled.update_min_max()
    self.highest_point = self.dem_filled.configs.maximum
    print(f"Highest point detected: {self.highest_point}m")

  def _initialize_lava_dist(self): ##NEED TO MATCH ORIGIN RASTER
     PROJECTION, EXTENT, AND RESOLUTION TO DEM
    if self.origin_flag == 2:
      self.lava_dist = self.origin_raster.deep_copy()
    elif self.origin_flag == 1:
      self.lava_dist = self.wbe.buffer_raster(input = self.
         dem_filled == self.highest_point, buffer_size = self.
         origin_buffer, grid_cells_units = True)
    else:
      print("Error: Invalid lava origin flag. Please ensure lava
         origin flag is set to either 1 (height) or 2 (mask).")
      return

  def _get_traversal_time(self):
    ##TRAVERSAL TIME ASSUMES LAVA CONSUMES ONE EDGE AND MOVES
       STRAIGHT ACROSS UNTIL IT CONSUMES THE OPPOSITE EDGE, THEN
       MOVES TO THE NEXT CELL - NOT NECESSARILY IN THE SAME
       DIRECTION AS THE MOVEMENT
    self.avg_traversal_time = (self.flow_rates ** -1) * ((self.
       cell_meters_x + self.cell_meters_y) * .5) ##ORDER MATTERS,
       RASTER FIRST THEN SCALAR
```

```
def _preprocess_material(self):
  if self.material is not None and self.lava_data is not None:
    self.density = self.lava_data.loc[self.lava_data['material']
       == self.material, 'density'].values[0]
    row_data = self.lava_data.loc[self.lava_data['material'] ==
       self.material].to_dict(orient='records')[0]
    non_oxide_keys = ['material', 'density']
    ##SPLIT RELEVANT ROW DATA, CONVERT OXIDE PORTION INTO MOLE
       PERCENT, THEN REJOIN
    self.material_data = {key: row_data[key] for key in row_data
       if key in non_oxide_keys} | self.
       _wght_pct_to_mole_pct_conversion({key: row_data[key] for
       key in row_data if key not in non_oxide_keys})
    self._calc_giordano_b()
    self._calc_giordano_c()
    self._calc_vft() ##RECALCULATE VISCOSITY WHEN LAVA TYPE
       CHANGES
  else:
    print("Error: Material data not properly defined. Please
       ensure material type and lava data are properly defined.")

def generate_flow_rate_raster(self):
  try:
    flow_rate_statics = (self.density * self.g * self.thickness
       **2) / (3 * self.viscosity)
    self.flow_rates = self.sin_slope * flow_rate_statics #ORDER
       MATTERS, (POSSIBLE) RASTER FIRST THEN SCALAR
    print("Flow rate raster successfully calculated.")
  except Exception as e:
    print(f"Error calculating flow rate raster: {e}")

def generate_lava_dist_raster(self):
  print(f"Modeling lava distribution over {self.timestep} minutes
     .")
  self._initialize_lava_dist()

  accumulated_time = 0
```

```
status_update = 1

keep_processing = True
lava_dist_nodata = self.lava_dist.configs.nodata
active_cells = []
active_traversal_times = []

##INITIALIZE LIST OF ACTIVE CELLS
for r in range(0, self.lava_dist.configs.rows):
  for c in range(0, self.lava_dist.configs.columns):
    cell_val = self.lava_dist[r, c]
    if cell_val != 0 and cell_val != lava_dist_nodata:
      active_cells.append((r, c))

##CLEAR "LOCKED" CELLS FROM ACTIVE CELLS
for cell in active_cells:
  r, c = cell

  ##SET ASPECT REFERENCE TO DIRECTIONAL DEGREES
  if self.aspect[r,c] > 337.5 or self.aspect[r,c] <= 22.5: ##
     NORTH
    direction = 0
  elif self.aspect[r,c] > 22.5 and self.aspect[r,c] <= 67.5: ##
     NORTHEAST
    direction = 45
  elif self.aspect[r,c] > 67.5 and self.aspect[r,c] <= 112.5: ##
     EAST
    direction = 90
  elif self.aspect[r,c] > 112.5 and self.aspect[r,c] <= 157.5:
     ##SOUTHEAST
    direction = 135
  elif self.aspect[r,c] > 157.5 and self.aspect[r,c] <= 202.5:
     ##SOUTH
    direction = 180
  elif self.aspect[r,c] > 202.5 and self.aspect[r,c] <= 247.5:
     ##SOUTHWEST
    direction = 225
  elif self.aspect[r,c] > 247.5 and self.aspect[r,c] <= 292.5:
```

