# Software requirements

- Python package manager **uv**
  - https://docs.astral.sh/uv/getting-started/installation/
  - You can pre-cache Python 3.10 and any more modern ones
  - You can pre-cache Python packages:
    - ***jupyterlab***, ***trame***, ***trame-vuetify***, ***vtk***, ***trame-vtk***, ***trame-vtklocal***
- ParaView 5.13.3
  - https://www.paraview.org/download/
- Browser (Chrome, Firefox or from system)
  - https://www.google.com/chrome/
  - https://www.mozilla.org/en-US/firefox/
- Text / Code editor (anything you want or VSCode)
  - https://code.visualstudio.com/

# Hands-On Creation of Bespoke Scientific Visualization Applications with Trame

## HPC-Enabled Web, Jupyter, and Desktop Apps for Simulation, Experiment, and AI Data

Sebastien Jourdain (Kitware), Logan Harbour (INL), Axel Huebl (LBNL), Brian DeVincentis (M-Star), David Rogers (LANL), Victor Mateevitsi (ANL)

# Morning

- Trame fundamentals
  - State and events
- Dynamic web interface
  - Vue.js templating
- Vuetify for beautiful user interface
  - Designing a beautiful UI

---

- 3D Visualization
  - Using VTK and ParaView
- 2D Charts
  - Matplotlib, plotly, altair
- Interactive data processing
  - Connecting chart selection with 3D visualization

# Afternoon

- Evolving Peacock into a more robust interface for MOOSE modeling and simulation (INL)
- Building Interactive Dashboards for the Beam, Plasma & Accelerator Simulation Toolkit (BLAST) with trame (LBNL)
- Skip the process bloat and run your CFD anywhere with M-Star & trame (M-Star)

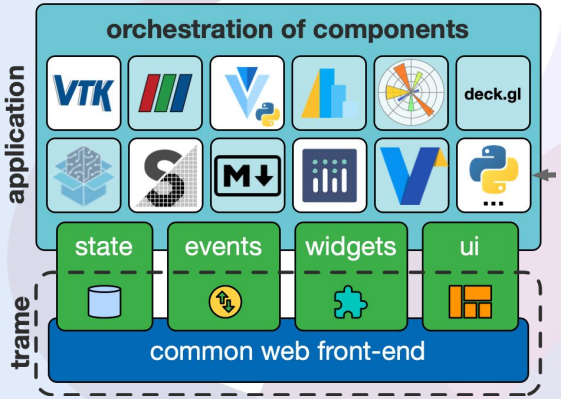
---

- Genomic visualization at scale with ParaView and trame (LANL)
- Real-Time Simulation Control and Visualization through trame (ANL)
- Natural Robustness Testing and Model Explainability for AI Test and Evaluation (Kitware)

# Trame fundamentals

State, Events, Widgets, UI

# What is trame?




- **Simple**

  All the logic and UI definition can be done in plain Python

- **Powerful**

  Python offer scientific and information data visualization with capable data processing (numpy, Plotly, Matplotlib, VTK, ParaView…)

- **Ubiquitous**

  Runs on laptops, desktops, clusters, and the cloud while displaying everywhere (phone, tablet, laptop, workstation)

# What it is not?

- ## Another stateless backend framework

  Take ideas from Dash and Streamlit but go beyond by enabling infinite possibilities while reducing integration/maintenance cost.

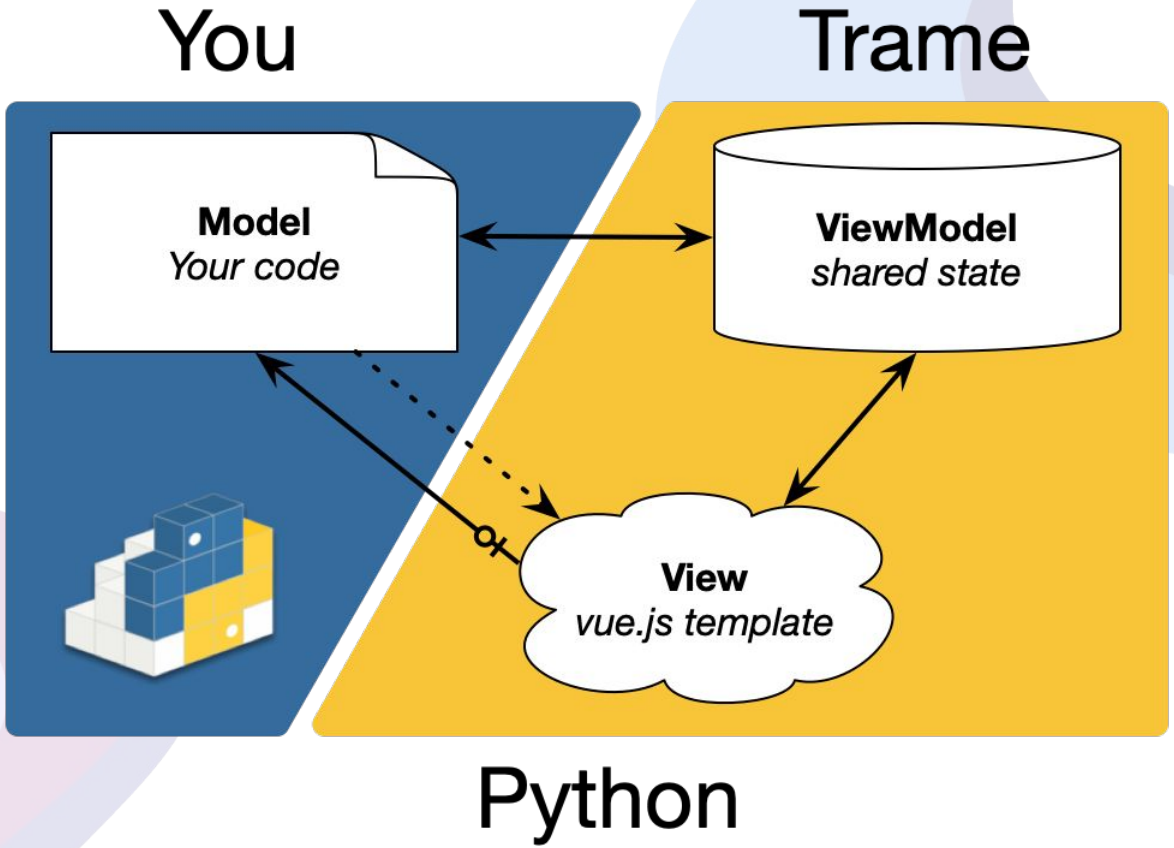
- ## Another visualization/data analytic library

  It does not create something new, but enable interactive interaction with existing libraries like VTK, Plotly, matplotlib and more.

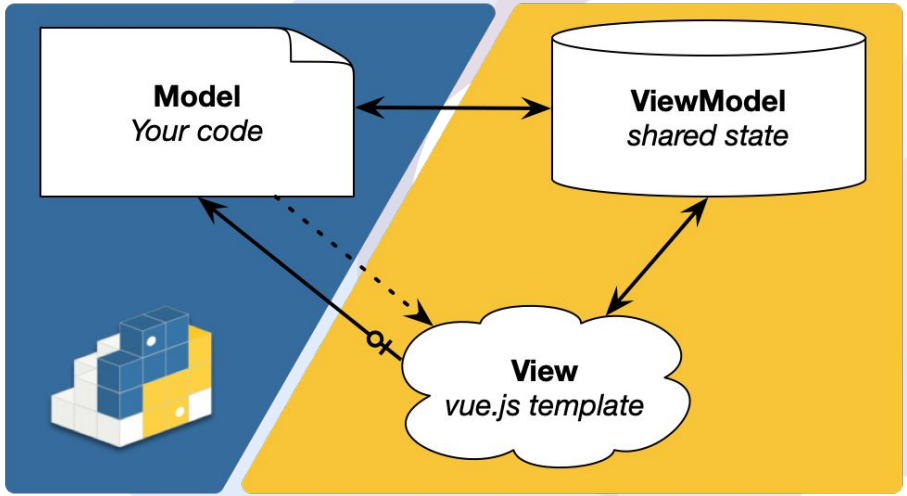
- ## Another JavaScript frontend framework

  Only expect Vue.js components/plugins without any API constraint or trame specificity.

# MVVM Pattern: Model - View - ViewModel



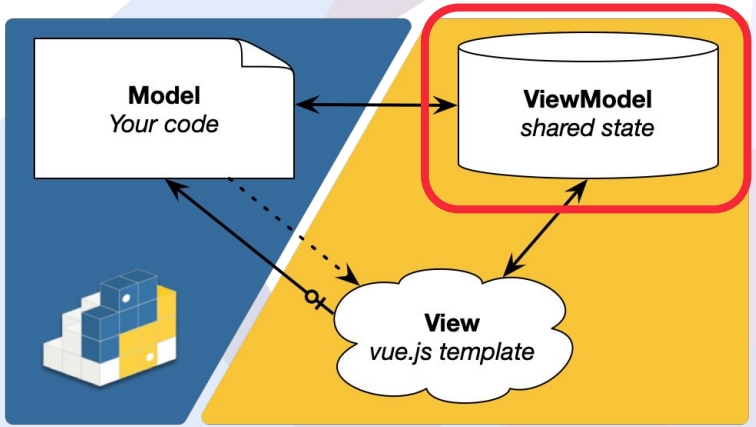
# The split




- **Model**
  - Your business (trame does not care)
- **ViewModel**
  - Reactive data model that drives the presentation layer
  - The ***Model*** can react to changes and modify it
  - The ***View*** reflect its state and can modify it
- **View**
  - Present the ***ViewModel*** into a graphical form
  - The ***Model*** can connect actions to events (click, mouse)

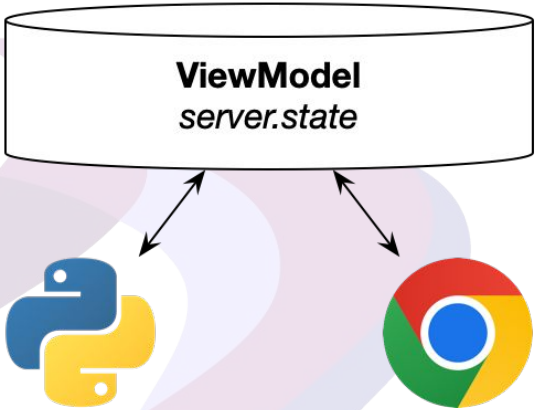
# ViewModel

Dictionary like structure to store data to present in the view. The data needs to be serializable.