```
      ##WEST
      direction = 270
    elif self.aspect[r,c] > 292.5 and self.aspect[r,c] <= 337.5:
      ##NORTHWEST
      direction = 315

    ##IDENTIFY CELL TO FLOW LAVA TO
    r_shift = (-1, -1, 0, 1, 1, 1, 0, -1)
    c_shift = (0, 1, 1, 1, 0, -1, -1, -1)
    cell_locs = list(zip(r_shift, c_shift))

    target_cell = cell_locs[direction // 45]

    if self.dem_filled[r + target_cell[0], c + target_cell[1]] ==
      self.dem_filled.configs.nodata or self.lava_dist[r +
      target_cell[0], c + target_cell[1]] == 1:
      active_cells.remove(cell)

##FIND TRAVERSAL TIME FOR ACIVE CELLS
active_traversal_times = [self.avg_traversal_time[cell] for cell
    in active_cells]

##WHILE THERE ARE CELLS TO PROCESS
while (len(active_cells) > 0):
    min_traversal_index = active_traversal_times.index(min(
      active_traversal_times))
    min_traversal_time = active_traversal_times.pop(
      min_traversal_index)
    r,c = active_cells.pop(min_traversal_index)

    ##SET ASPECT REFERENCE TO DIRECTIONAL DEGREES
    if self.aspect[r,c] > 337.5 or self.aspect[r,c] <= 22.5: ##
      NORTH
      direction = 0
    elif self.aspect[r,c] > 22.5 and self.aspect[r,c] <= 67.5: ##
      NORTHEAST
      direction = 45
    elif self.aspect[r,c] > 67.5 and self.aspect[r,c] <= 112.5: ##
```

```
   EAST
  direction = 90
elif self.aspect[r,c] > 112.5 and self.aspect[r,c] <= 157.5:
   ##SOUTHEAST
  direction = 135
elif self.aspect[r,c] > 157.5 and self.aspect[r,c] <= 202.5:
   ##SOUTH
  direction = 180
elif self.aspect[r,c] > 202.5 and self.aspect[r,c] <= 247.5:
   ##SOUTHWEST
  direction = 225
elif self.aspect[r,c] > 247.5 and self.aspect[r,c] <= 292.5:
   ##WEST
  direction = 270
elif self.aspect[r,c] > 292.5 and self.aspect[r,c] <= 337.5:
   ##NORTHWEST
  direction = 315

##IDENTIFY CELL TO FLOW LAVA TO
r_shift = (-1, -1, 0, 1, 1, 1, 0, -1)
c_shift = (0, 1, 1, 1, 0, -1, -1, -1)
cell_locs = list(zip(r_shift, c_shift))

target_cell = cell_locs[direction // 45]

if self.dem_filled[r + target_cell[0], c + target_cell[1]] !=
   self.dem_filled.configs.nodata and self.lava_dist[r +
   target_cell[0], c + target_cell[1]] != 1:
  self.lava_dist[r + target_cell[0], c + target_cell[1]] = 1
     ##ADD LAVA TO TARGET CELL

  accumulated_time += min_traversal_time
  active_traversal_times = [time - min_traversal_time for time
      in active_traversal_times] ##REDUCE TRAVERSAL TIMES BY
     TIME USED BY CURRENT CELL

  active_cells.append((r + target_cell[0], c + target_cell[1])
     ) ##ADD NEWEST CELL
```

```
        active_traversal_times.append(self.avg_traversal_time[r +
            target_cell[0], c + target_cell[1]]) ##ADD TRAVERSAL TIME
             FOR NEW CELL - SHOULD BE DONE AFTER TIME REDUCTION FOR
            CURRENT CELL

      ##PERIODIC UPDATES
      # if (accumulated_time // 60) == status_update:
      #   if status_update == 1:
      #     print(f"Modeling status: {status_update} minute complete
          .")
      #     status_update += 1
      #   else:
      #     print(f"Modeling status: {status_update} minutes
          complete.")
      #     status_update += 1

      ##CONFIRM NEXT ROUND SHOULD BE PROCESSED
      if accumulated_time >= (self.timestep * 60) or len(
          active_cells) == 0:
        break

    self.lava_dist = self.lava_dist * self.flow_rates ##CONVERT TO
       FLOW RATES TO VISUALIZE FLOW RATE THROUGHOUT FLOW
       DISTRIBUTION
    time_mins = accumulated_time / 60
    print(f"Lava dist modeling complete. Elapsed time: {time_mins:.2
       f} minutes.")

def write_flow_rate_raster(self):
    if self.output_path_flow_rate is not None:
      self.wbe.write_raster(self.flow_rates, self.
          output_path_flow_rate)
      print(f"Flow rate raster created: {self.output_path_flow_rate
          }")
      if self.output_flow_rate_hist_flag:
        self.wbe.raster_histogram(self.flow_rates, output_html_file
            = self.output_path_lava_flow_rate_hist)
        webbrowser.open(self.output_path_lava_flow_rate_hist)
```

```
def write_lava_dist_raster(self):
  if self.output_path_lava_dist is not None:
    self.wbe.write_raster(self.lava_dist, self.
       output_path_lava_dist)
    print(f"Lava distribution raster created: {self.
       output_path_lava_dist}")
  else:
    print("Error: Lava distribution output path not defined.")

def run_chunk_one(self):
  if (self.dem_path != "" and
    ((self.origin_flag == 1 and self.origin_buffer >= 0) or (self.
       origin_flag == 2 and self.origin_raster_path != "")) and
    len(self.lava_data) > 0 and
    self.material != "" and
    self.thickness > 0 and
    self.t_celsius > 0 and
    self.timestep > 0 and
    (self.output_path_lava_dist != "" or self.
       output_path_lava_dist != "") and
    (self.output_flow_rate_flag == True or self.
       output_lava_dist_flag == True)):
    self.run_flag = True
    self._convert_celsius_to_kelvin()
    if self.origin_flag == 2:
      self._get_origin_raster()
    self._get_dem()
    self._preprocess_material()

    ##SET CELL SIZE - ASSUMED IN METERS
    self.cell_meters_x = self.dem.configs.resolution_x
    self.cell_meters_y = self.dem.configs.resolution_y
    print(f"Cell size: {self.cell_meters_x}m x {self.cell_meters_y
       }m")
  else:
    run_flag = False
    print("Error: Missing required input values. Please ensure all
```

```
        model parameters are defined before running the model.")

def run_chunk_two(self):
    if self.run_flag:
      ##RASTER PRE-PROCESSING CALCULATIONS - ORDER MATTERS
      self._dem_to_filled()
      self._filled_to_aspect()
      self._filled_to_slope()
      self._slope_to_sin_slope()
      if self.origin_flag == 1:
        self._find_highest_point()

def run_chunk_three(self):
    if self.run_flag:
      ##RUN METHODS THAT ARE DEPENDENT ON USER-DEFINED VARIABLES
      self.generate_flow_rate_raster()
      ##GET PER CELL TRAVERSAL TIMES BASED ON CELL SIZE AND FLOW
         RATES (DEFINED ON INIT)
      self._get_traversal_time()

def run_chunk_four(self):
    if self.run_flag:
      ##ITERATE THROUGH LAVA DISTRIBUTION RASTER, UPDATING BASED
         ON FLOW PROGRESSION
      self.generate_lava_dist_raster()

def run_chunk_five(self):
      if self.output_flow_rate_flag:
        self.write_flow_rate_raster()

      if self.output_lava_dist_flag:
        self.write_lava_dist_raster()
      self.run_flag = False
```

C Python Code: st_meld_ui.py

```
import os
from datetime import date
from pathlib import Path

import pandas as pd
import matplotlib.pyplot as plt
import tkinter as tk
from tkinter import filedialog, ttk

import st_meld_model as meld


class LavaFlowUi(tk.Frame):
  def __init__(self, master=None):
    super().__init__(master)
    self.load_lava_statistics() ##PRELOAD LAVA STATISTICS FOR CONFIG
        AND MODEL RUN
    self.lava_model = meld.LavaFlowModel()

    y_padding = 3

    ##FRAME DEFINITIONS
    ##LEVEL 0
    self.master = master

    ##LEVEL 1
    self.outer_pad_frame = tk.Frame(self.master, padx=10, pady=10)
    self.outer_pad_frame.grid(row=0, column=0, sticky = tk.W + tk.E)

    ##LEVEL 2
    self.title_frame = tk.Frame(self.outer_pad_frame, padx=10, pady
       =10)
    self.title_frame.grid(row=0, column=0, sticky = tk.W + tk.E)

    self.input_frame = tk.Frame(self.outer_pad_frame, bd = 2, relief
        = "groove", padx=10, pady=10)
```