**Read/Write**

```
state.a = 1
state["b"] = state.a * 2
assert state.b == state["b"]
```

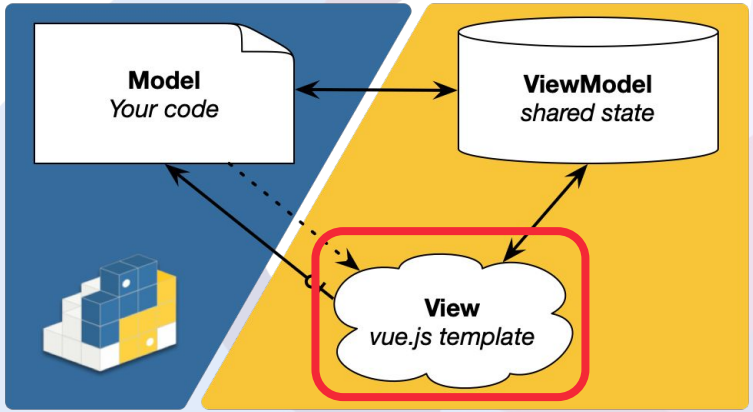


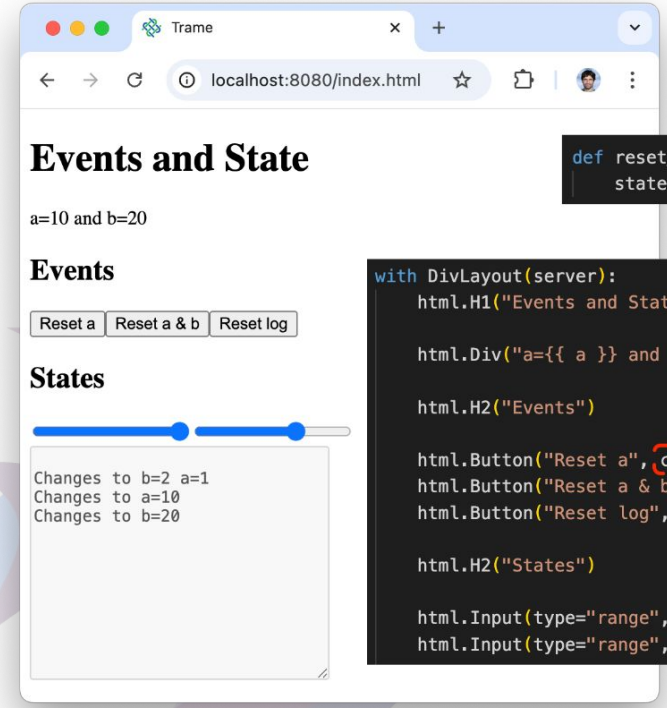

**Reactivity**

```
@state.change("a")
def update_b(a, **_):
    state.b = int(a) * 2


@state.change("a", "b")
def update_log(**_):
    msg = ["\nChanges to"]
    for var_name in state.modified_keys & {"a", "b"}:
        msg.append(f"{var_name}={state[var_name]}")

    state.log += " ".join(msg)
```

# View

Template language (html/vue) in Python to ease data and event binding.




```python
def reset_a():
    state.a = 10
```

```python
with DivLayout(server):
    html.H1("Events and State")

    html.Div("a={{ a }} and b={{ b }}")

    html.H2("Events")

    html.Button("Reset a", click=reset_a)
    html.Button("Reset a & b", click="setAll({ b:2, a:1 })")
    html.Button("Reset log", click="log = ''")

    html.H2("States")

    html.Input(type="range", min=0, max=10, v_model="a")
    html.Input(type="range", min=0, max=30, v_model="b")
```

HTML

```html
<div>
  <h1>
    Events and State
  </h1>
  <div>
    a={{ a }} and b={{ b }}
  </div>
  <h2>
    Events
  </h2>
  <button @click="trigger('trigger__1')">
    Reset a
  </button>
  <button @click="setAll({ b:2, a:1 })">
    Reset a & b
  </button>
  <button @click="log = ''">
    Reset log
  </button>
  <h2>
    States
  </h2>
  <input v-model="a" max="10" min="0" type="range" />
  <input v-model="b" max="30" min="0" type="range" />
  <br />
  <textarea style="width: 15rem;" v-model="log" disabled rows="12" />
</div>
```

# Full application

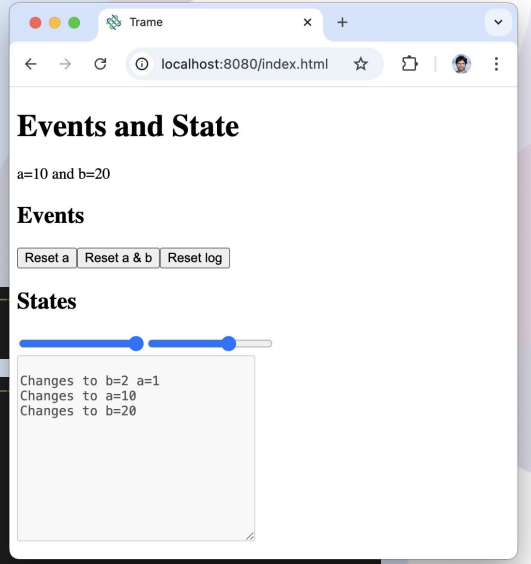
```python
from trame.app import get_server
from trame.widgets import html
from trame.ui.html import DivLayout

# Trame setup ------------------------------------------------------------
server = get_server()
```

```python
# ViewModel --------------------------------------------------------------
state = server.state

# Read/Write
state.a = 1
state["b"] = state.a * 2
assert state.b == state["b"]


# Reactivity
@state.change("a")
def update_b(a, **_):
    state.b = int(a) * 2


@state.change("a", "b")
def update_log(**_):
    msg = ["\nChanges to"]
    for var_name in state.modified_keys & {"a", "b"}:
        msg.append(f"{var_name}={state[var_name]}")

    state.log += " ".join(msg)


@state.change("log")
def trim_log(log, **_):
    lines = log.split("\n")
    if len(lines) > 10:
        state.log = "\n".join(lines[-10:])
```

```python
# Model ------------------------------------------------------------------
def reset_a():
    state.a = 10
```

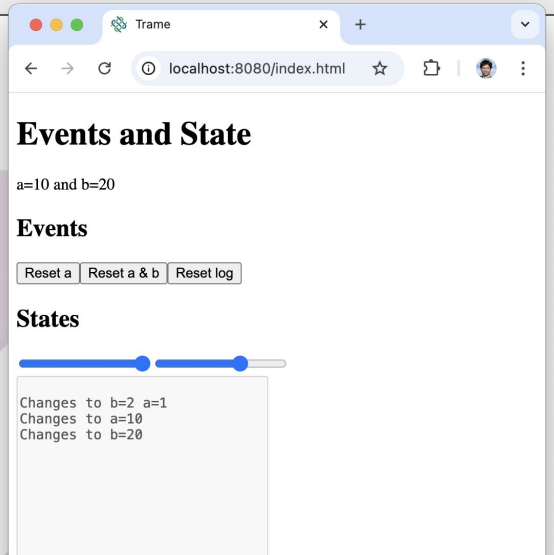
```python
# View -------------------------------------------------------------------
with DivLayout(server):
    html.H1("Events and State")

    html.Div("a={{ a }} and b={{ b }}")

    html.H2("Events")

    html.Button("Reset a", click=reset_a)
    html.Button("Reset a & b", click="setAll({ b:2, a:1 })")
    html.Button("Reset log", click="log = ''")

    html.H2("States")

    html.Input(type="range", min=0, max=10, v_model="a")
    html.Input(type="range", min=0, max=30, v_model="b")

    html.Br()

    html.Textarea(
        v_model=("log", ""),
        disabled=True,
        rows=12,
        style="width: 15rem;",
    )
```

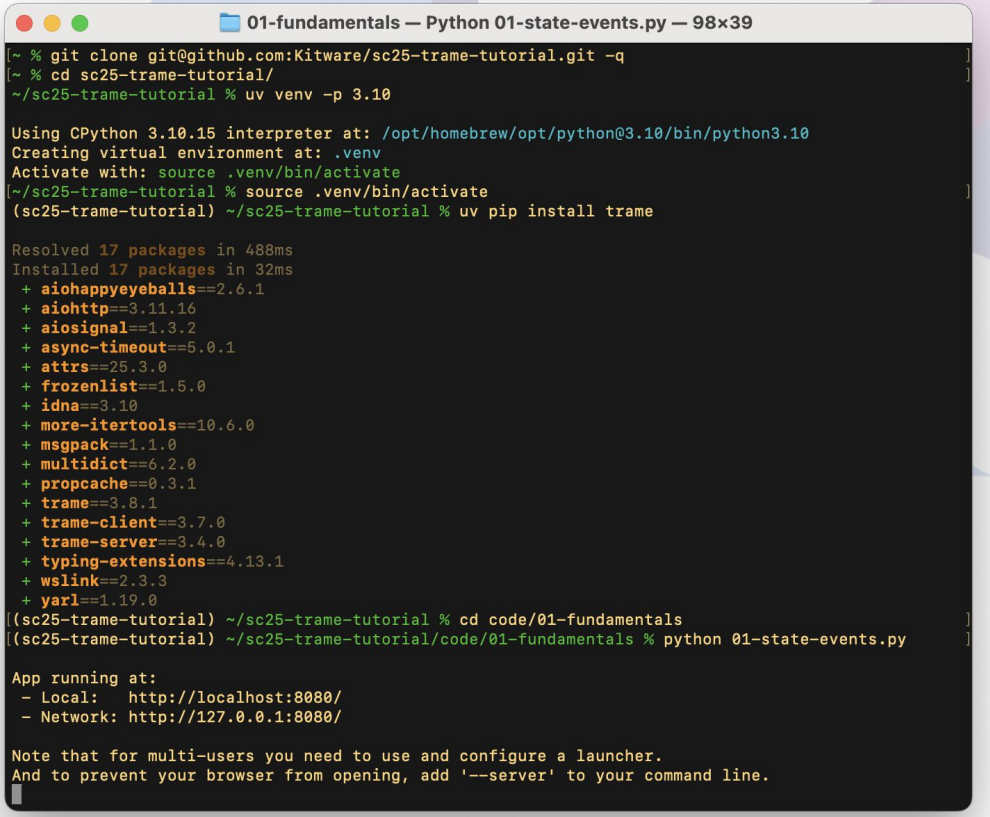
```python
# Start Server -----------------------------------------------------------
server.start()
```



# Hands on (code)

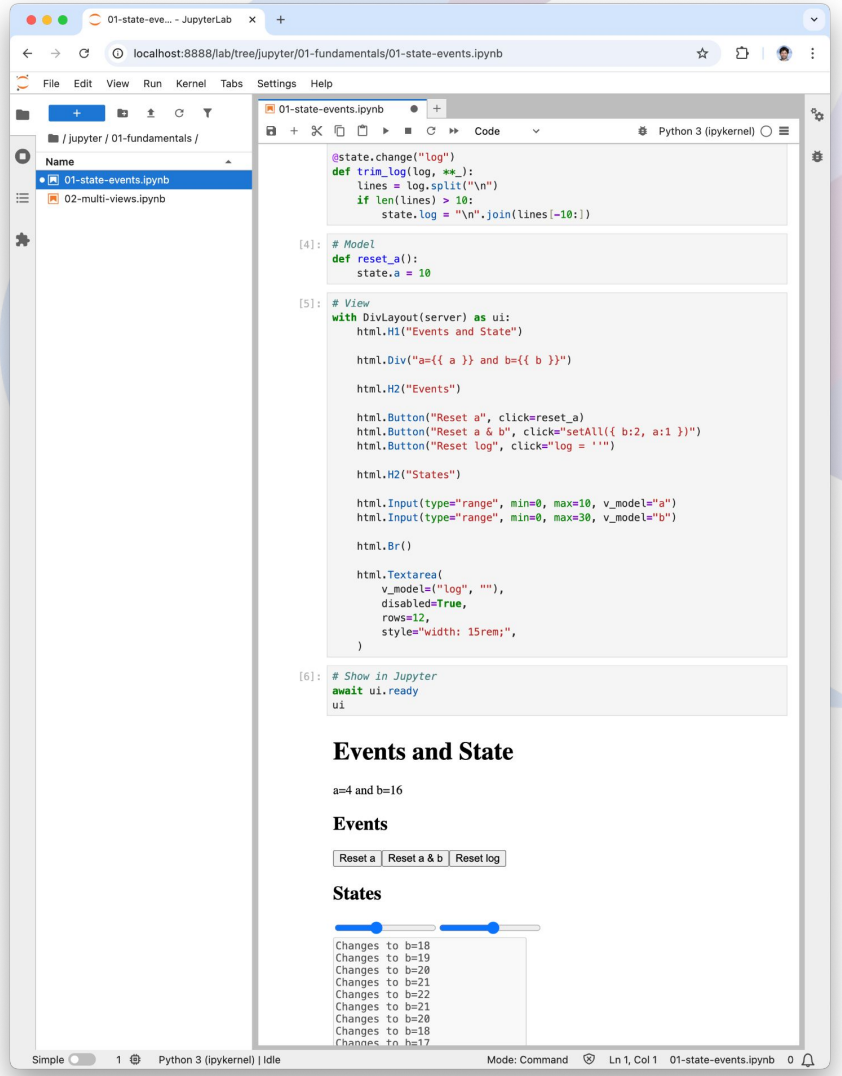
```
$ git clone git@github.com:Kitware/sc25-trame-tutorial.git
$ cd sc25-trame-tutorial/
$ uv venv -p 3.10
$ source .venv/bin/activate
$ uv pip install trame
$ python ./code/01-fundamentals/01-state-events.py
```