```
self.input_frame.grid(row=0, column=0, sticky = tk.W + tk.E)

self.config_frame = tk.Frame(self.outer_pad_frame, bd = 2,
   relief = "groove", padx=10, pady=10)
self.config_frame.grid(row=1, column=0, sticky = tk.W + tk.E)

self.output_frame = tk.Frame(self.outer_pad_frame, bd = 2,
   relief = "groove", padx=10, pady=10)
self.output_frame.grid(row=2, column=0, sticky = tk.W + tk.E)

self.execution_frame = tk.Frame(self.outer_pad_frame, padx=10,
   pady=10)
self.execution_frame.grid(row=99, column=0, sticky = tk.W + tk.E
   )
self.execution_frame.grid_columnconfigure(0, weight=0)
self.execution_frame.grid_columnconfigure(1, weight=0)
self.execution_frame.grid_columnconfigure(2, weight=0)
self.execution_frame.grid_columnconfigure(3, weight=1) ##
   EXPANSION COLUMN
self.execution_frame.grid_columnconfigure(4, weight=0)

##LEVEL 3
self.lava_origin_method_frame = tk.Frame(self.input_frame)
self.lava_origin_method_frame.grid(row=1, column=0, columnspan
   =3, sticky = tk.W + tk.E)

##==TITLE FRAME ELEMENTS==

##==INPUT FRAME ELEMENTS==
##--MODEL DEM FILE PATH CAPTURE--
self.model_dem_path = tk.StringVar()
self.model_dem_path.trace_add("write", lambda *_: self.
   ready_to_run_check_cmd())
##LABEL
model_dem_input_label = tk.Label(self.input_frame, text="Model
   DEM:")
model_dem_input_label.grid(column = 0, row = 0, padx = (0, 1),
   pady = (y_padding, y_padding + 10), sticky=tk.W)
```

```
##FIELD
model_dem_input_field = tk.Entry(self.input_frame, width=80,
   textvariable=self.model_dem_path)
model_dem_input_field.grid(column = 1, row = 0, pady = (
   y_padding, y_padding + 10), sticky=tk.W)
##BUTTON
model_dem_input_button = tk.Button(self.input_frame, text="...",
    command=self.model_dem_file_selector)
model_dem_input_button.grid(column = 2, row = 0, pady = (
   y_padding, y_padding + 10), sticky = tk.W)

##--LAVA SOURCE METHOD--
self.lava_origin_method_choice = tk.IntVar(value=1)
lava_origin_method_label = tk.Label(self.
   lava_origin_method_frame, text="Method for identifying lava
   origin:")
lava_origin_method_label.grid(column = 0, row = 0, columnspan =
   3, padx = (0, 1), pady = y_padding, sticky=tk.W)
lava_origin_rb1 = tk.Radiobutton(self.lava_origin_method_frame,
   text="Height", variable=self.lava_origin_method_choice, value
   =1, command = self.lava_origin_method_height)
lava_origin_rb1.grid(column = 0, row = 1, padx = (0, 15), pady =
    y_padding-2, sticky=tk.W)
lava_origin_rb2 = tk.Radiobutton(self.lava_origin_method_frame,
   text="Mask ", variable=self.lava_origin_method_choice, value
   =2, command = self.lava_origin_method_mask)
lava_origin_rb2.grid(column = 0, row = 2, padx = (0, 15), pady =
    y_padding-2, sticky=tk.W)

##--SET HEIGHT BUFFER--
self.origin_height_buffer = tk.IntVar(value = 10)
origin_height_buffer_label = tk.Label(self.
   lava_origin_method_frame, text="Buffer (cells):")
origin_height_buffer_label.grid(column = 1, row = 1, padx = (0,
   1), pady = y_padding-2, sticky=tk.W)
self.origin_height_buffer_field = ttk.Spinbox(self.
   lava_origin_method_frame, from_= 1, to = 100, increment = 1,
   width = 5, textvariable = self.origin_height_buffer)
```

```
self.origin_height_buffer_field.bind("<KeyRelease>", self.
   ready_to_run_check)
self.origin_height_buffer_field.grid(column = 2, row = 1, pady =
    y_padding-2, sticky=tk.W)

##--LAVA ORIGIN DEM FILE PATH CAPTURE--
self.origin_raster_path = tk.StringVar()
self.origin_raster_path.trace_add("write", lambda *_: self.
   ready_to_run_check_cmd())
##LABEL
origin_raster_input_label = tk.Label(self.
   lava_origin_method_frame, text="Mask Raster:")
origin_raster_input_label.grid(column = 1, row = 2, padx = (0,
   1), pady = y_padding-2, sticky=tk.W)
##FIELD
self.origin_raster_input_field = tk.Entry(self.
   lava_origin_method_frame, width=65, textvariable=self.
   origin_raster_path)
self.origin_raster_input_field.grid(column = 2, row = 2, pady =
   y_padding-2, sticky=tk.W)
self.origin_raster_input_field.config(state = 'disabled')
##BUTTON
self.origin_raster_input_button = tk.Button(self.
   lava_origin_method_frame, text="...", command=self.
   origin_raster_file_selector)
self.origin_raster_input_button.grid(column = 3, row = 2, pady =
    y_padding-2, sticky = tk.W)
self.origin_raster_input_button.config(state = 'disabled')

##==CONFIG FRAME ELEMENTS==
##--SELECT LAVA TYPE--
##LABEL
lava_material_label = tk.Label(self.config_frame, text="Lava
   Material:")
lava_material_label.grid(column = 0, row = 0, pady = (0,
   y_padding), sticky=tk.W)
##PICK LIST
self.lava_material_choice = tk.StringVar(value=self.lava_stats['
```

```
   material'].tolist()[0])
plot_2_symbol_picklist = ttk.Combobox(self.config_frame,
   textvariable = self.lava_material_choice, values = self.
   lava_stats['material'].tolist(), state = "readonly")
plot_2_symbol_picklist.grid(column = 0, row = 1, pady = (0,
   y_padding), sticky=tk.W)

##--SET THICKNESS--
self.lava_thickness = tk.IntVar(value = 1)
lava_thickness_label = tk.Label(self.config_frame, text="Lava
   Thickness (m):")
lava_thickness_label.grid(column = 1, row = 0, padx = (10, 0),
   pady = (0, y_padding), sticky=tk.W)
lava_thickness_field = ttk.Spinbox(self.config_frame, from_= 1,
   to = 100, increment = 1, textvariable = self.lava_thickness)
lava_thickness_field.bind("<KeyRelease>", self.
   ready_to_run_check)
lava_thickness_field.grid(column = 1, row = 1, padx = (10, 0),
   pady = (0, y_padding), sticky=tk.W)

##--SET TEMPERATURE--
self.lava_temperature = tk.IntVar(value = 1000)
lava_temperature_label = tk.Label(self.config_frame, text="Lava
   Temperature (°C):")
lava_temperature_label.grid(column = 2, row = 0, padx = (10, 0),
    pady = (0, y_padding), sticky=tk.W)
lava_temperature_field = ttk.Spinbox(self.config_frame, from_=
   600, to = 1300, increment = 50, textvariable = self.
   lava_temperature)
lava_temperature_field.bind("<KeyRelease>", self.
   ready_to_run_check)
lava_temperature_field.grid(column = 2, row = 1, padx = (10, 0),
    pady = (0, y_padding), sticky=tk.W)

##--SET MODEL DURATION--
self.model_duration = tk.IntVar(value = 60)
model_duration_label = tk.Label(self.config_frame, text="Model
   Duration (minutes):")
```

```
model_duration_label.grid(column = 3, row = 0, padx = (10, 0),
   pady = (0, y_padding), sticky=tk.W)
model_duration_field = ttk.Spinbox(self.config_frame, from_= 1,
   to = 1440, increment = 5, textvariable = self.model_duration)
model_duration_field.bind("<KeyRelease>", self.
   sample_file_output_update)
model_duration_field.grid(column = 3, row = 1, padx = (10, 0),
   pady = (0, y_padding), sticky=tk.W)

##==OUTPUT FRAME ELEMENTS==
##--OUTPUT DIRECTORY--
self.output_directory = tk.StringVar(value = str(Path(os.getcwd
   ())))
self.output_directory.trace_add("write", lambda *_: self.
   ready_to_run_check_cmd())
output_directory_label = tk.Label(self.output_frame, text="
   Output Directory:")
output_directory_label.grid(column = 0, row = 0, pady = (0,
   y_padding), sticky=tk.W)
self.output_directory_field = tk.Entry(self.output_frame, width
   =60, textvariable=self.output_directory)
self.output_directory_field.grid(column = 1, row = 0, pady = (0,
    y_padding), columnspan = 2, sticky=tk.W)
output_directory_button = tk.Button(self.output_frame, text
   ="...", command=self.output_directory_selector)
output_directory_button.grid(column = 3, row = 0, pady = (0,
   y_padding), sticky = tk.W)

##--SET OUTPUT FILE PREFIX--
self.output_prefix = tk.StringVar(value = "sample_volcano")
output_prefix_label = tk.Label(self.output_frame, text="Output
   File(s) Prefix:")
output_prefix_label.grid(column = 0, row = 1, pady = (0,
   y_padding + 10), sticky=tk.W)
self.output_prefix_field = tk.Entry(self.output_frame, width=25,
    textvariable=self.output_prefix)
self.output_prefix_field.grid(column = 1, row = 1, padx = (0, 5)
   , pady = (0, y_padding + 10), sticky=tk.W)
```