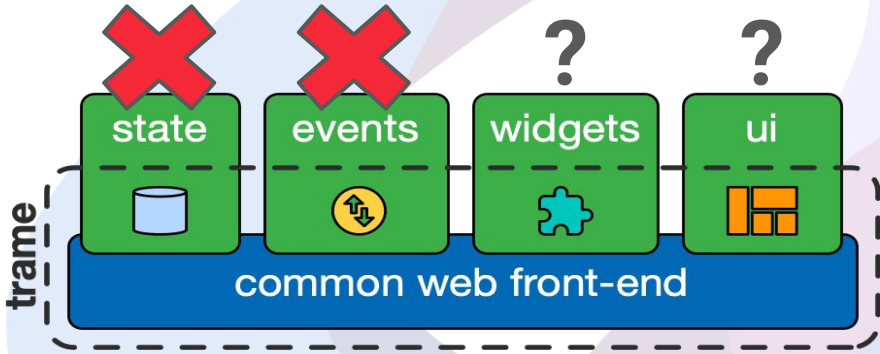



# Hands on (Jupyter)

```
$ git clone git@github.com:Kitware/sc25-trame-tutorial.git
$ cd sc25-trame-tutorial/
$ uv venv -p 3.10
$ source .venv/bin/activate
$ uv pip install trame  jupyterlab
$ jupyter lab
```



# Fundamentals

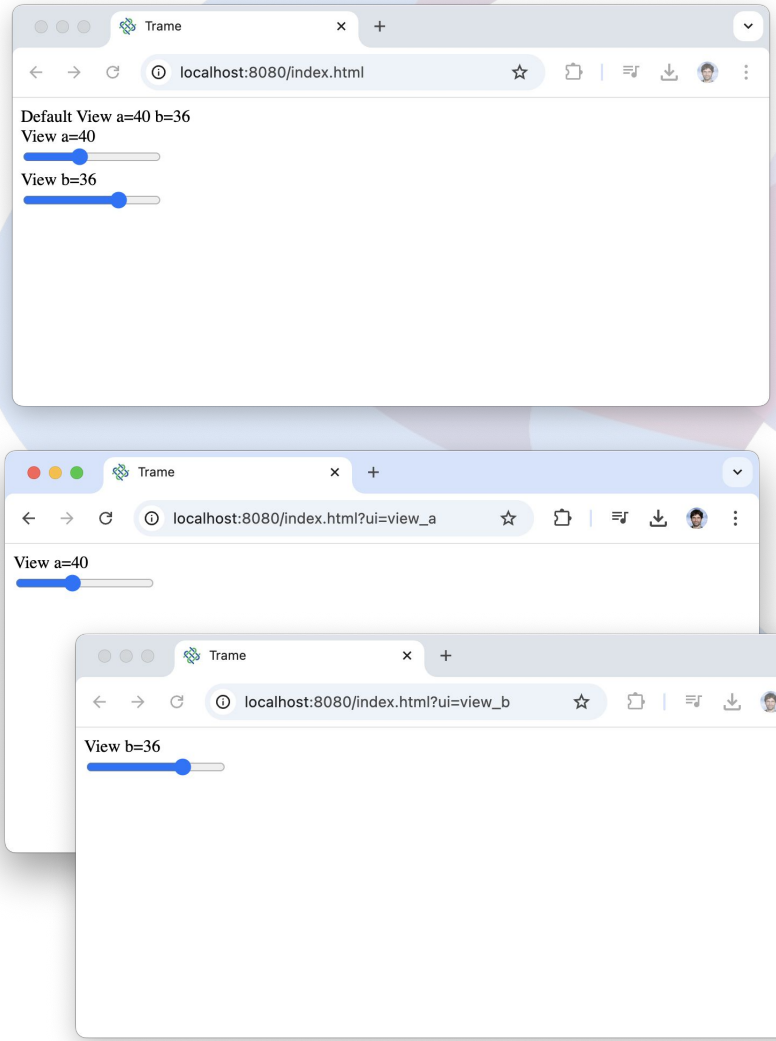



- **Widgets**
  - Any element that compose the UI.
  - This can be an HTML element or a composition of HTML elements.
  - i.e. Slider, Button, Plotly figure, VTK 3D view, xTerm, VS Code Editor, Interactive map…
- **UI**
  - A root widget or a view containing widgets.
  - You can have more than one view/ui.
  - A view/ui can be included into another view using the widget ***trame.widget.client.ServerTemplate***
  - A view can be overridden.

# Hands on (code)

```python
from trame.app import get_server
from trame.widgets import html, client
from trame.ui.html import DivLayout

# Trame setup ------------------------------

server = get_server()

# View -------------------------------------------------------------------

# Main view
with DivLayout(server):  # template_name="main"
    html.Div("Main View a={{a}} b={{b}}")
    client.ServerTemplate(name="view_a")
    client.ServerTemplate(name="view_b")

# view_a
with DivLayout(server, template_name="view_a"):
    html.Div("View a={{a}}")
    html.Input(
        type="range",
        min=0,
        max=100,
        step=1,
        v_model=("a", 0),  # Set a default value of 0 to variable "a"
    )

# view_b
with DivLayout(server, template_name="view_b"):
    html.Div("View b={{b}}")
    html.Input(
        type="range",
        min=0,
        max=50,
        step=2,
        v_model=("b", 2),  # Set a default value of 0 to variable "b"
    )

# Start Server -----------------------------------------------------------

server.start()
```

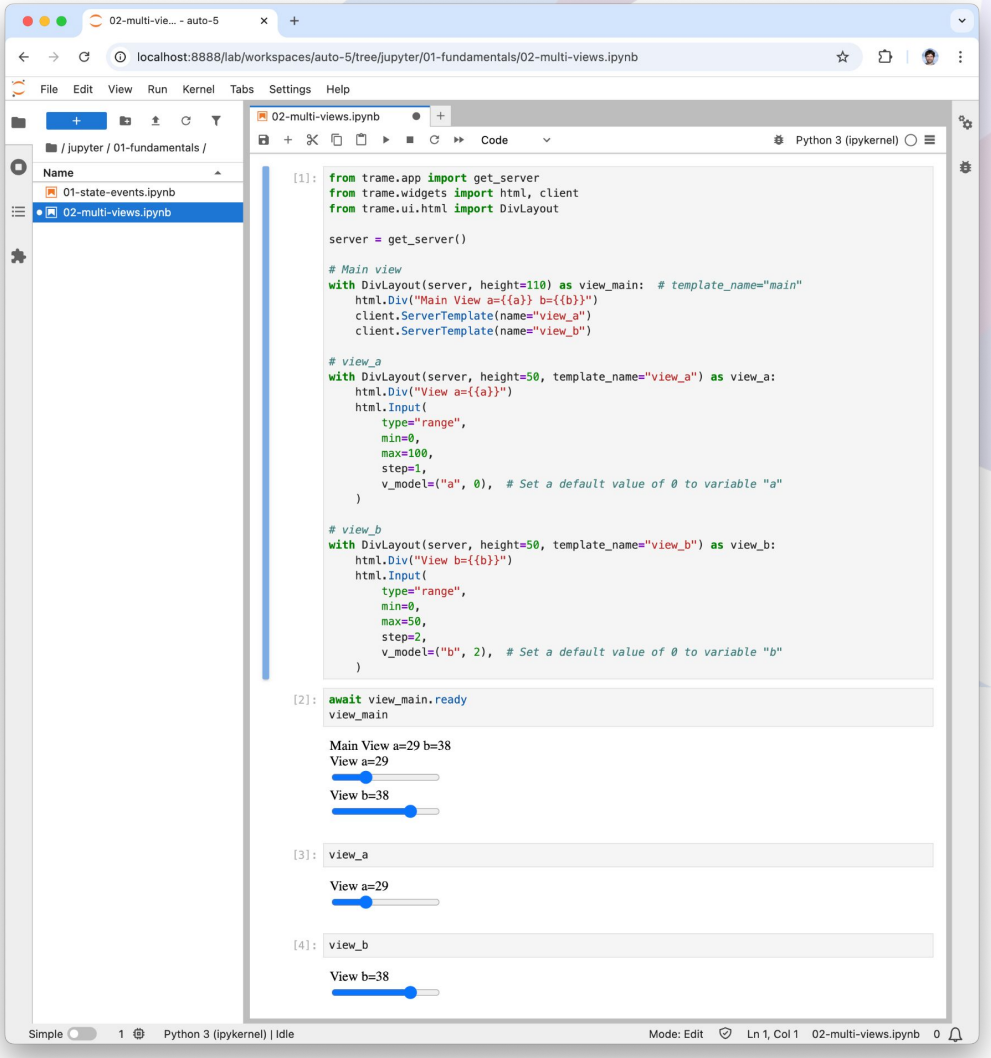
$ python 01-multi-views.py



# Hands on (Jupyter)

$ jupyter lab

=> ./jupyter/01-fundamentals/01-multi-views.ipynb



# Dynamic web interface

## Vue.js templating infrastructure

# Trame and its User Interface

- ## Vue.js and its HTML template

  An approachable, performant and versatile framework for building web user interfaces.

  Vue uses an HTML-based template syntax that allows you to declaratively bind the rendered DOM to the underlying data.

  Under the hood, Vue compiles the templates into optimized JavaScript code.

https://vuejs.org/guide/essentials/template-syntax.html

```python
with DivLayout(server):
    html.H1("Events and State")

    html.Div("a={{ a }} and b={{ b }}")

    html.H2("Events")

    html.Button("Reset a", click=reset_a)
    html.Button("Reset a & b", click="setAll({ b:2, a:1 })")
    html.Button("Reset log", click="log = ''")

    html.H2("States")

    html.Input(type="range", min=0, max=10, v_model="a")
    html.Input(type="range", min=0, max=30, v_model="b")

    html.Br()

    html.Textarea(
        v_model=("log", ""),
        disabled=True,
        rows=12,
        style="width: 15rem;",
    )
```

print(layout_or_widget)

Converted to vue.js template

```vue
<div>
  <h1>
    Events and State
  </h1>
  <div>
    a={{ a }} and b={{ b }}
  </div>
  <h2>
    Events
  </h2>
  <button @click="trigger('trigger__1')">
    Reset a
  </button>
  <button @click="setAll({ b:2, a:1 })">
    Reset a & b
  </button>
  <button @click="log = ''">
    Reset log
  </button>
  <h2>
    States
  </h2>
  <input v-model="a" max="10" min="0" type="range" />
  <input v-model="b" max="30" min="0" type="range" />
  <br />
  <textarea style="width: 15rem;" v-model="log" disabled rows="12" />
</div>
```

# What can you do with vue.js templating?

- Text interpolation
- Raw HTML
- Attribute Bindings
- Directives
- Event handling
- JavaScript Expressions

# Text interpolation

Vue.js

```
<label>Hello {{ name }}</label>
```

Trame (Python)

```
html.Label("Hello {{ name }}")
```

*

```
<SomeWidget
    :label="`Hello ${name}`"
/>
```

```
SomeWidget(
  label=("`Hello ${name}`",),
)
```

*see attribute binding + JavaScript expression

# Raw HTML

Vue.js

```
<span v-html=”rawHTML”></span>
```

Trame (Python)

```
html.Span(v_html=”rawHTML”)
```

```
state.rawHTML = “‘
  <label>Hello SC25</label>
  <p>
    We are <b>happy</b> to have you here today in this tutorial.
  </p>
“‘
```

# Attribute binding HTML

Mustaches cannot be used inside HTML attributes.

Vue.js

```
<SomeWidget
  label="Sample code"
  min="0"

  v-bind:value="a"
  :style="`color: ${ a > 10 ? 'red' : 'green'};`"
  v-bind="{ a:1, b:2, c:3 }"
/>
```

Trame (Python)

```
SomeWidget(
  label="Sample code",
  min=0,

  value=("a",),
  style=("...same as JS...",),
  v_bind=("manyProps", dict(...)),
)
```

* The tuple expression is to indicate we have a JavaScript expression. Moreover, the second argument of the tuple can be used to assign a default value and reserve that variable name.

** Missing attribute mapping can be added using the **__properties=[("py_name", "vue-name:weird")]** as keyword argument.

# Directives

Directives are special attributes with the **v-** prefix. Vue provides a number of built-in directives, including **v-html** and **v-bind** which we have introduced above. Directive attribute values are expected to be single JavaScript expressions (with the exception of **v-for**, **v-on** and **v-slot**, which will be discussed in their respective sections later). A directive's job is to reactively apply updates to the DOM when the value of its expression changes.

Vue.js

```
<SomeWidget
  v-if="a > 10"
  v-show="isVisible"

  v-model="b"
/>
```

Trame (Python)

```
SomeWidget(
  v_if="a > 10",
  v_show="isVisible",

  v_model="b"
)
```

**No need to use the tuple expression as it is ***always*** a JavaScript expression, unless we want to assign a default value (and reserve that variable name).

# Event handling

Vue.js

```
<button
  v-on:click=”doSomething”
>Click Me</button>
```

```
<button
  @click=”doSomething”
>Click Me</button>
```

Trame (Python)

```
html.Button(
  “Click Me”,
  click=”doSomething”,
)
```

* Trame do not show any difference between an attribute or an event.

** Modifiers can be added to the event (**stop**, **prevent…**) [**@click.prevent=** / **click_prevent=**]

*** Events mapping can be missing in trame but could be added using the **__events=[(“click_prevent_stop”, “click.prevent.stop”)]** keyword arg.

# JavaScript expressions

- ## String templating

```
<b>static {{ dynamic }}</b>
```

```
<widget  :label="`static ${dynamic}`" />
```

```
<b>
  static
  {{ message.split(' ').reverse().join(' ') }}
</b>
```

- ## Ternary evaluation

```
<b>static {{ ok ? 'yes' : 'no' }}</b>
```

```
<widget  :label="`static ${ok ? 'yes' : 'no'}`" />
```

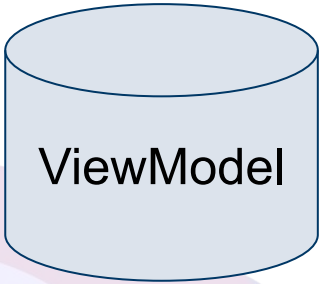
- ## Conditions

```
a && b || c || d === e || f == g || h > 3 || array.includes(search) || obj?.a?.c?.[6]
```

# State handling cheat sheet

```python
from trame.app import get_server
server = get_server()
state = server.state
```

```python
@state.change("a", "b", "c")
def change_detected(a, b, c, d, **kwargs):
    pass

state.a = 10
print(f"a={state.a}")

def update_var(name, value)
    print(f"Before: {name}={state[name]}")
    state[name] = value
    print(f"After: {name}={state[name]}")
```



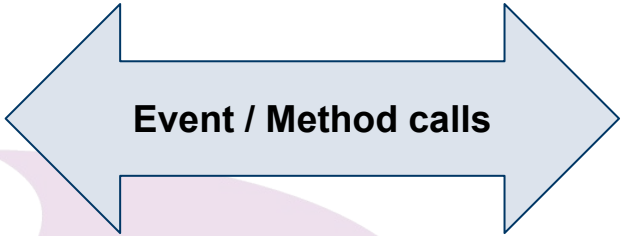

JS

```
Widget(key="welcome")          : Use string 'welcome'
Widget(key=("welcome",))       : Evaluate expression 'welcome'
Widget(key=("welcome","Hello")) : Evaluate expression 'welcome' and initialize it to "Hello"
```

# Event handling cheat sheet

```python
from trame.app import get_server
server = get_server()
ctrl = server.controller
```

```python
# @ctrl.set("hello_alias")
def hello(*args, **kwargs):
    print(args, kwargs)

@ctrl.add("hello_alias")
def hello2(*args, **kwargs):
    print("v2:", args, kwargs)

ctrl.hello_alias = hello
ctrl.hello_alias('arg_0', key='yes')
```

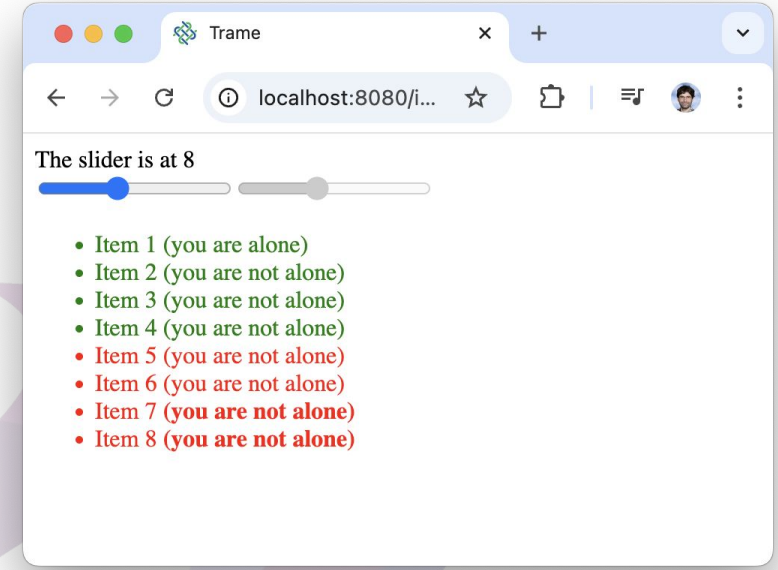



JS

```
VBtn(click="js_fn")                          : Call the JS code
VBtn(click=fn)                               : Call the function with no args
VBtn(click=(fn, "[1, $event, 'hello']"))     : Call the function like fn(1, event, "hello")
VBtn(click=(fn, "[$event]", "{ x: 5 }"))     : Call the function like fn(event, x=5)
```

# Hands on (code)

$ python ./code/02-dynamic-ui/01-directives.py

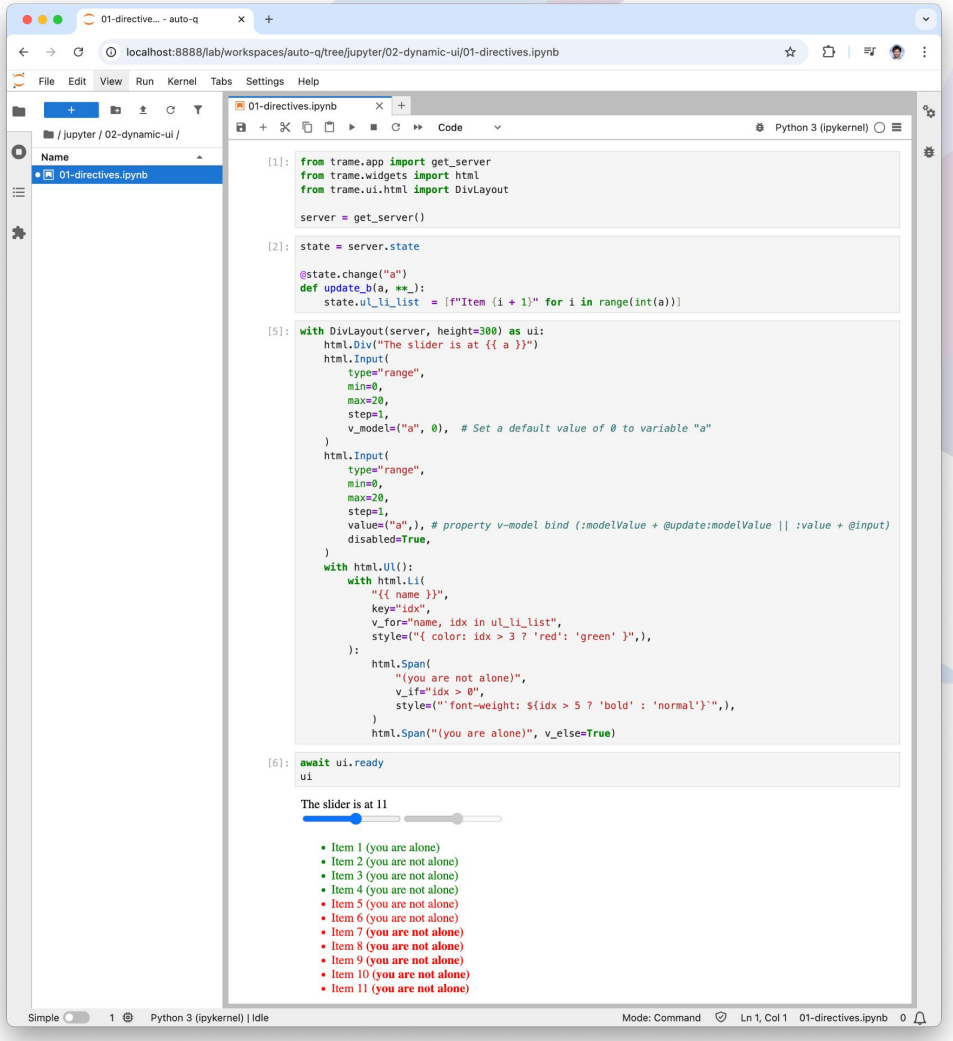



```python
# Reactivity
@state.change("a")
def update_b(a, **_):
    state.ul_li_list  = [f"Item {i + 1}" for i in range(int(a))]

# View ----------------------------------------------------------------------

# Main view
with DivLayout(server):
    html.Div("The slider is at {{ a }}")
    html.Input(
        type="range",
        min=0,
        max=20,
        step=1,
        v_model=("a", 0),  # Set a default value of 0 to variable "a"
    )
    html.Input(
        type="range",
        min=0,
        max=20,
        step=1,
        value=("a",), # property v-model bind (:modelValue + @update:modelValue
        disabled=True,
    )
    with html.Ul():
        with html.Li(
            "{{ name }}",
            key="idx",
            v_for="name, idx in ul_li_list",
            style=("{ color: idx > 3 ? 'red': 'green' }",),
        ):
            html.Span(
                "(you are not alone)",
                v_if="idx > 0",
                style=("`font-weight: ${idx > 5 ? 'bold' : 'normal'}`",),
            )
            html.Span("(you are alone)", v_else=True)
```