```
self.output_prefix_field.bind("<KeyRelease>", self.
   sample_file_output_update)
self.sample_file_output_label = tk.Label(self.output_frame, font
    = ("", 8, "italic"),
                text=f"e.g. {self.output_prefix_field.get()}
                   _lava_dist_{model_duration_field.get()}min_
                   {date.today().strftime('%Y%m%d')}.tif")
self.sample_file_output_label.grid(column = 2, row = 1, pady =
   (0, y_padding + 10), columnspan = 2, sticky=tk.E)

##--SELECT RASTERS TO OUTPUT--
self.ouput_flow_rate_raster = tk.BooleanVar(value = False)
self.ouput_lava_dist_raster = tk.BooleanVar(value = True)
output_raster_label = tk.Label(self.output_frame, text="Rasters
   to Output:")
output_raster_label.grid(column = 0, row = 2, pady = (0,
   y_padding), sticky=tk.W)
output_flow_rate_checkbox = tk.Checkbutton(self.output_frame,
   text="Flow Rate", variable=self.ouput_flow_rate_raster,
   command = self.hist_checkbox_update)
output_flow_rate_checkbox.grid(column = 0, row = 3, pady = (0,
   y_padding), sticky=tk.W)
output_lava_dist_checkbox = tk.Checkbutton(self.output_frame,
   text="Lava Distribution", variable=self.
   ouput_lava_dist_raster, command = self.ready_to_run_check_cmd
   )
output_lava_dist_checkbox.grid(column = 0, row = 4, pady = (0,
   y_padding), sticky=tk.W)

##--OUTPUT HISTOGRAM CHECKBOX--
self.output_flow_rate_histogram = tk.BooleanVar(value = False)
self.output_flow_rate_histogram_checkbox = tk.Checkbutton(self.
   output_frame, text = "Include Histogram", variable=self.
   output_flow_rate_histogram)
self.output_flow_rate_histogram_checkbox.grid(column = 1, row =
   3, padx = 5, pady = (0, y_padding), sticky=tk.W)
self.output_flow_rate_histogram_checkbox.config(state = '
   disabled')
```

```
  ##==EXECUTION FRAME ELEMENTS==
  ##--RUN BUTTON--
  self.run_btn = tk.Button(self.execution_frame, text="Run",
     command=self.run_model, width = 15)
  self.run_btn.grid(column=0, row=0, padx = (0, 10), sticky=tk.W)
  self.run_btn.config(state = 'disabled')

  ##--PROGRESS BAR--
  self.progress_bar = ttk.Progressbar(self.execution_frame, orient
     ='horizontal', mode='determinate', length=200)
  self.progress_bar.grid(column=1, row=0, pady = y_padding, sticky
     =tk.W)

  self.progress_label_text = tk.StringVar(value='Ready')
  progress_label = tk.Label(self.execution_frame, textvariable =
     self.progress_label_text)
  progress_label.grid(column=2, row=0, pady = y_padding, sticky=tk
     .W)

  ##--EXIT BUTTON--
  exit_btn = tk.Button(self.execution_frame, text='Exit', command=
     self.exit, width = 15)
  exit_btn.grid(column=4, row=0, sticky=tk.E)


##=======================================

##BIND BASED VERSION OF READY TO RUN
def ready_to_run_check(self, event):
  if self.model_dem_path.get() == "":
    self.run_btn.config(state = 'disabled')
  elif self.lava_origin_method_choice.get() == 1 and (self.
     origin_height_buffer_field.get().isdigit() == False or int(
     self.origin_height_buffer_field.get()) <= 0):
    self.run_btn.config(state = 'disabled')
  elif self.lava_origin_method_choice.get() == 2 and self.
     origin_raster_path.get() == "":
```

```
    self.run_btn.config(state = 'disabled')
  elif self.output_directory.get() == "":
    self.run_btn.config(state = 'disabled')
  elif self.output_prefix.get() == "":
    self.run_btn.config(state = 'disabled')
  elif self.ouput_flow_rate_raster.get() == False and self.
     ouput_lava_dist_raster.get() == False:
    self.run_btn.config(state = 'disabled')
  else:
    try:
      self.lava_thickness.get()
      self.lava_temperature.get()
      self.model_duration.get()
      self.run_btn.config(state = 'normal')
    except tk.TclError:
      self.run_btn.config(state = 'disabled')