# Hands on (Jupyter)

$ jupyter lab

=> ./jupyter/02-dynamic-ui/01-directives.ipynb

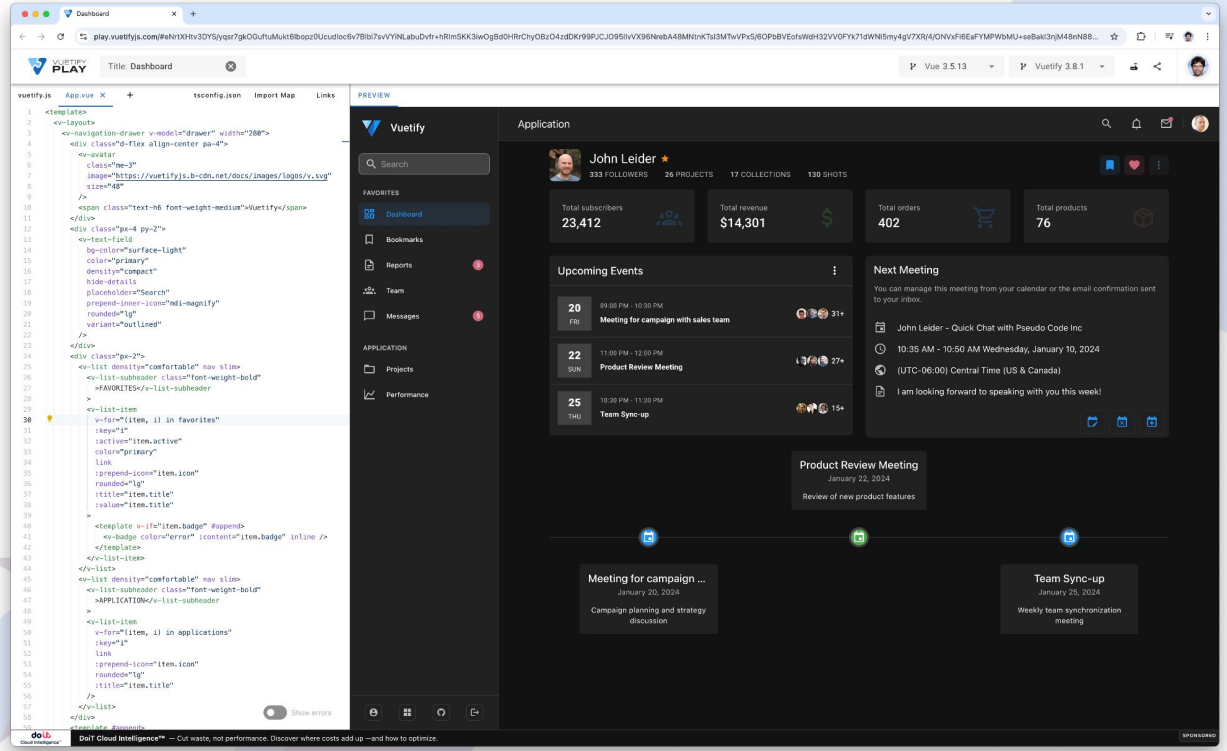


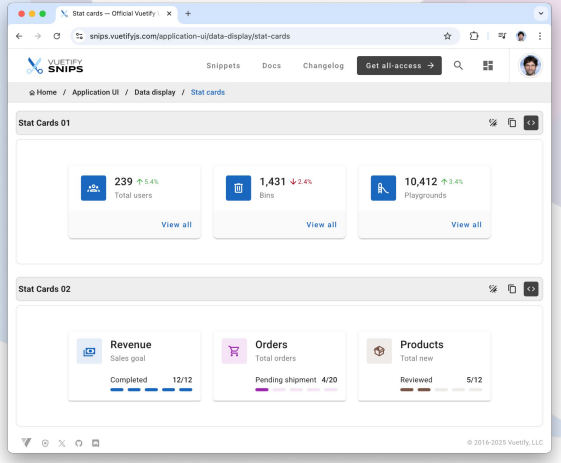
# Vuetify for beautiful UI

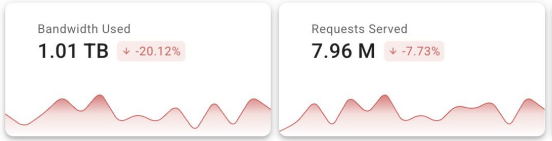
From playground to trame user interface

https://vuetifyjs.com/

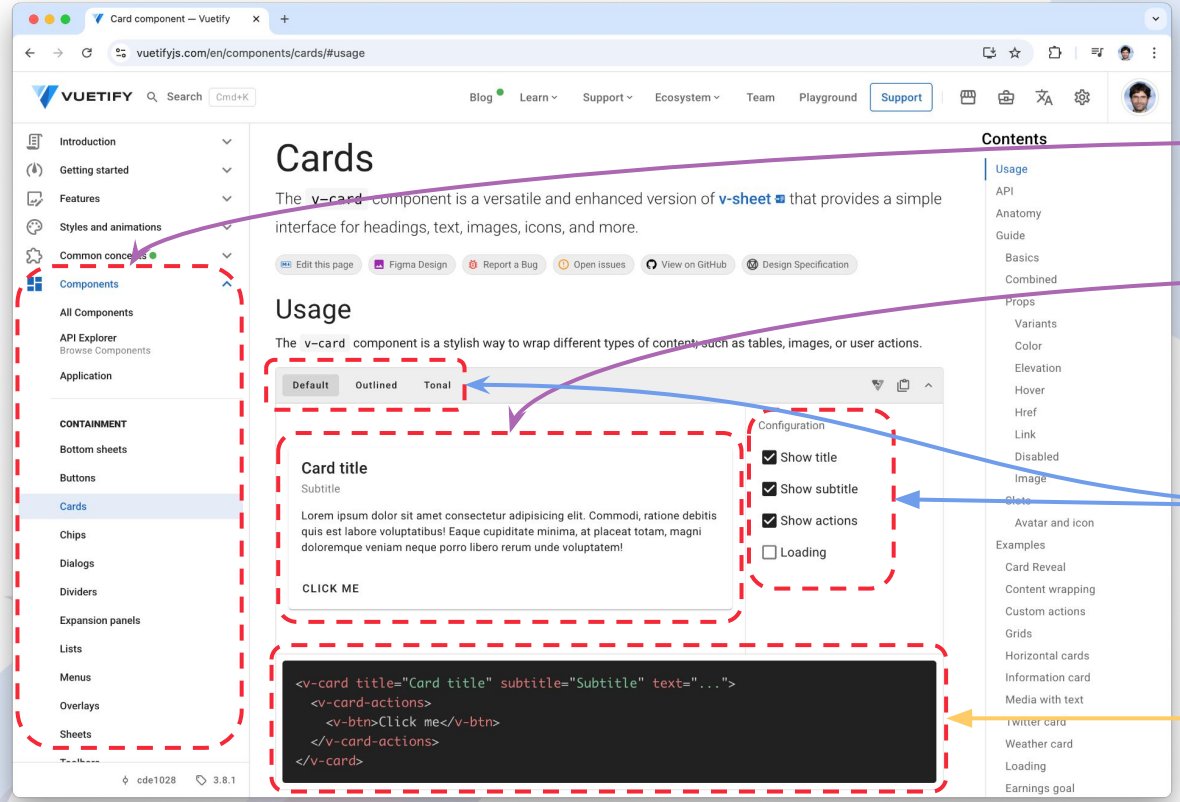
# From basic to pretty with Vuetify



# Explore components and options




Select component

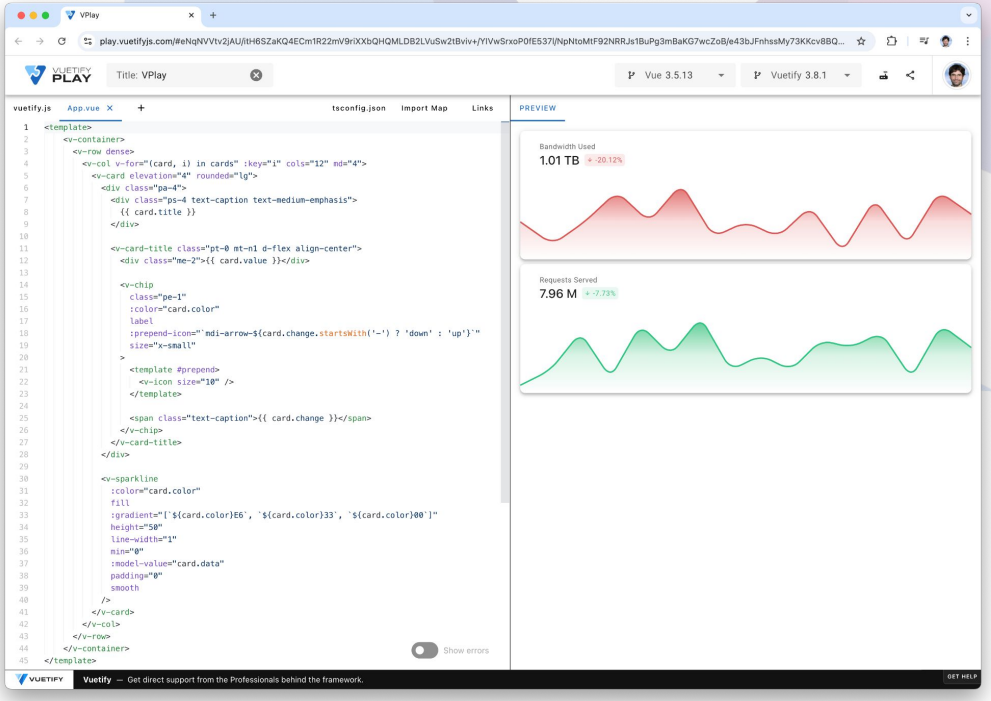
Preview

Options

Template code

# Using the playground to design UI

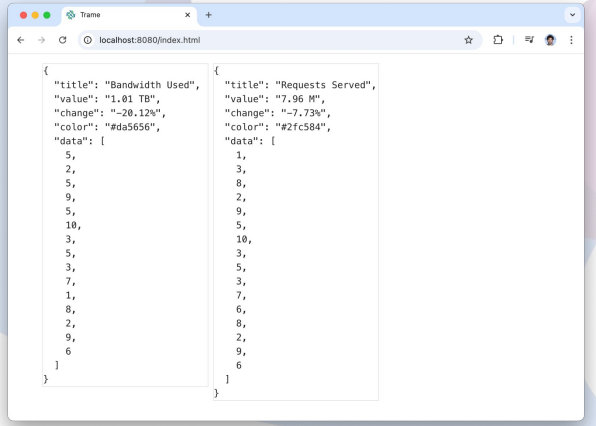
- **Hands on in**
  - Open **./code/03-vuetify/App.vue**
  - Paste it inside the playground
  - Try editing or adding other widgets from vuetify

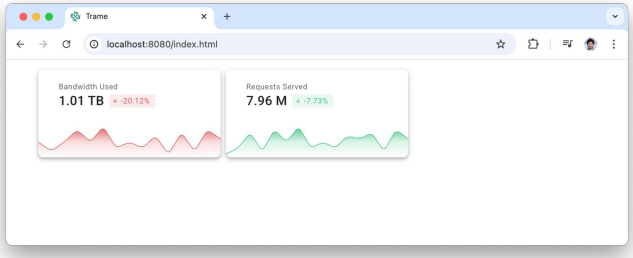


# From the playground to trame

- **Hands on in**
  - **uv pip install trame-vuetify**
  - Edit **./code/03-vuetify/01-static-card-edit.py**
  - Reproduce your playground template in Python