##COMMAND BASED VERSION OF READY TO RUN
def ready_to_run_check_cmd(self):
  if self.model_dem_path.get() == "":
    self.run_btn.config(state = 'disabled')
  elif self.lava_origin_method_choice.get() == 1 and (self.
     origin_height_buffer_field.get().isdigit() == False or int(
     self.origin_height_buffer_field.get()) <= 0):
    self.run_btn.config(state = 'disabled')
  elif self.lava_origin_method_choice.get() == 2 and self.
     origin_raster_path.get() == "":
    self.run_btn.config(state = 'disabled')
  elif self.output_directory.get() == "":
    self.run_btn.config(state = 'disabled')
  elif self.output_prefix.get() == "":
    self.run_btn.config(state = 'disabled')
  elif self.ouput_flow_rate_raster.get() == False and self.
     ouput_lava_dist_raster.get() == False:
    self.run_btn.config(state = 'disabled')
  else:
    try:
      self.lava_thickness.get()
```

```
      self.lava_temperature.get()
      self.model_duration.get()
      self.run_btn.config(state = 'normal')
    except tk.TclError:
      self.run_btn.config(state = 'disabled')

def model_dem_file_selector(self):
  model_pfn = filedialog.askopenfilename(title='Select modeling
     DEM...', filetypes=[("GeoTIFF Files (.tif)", "*.tif")])
  self.model_dem_path.set(model_pfn)

def lava_origin_method_height(self):
  self.origin_raster_input_field.config(state = 'disabled')
  self.origin_raster_input_button.config(state = 'disabled')

  self.origin_height_buffer_field.config(state = 'normal')

  self.ready_to_run_check_cmd()

def lava_origin_method_mask(self):
  self.origin_raster_input_field.config(state = 'normal')
  self.origin_raster_input_button.config(state = 'normal')

  self.origin_height_buffer_field.config(state = 'disabled')

  self.ready_to_run_check_cmd()

def origin_raster_file_selector(self):
  origin_pfn = filedialog.askopenfilename(title='Select lava
     source mask raster...', filetypes=[("GeoTIFF Files (.tif)",
     "*.tif")])
  self.origin_raster_path.set(origin_pfn)

def hist_checkbox_update(self):
  if self.ouput_flow_rate_raster.get() == False:
    self.output_flow_rate_histogram.set(False)
    self.output_flow_rate_histogram_checkbox.config(state = '
       disabled')
```

```
    else:
      self.output_flow_rate_histogram_checkbox.config(state = '
         normal')

    self.ready_to_run_check_cmd()

  def load_lava_statistics(self):
    self.lava_stats = pd.read_csv(r".\data\lava_statistics.csv")

  def sample_file_output_update(self, event):
    try:
      self.sample_file_output_label.config(text=f"e.g. {self.
         output_prefix.get()}_lava_dist_{self.model_duration.get()}
         min_{date.today().strftime('%Y%m%d')}.tif")
    except tk.TclError:
      self.sample_file_output_label.config(text=f"e.g. {self.
         output_prefix.get()}_lava_dist_<INVALID>min_{date.today().
         strftime('%Y%m%d')}.tif")
    self.ready_to_run_check(event)

  def output_directory_selector(self):
    output_p = filedialog.askdirectory(title='Select output raster
       directory...')
    self.output_directory.set(output_p)

  def run_model(self):
    self.lava_model.dem_path = str(Path(self.model_dem_path.get()))
    self.lava_model.origin_flag = self.lava_origin_method_choice.get
       ()
    if self.lava_origin_method_choice.get() == 1:
      self.lava_model.origin_buffer = int(self.
         origin_height_buffer_field.get())
      self.lava_model.origin_raster_path = ""
    elif self.lava_origin_method_choice.get() == 2:
      self.lava_model.origin_buffer = -1
      self.lava_model.origin_raster_path = str(Path(self.
         origin_raster_path.get()))
    else:
```

```
  self.lava_model.origin_buffer = -1
  self.lava_model.origin_raster_path = ""
self.lava_model.origin_raster_path = self.origin_raster_path.get
   ()
self.lava_model.lava_data = self.lava_stats
self.lava_model.material = self.lava_material_choice.get()
self.lava_model.thickness = int(self.lava_thickness.get())
self.lava_model.t_celsius = int(self.lava_temperature.get())
self.lava_model.timestep = int(self.model_duration.get())
self.lava_model.output_path_flow_rate = get_output_flow_rate_pfn
   (self.output_directory.get(), self.output_prefix.get())
self.lava_model.output_path_lava_flow_rate_hist =
   get_output_flow_rate_hist_pfn(self.output_directory.get(),
   self.output_prefix.get())
self.lava_model.output_path_lava_dist = get_output_lava_dist_pfn
   (self.output_directory.get(), self.output_prefix.get(), self.
   model_duration.get())
self.lava_model.output_flow_rate_flag = self.
   ouput_flow_rate_raster.get()
self.lava_model.output_lava_dist_flag = self.
   ouput_lava_dist_raster.get()
self.lava_model.output_flow_rate_hist_flag = self.
   output_flow_rate_histogram.get()

self.progress_bar['value'] = 10
self.progress_label_text.set('Importing data...')
self.lava_model.run_chunk_one()
self.master.update_idletasks()
self.progress_bar['value'] = 20
self.progress_label_text.set('Transforming DEM...')
self.lava_model.run_chunk_two()
self.master.update_idletasks()
self.progress_bar['value'] = 40
self.progress_label_text.set('Modeling flow rate...')
self.lava_model.run_chunk_three()
self.master.update_idletasks()
self.progress_bar['value'] = 60
self.progress_label_text.set('Modeling lava distribution...')
```

```
    self.lava_model.run_chunk_four()
    self.master.update_idletasks()
    self.progress_bar['value'] = 95
    self.progress_label_text.set('Saving files...')
    self.lava_model.run_chunk_five()
    self.master.update_idletasks()
    self.progress_bar['value'] = 0
    self.progress_label_text.set('Ready')



  def exit(self):
    exit(0)

def get_output_flow_rate_pfn(path, prefix):
  return str(Path(os.path.join(path, f"{prefix}_flow_rate_{date.
     today().strftime('%Y%m%d')}.tif")))

def get_output_flow_rate_hist_pfn(path, prefix):
  return str(Path(os.path.join(path, f"{prefix}_flow_rate_{date.
     today().strftime('%Y%m%d')}_histogram.html")))

def get_output_lava_dist_pfn(path, prefix, timestep):
  return str(Path(os.path.join(path, f"{prefix}_lava_dist_{timestep}
     min_{date.today().strftime('%Y%m%d')}.tif")))

def run():
  root = tk.Tk()
  app = LavaFlowUi(master=root)
  root.wm_title("ST-MELD")
  root.mainloop()
```

D Lava Material Characteristic Data

Material	ρ	**SiO_2**	**TiO_2**	**Al_2O_3**	**Fe_2O_3**	**FeO**	**MnO**	**MgO**	**CaO**	**Na_2O**	**K_2O**	**P_2O_5**
Basalt	2768	50.06	1.87	15.94	3.90	7.50	0.20	6.98	9.70	2.94	1.08	0.34
Andesite	2565	58.85	0.96	16.98	2.55	5.13	0.12	3.73	6.66	3.60	1.81	0.29
Rhyolite	2207	72.04	0.30	14.42	1.22	1.68	0.05	0.71	1.82	3.69	4.12	0.12

Table 2: Material characteristic data utilized by the ST-MELD. Density (ρ) is in kg/m^3; oxides are in wt%.

E VFT B Coefficient Composite Oxide Weights

Factor	Chemical Composition Formula	Weight
b_1	$SiO_2 + TiO_2$	159.6
b_2	Al_2O_3	-173.3
b_3	$(FeO + Fe_2O_3) + MnO + P_2O_5$	72.1
b_4	MgO	75.7
b_5	CaO	-39.0
b_6	$Na_2O + (H_2O + F_2O)$	-84.1
b_7	$(H_2O + F_2O) + \ln(1 + H_2O)$	141.5
b_{11}	$(SiO_2 + TiO_2) \times (FeO + Fe_2O_3 + MnO + MgO)$	-2.43
b_{12}	$(SiO_2 + TiO_2 + Al_2O_3 + P_2O_5) \times (Na_2O + K_2O + H_2O)$	-0.91
b_{13}	$Al_2O_3 \times (Na_2O + K_2O)$	17.6

Table 3: VFT Equation B Coefficient Components and Weights

F VFT C Coefficient Composite Oxide Weights

Factor	Chemical Composition Formula	Weight
c_1	SiO_2	2.75
c_2	$TiO_2 + Al_2O_3$	15.7
c_3	$FeO + Fe_2O_3 + MnO + MgO$	8.3
c_4	CaO	10.2
c_5	$Na_2O + K_2O$	-12.3
c_6	$\ln(1 + H_2O + F_2O)$	-99.5
c_{11}	$(Al_2O_3 + FeO + Fe_2O_3 + MnO + MgO + CaO - P_2O_5) \times (Na_2O + K_2O + H_2O + F_2O)$	0.30

Table 4: VFT Equation C Coefficient Components and Weights