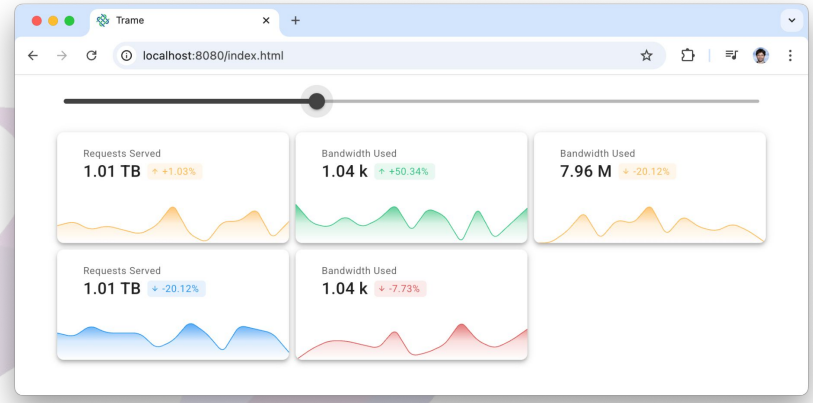
```
state.cards = [
    {
        "title": "Bandwidth Used",
        "value": "1.01 TB",
        "change": "-20.12%",
        "color": "#da5656",
        "data": [5, 2, 5, 9, 5, 10, 3, 5, 3, 7, 1, 8, 2, 9, 6],
    },
    {
        "title": "Requests Served",
        "value": "7.96 M",
        "change": "-7.73%",
        "color": "#2fc584",
        "data": [1, 3, 8, 2, 9, 5, 10, 3, 5, 3, 7, 6, 8, 2, 9, 6],
    },
]

# View ----------------------------------------------------------------------
with VAppLayout(server):
    with v3.VContainer():
        with v3.VRow(dense=True):
            with v3.VCol(v_for="(card, i) in cards", key="i", cols=12, md=4):
                html.Pre("{{ JSON.stringify(card, null, 2) }}", classes="border-thin")
                # Use App.vue template content and convert it to Python
```

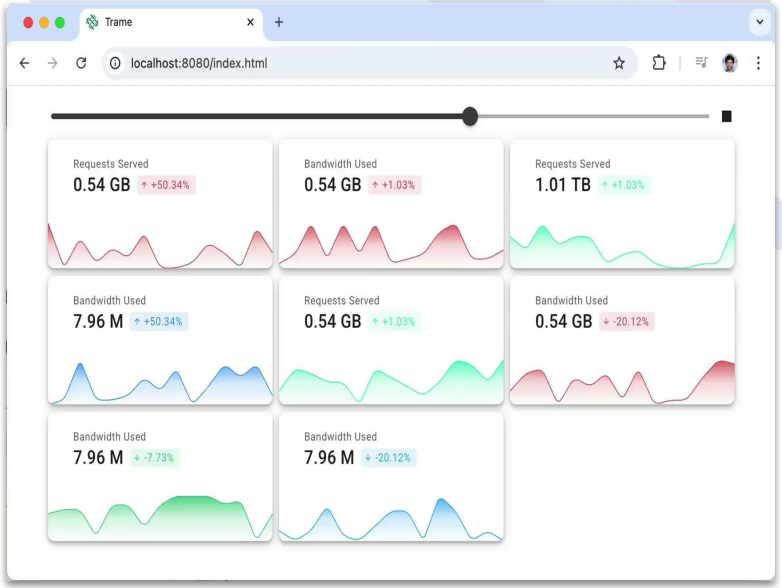


# Beautiful dynamic UI

- **Hands on in**
  - Run **./code/03-vuetify/03-dynamic-cards.py**
  - Run **./code/03-vuetify/04-animation.py**



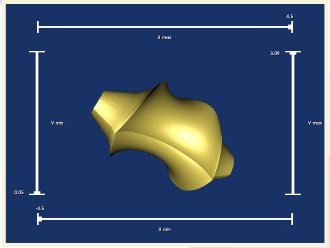




COFFEE
BREAK

# 3D Visualization

## Using VTK or ParaView for your visualization

# Let’s start with a VTK example



- **Install more libraries (VTK)**

  **$ uv pip install vtk trame-vtk**

- **Use existing code from VTK example Website**

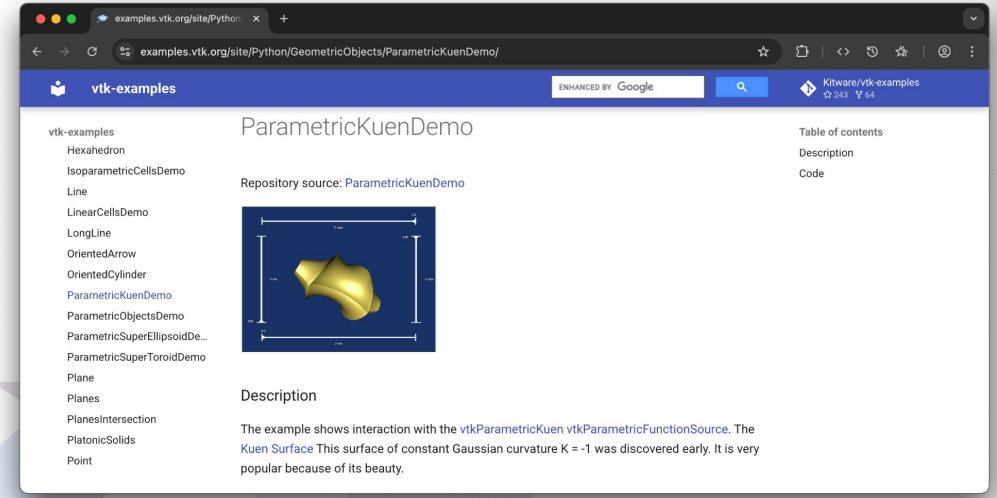
  https://examples.vtk.org/site/Python/GeometricObjects/ParametricKuenDemo/

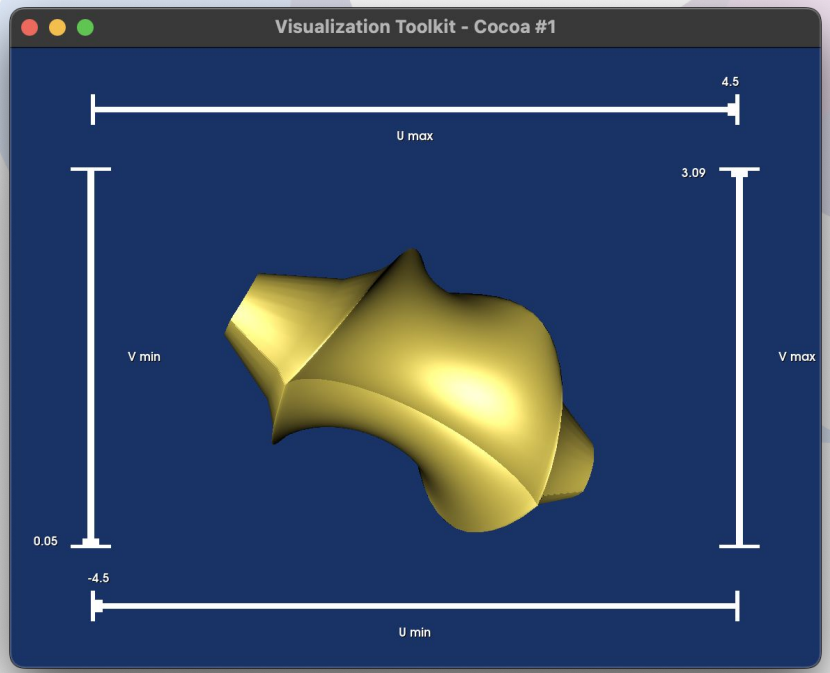
- **Convert it to a trame application with Vuetify**

  Use ***trame.ui.SinglePageLayout*** + ***widgets.vuetify3.VRangeSlider***.

# Hands on (code) - vtk example

```
$ uv pip install  vtk  trame-vtk
$ python ./code/04-visualization-3d/01-vtk-example.py
```

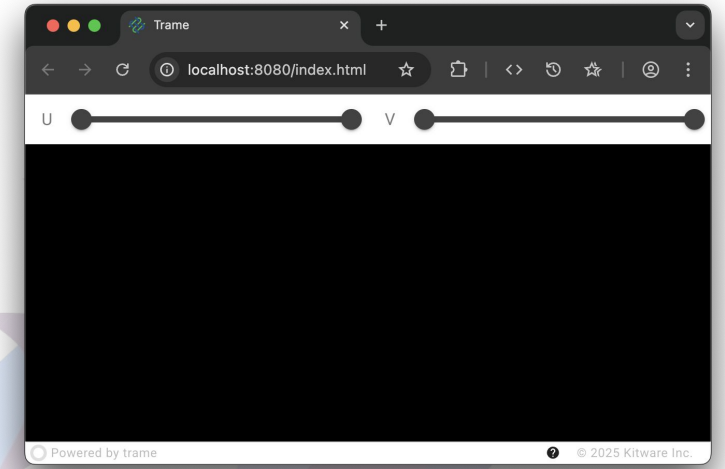


# Hands on (code) - make it trame

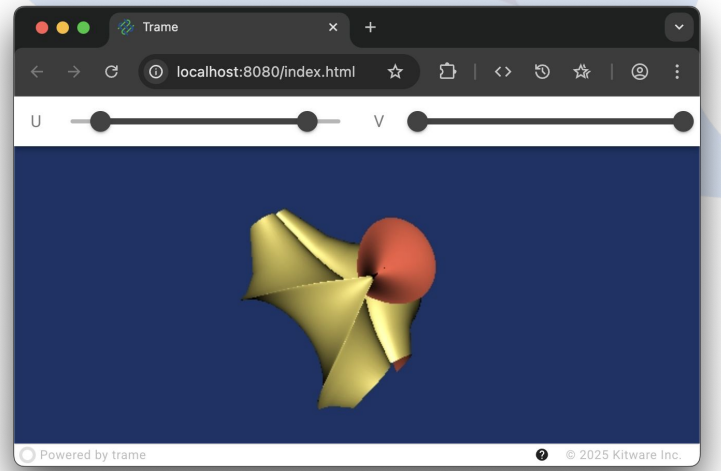
Edit **./code/04-visualization-3d/02-vtk-trame-edit.py**

```
$ python ./code/04-visualization-3d/02-vtk-trame-edit.py
$ python ./code/04-visualization-3d/03-vtk-trame-solution.py
```



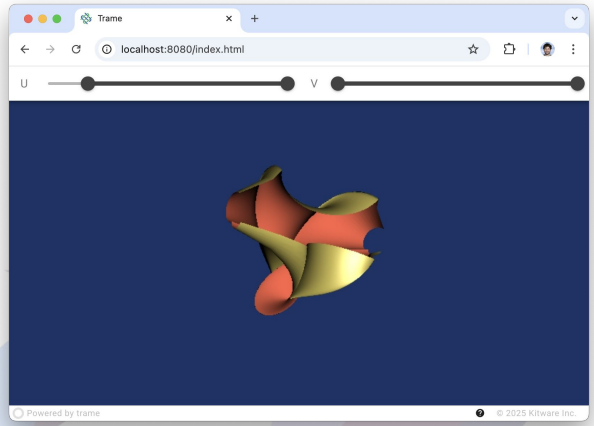

45% smaller than pure VTK

From 250 lines to 137



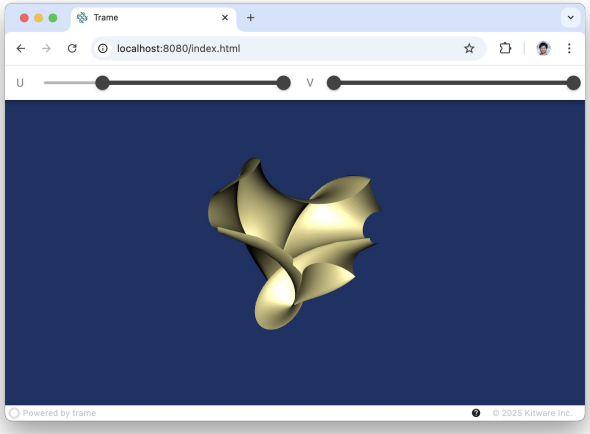
# Hands on (code) - remote/local rendering

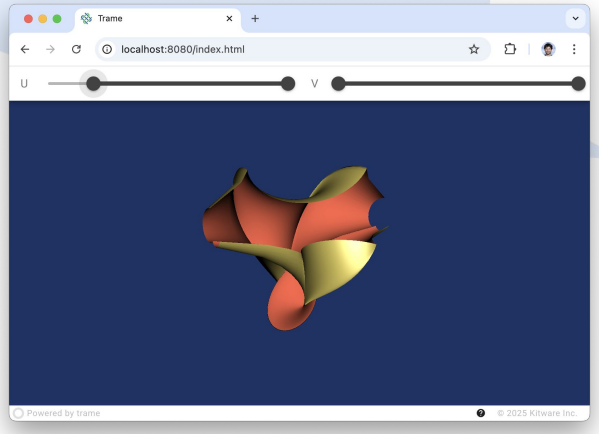
```
$ uv pip install  trame-vtklocal

$ python ./code/04-visualization-3d/04-vtk-trame-remote.py
$ python ./code/04-visualization-3d/04-vtk-trame-local-js.py
$ python ./code/04-visualization-3d/04-vtk-trame-local-wasm.py
```




VTK - Remote

vtk.js - Local




VTK.wasm - Local

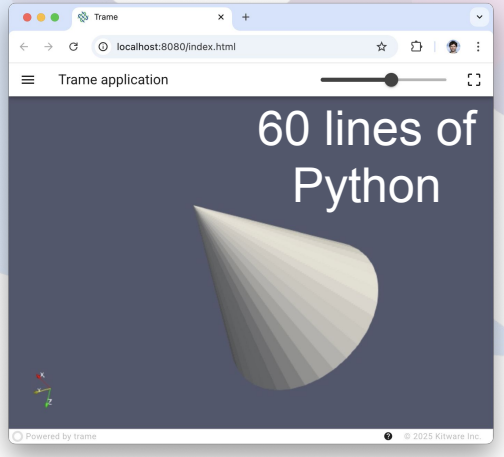
# Hands on (code) - ParaView

- **Need a venv compatible with ParaView**

  $ uv venv -p 3.10

  $ source .venv/bin/activate

  $ pip install trame trame-vtk trame-vuetify




- **Running the application**

  $ pvpython --venv .venv ./code/04-visualization-3d/05-paraview-cone.py

# Hands on (code) - ParaView with MPI

- **Running the application**

  **$ pvpython --venv .venv** ./code/04-visualization-3d/06-paraview-mpi.py

  **$ mpiexec -n 4 pvbatch --venv .venv** ./code/04-visualization-3d/06-paraview-mpi.py


!!! mac ARM - have a hanging issue with ParaView/MPI

# Charts

## Using matplotlib, plotly or altair

# Coming soon

# Interactive data processing

## Better data exploration with interactive processing

# Coming soon


LUNCH
BREAK

# Peacock 2.0

Logan Harbour (Idaho National Lab)

# Coming soon

# Peacock an interface to the moose solver

- Self documenting input deck
  - Language server available
  - CLI can provide runtime information on available node/model
- Graphical interface
  - Qt based ui, so local only
  - Web based via trame
    - Allow local, remote, hpc setup
- 3 graphical panels
  - Input deck
  - Run simulation
  - Post-process

# Trame implementation - Input deck

- Leverage Simput for generating the GUI forms for each node. The implementation query the command line for gathering node options and documentation.
-

# Trame implementation - Simulation run

- Leverage xterm to display the output of the simulation run

# Trame implementation - Post processing

- Leverage ParaView and some specific ui

# What’s next

The trame implementation is still work in progress to showcase that it is possbile

# VeraCore

Another viewer for nuclear simulation results

That work was achieved in 80h work hours

Dashboard for exploring hdf5 result file

# Trame provide a nice way to

Create simulation tools for pre-processing

Create viewer/dashboard to analyse results

Allow execution of application, locally, remotely, within jupyter and on HPC

Enable creation of bespoke/focus applications

# Hands on

VS Code usage?

XTerm

=> remote shell / Python

# Building Interactive Dashboards for the Beam, Plasma & Accelerator Simulation Toolkit (BLAST) with trame

Axel Huebl (Lawrence Berkeley National Lab)

# Coming soon

# Skip the process bloat and run your CFD anywhere with M-Star & trame

Brian DeVincentis (M-Star)

# Coming soon


COFFEE
BREAK

# Genomic visualization at scale with ParaView and trame

David Rogers (Los Alamos National Lab)

# Coming soon

# Real-Time Simulation Control and Visualization through trame

Victor Mateevitsi (Argonne National Lab)

# Coming soon

# Natural Robustness Testing and Model Explainability for AI Test and Evaluation

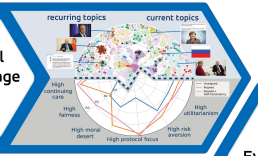
Sebastien Jourdain (Kitware)

Disclaimer: This material is based upon effort sponsored by the U.S. Government under the Tradewind Project Agreement. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government.

# Kitware is known for ParaView, but we do so much more...

CMake

Multimodal Large Language Models



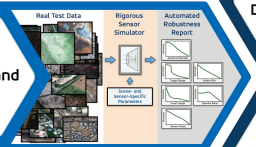

Generative AI

Dataset Collection and Annotation

Cyber Physical Systems (CPS)

Computational Imaging


AI Test, Evaluation, and Assurance




Activity, Threat, and Anomaly Detection

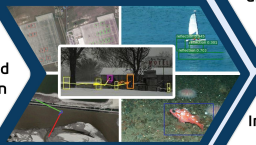
Tracking

*60+ team members*
*24 PhDs*
*40+ cleared*
*40+ active projects*
*Vision-ary since 2007*

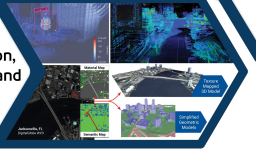
kitware
Delivering Innovation

Combatting Disinformation

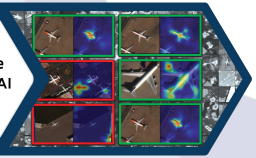
Object Detection and Classification



3D Reconstruction, Point Clouds, and Odometry



Explainable and Ethical AI



Interactive AI and Human Machine Teaming

## Open Source Toolkits

VIAME
XAITK
GeoWATCH
TeleSculptor
KWIVER
SMQTK
SMQTK
LidarView

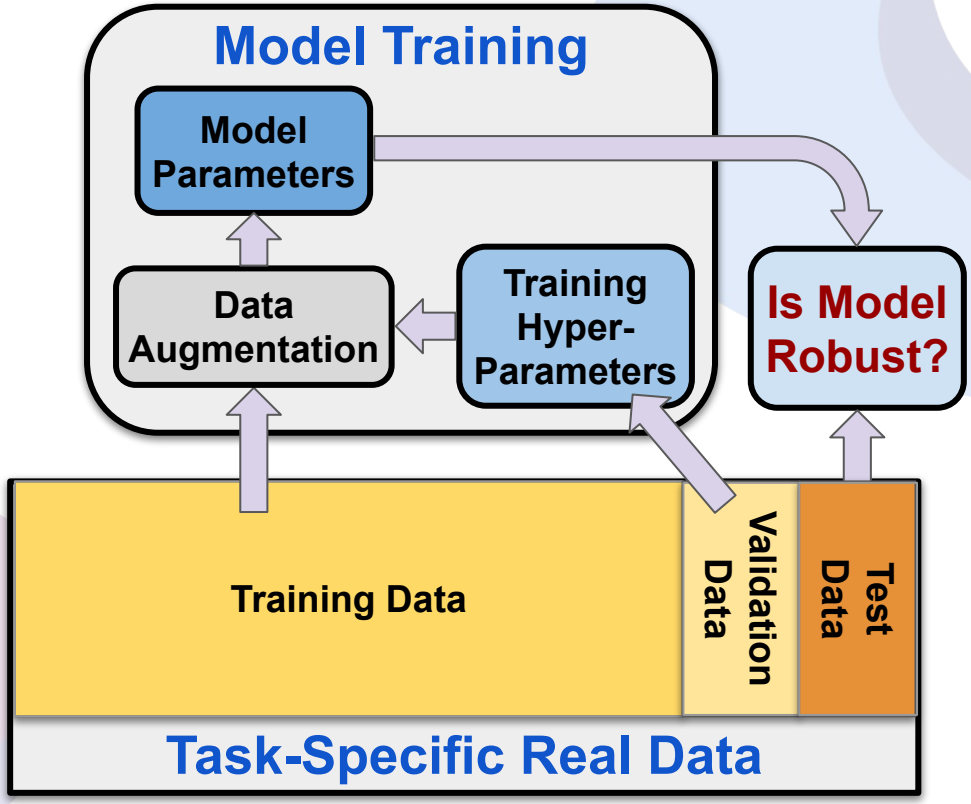
Space
Air
Land
Sea Surface
Undersea

# Under CDAO JATIC (Joint AI Test Infrastructure Capability)

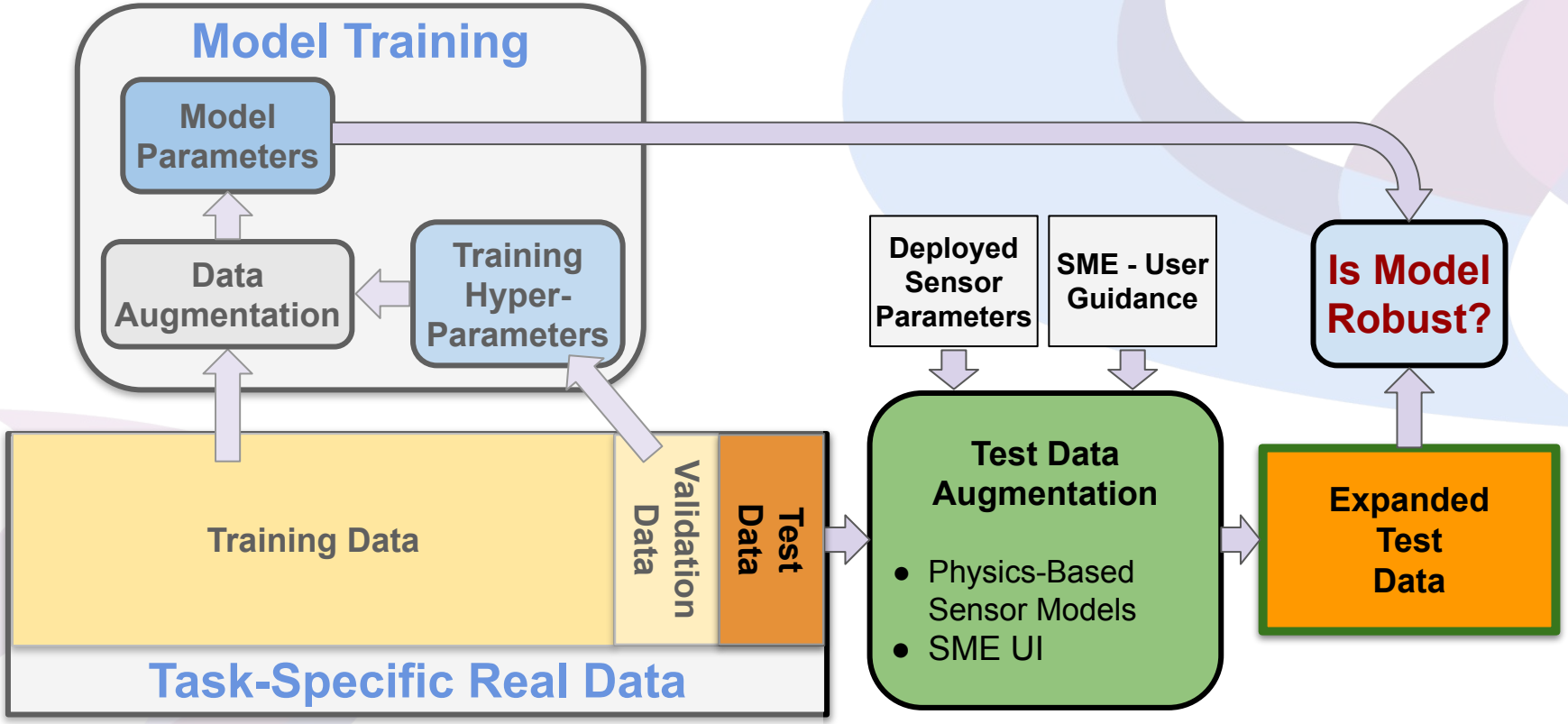
## Kitware is focusing on 2 Open Source projects

- **Natural Robustness Toolkit (NRTK)**
  - Evaluate models against physics-based, operationally-realistic perturbations
- **Explainable AI Toolkit (XAITK)**
  - Generate visual saliency maps on AI predictions using black-box and white-box techniques

# Existing AI Test and Evaluation Approach



# Robust AI Test and Evaluation Approach



# Natural Robustness Toolkit (NRTK)

JATIC
Joint AI Test Infrastructure Capability

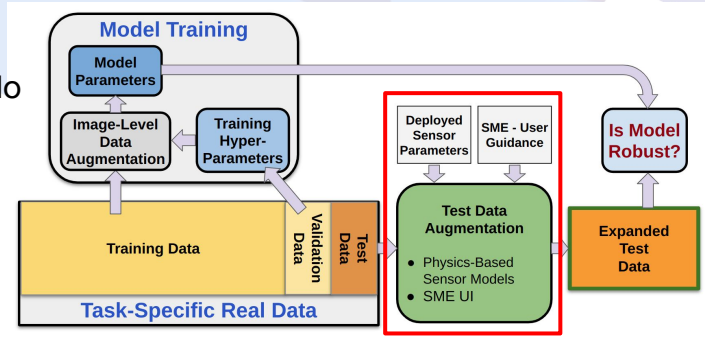
**Natural Robustness Toolkit (NRTK)** is a platform for generating validated, sensor-specific perturbations and transformations used to evaluate the robustness of computer vision models

## Background

- Computer vision models are sensitive to data distribution shifts (either due to synthetic image perturbations or those naturally occurring in real data)
- Existing image augmentation libraries (e.g. imgaug, albumentations) do not cover physics-based, sensor-specific perturbations that are relevant to operational data

## NRTK Use Cases

- Assess robustness of computer vision models trained on satellite images to changes in different sensor parameters (e.g. focal length, aperture, pixel pitch, etc)
- Develop validated sets of sensor perturbation parameters and expanded datasets for comprehensive model test and evaluation (T&E)



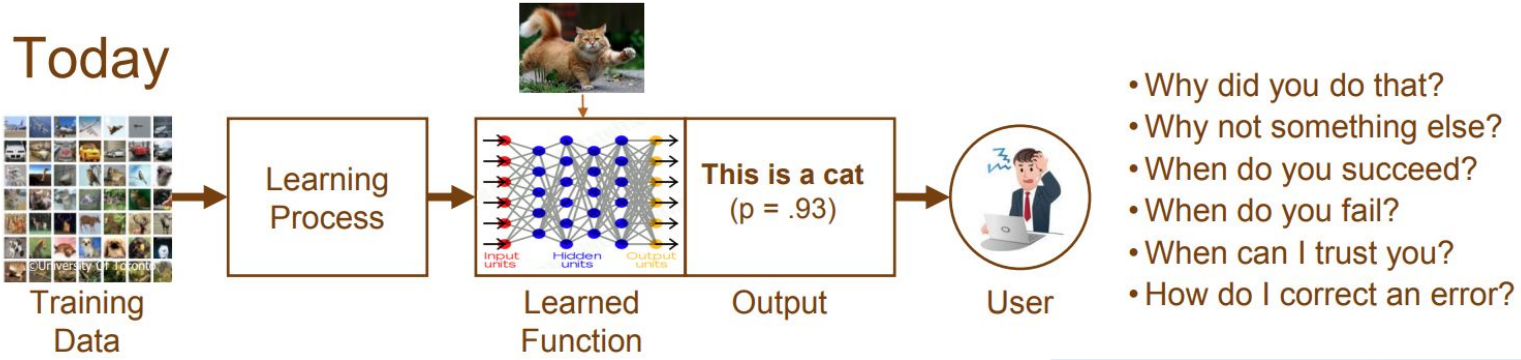

***NRTK** expands existing datasets to enable more robust test and evaluation of models*

https://github.com/Kitware/nrtk
https://github.com/Kitware/nrtk-explorer

kitware

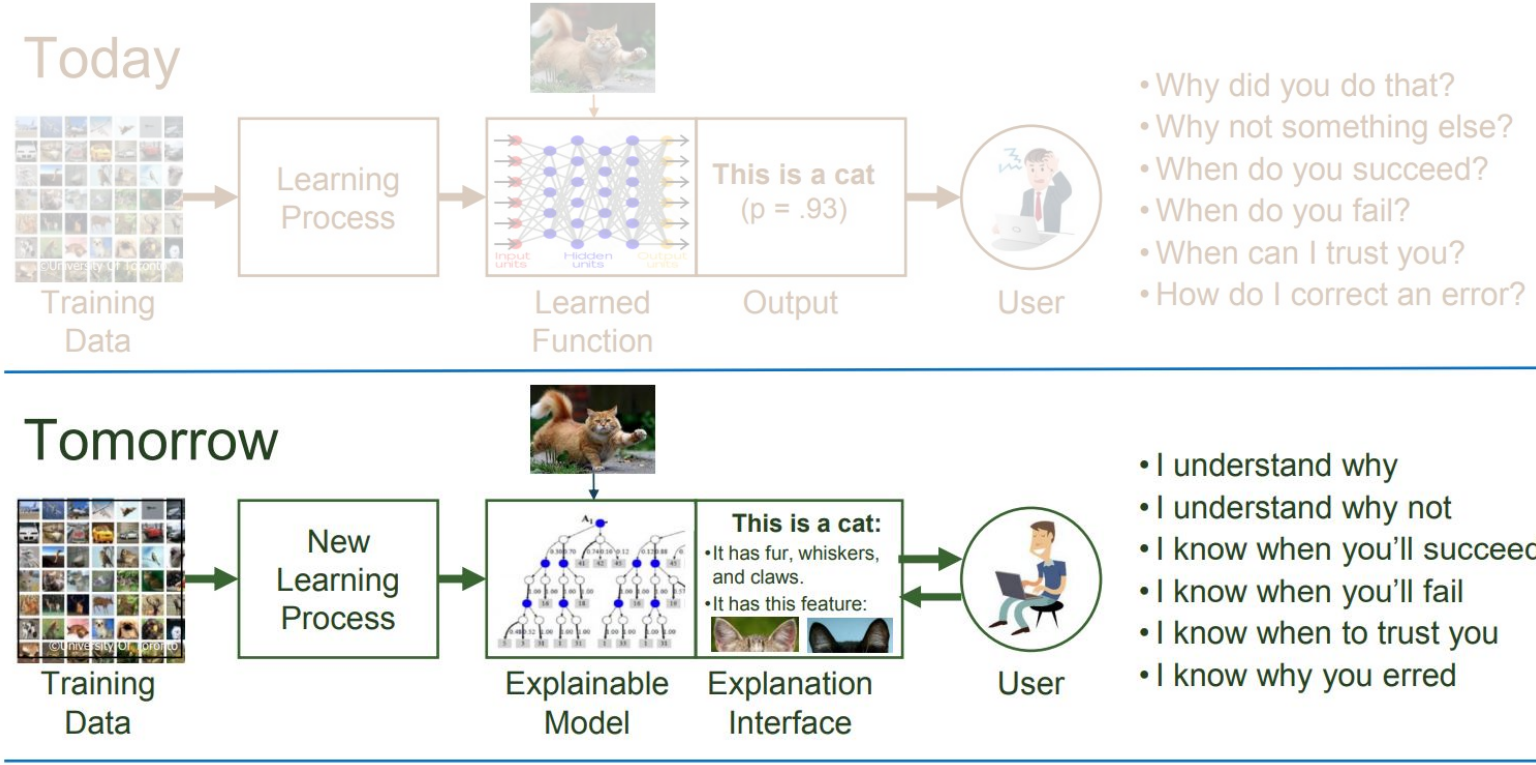
NRTK
Natural Robustness Toolkit

# Explainable AI (XAI) ?




https://www.darpa.mil/program/explainable-artificial-intelligence

# Explainable AI (XAI) !

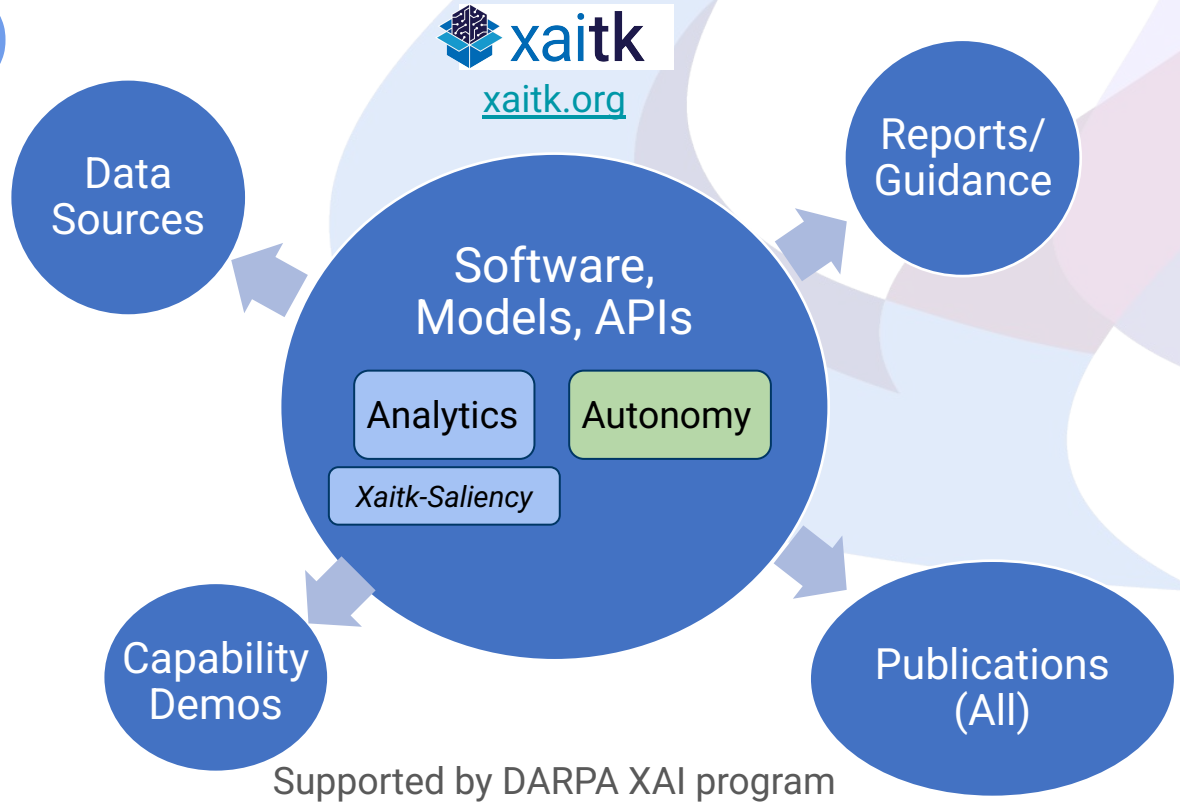



https://www.darpa.mil/program/explainable-artificial-intelligence

# XAI Toolkit (XAITK)

**Highlights:**

1. Searchable repository of XAI tools and resources
2. Easy to follow guide for what to use for a given task
3. Software framework built for analytics (*Xaitk-Saliency*)
4. Open-source, permissively licensed core with flexible licensing for plugins

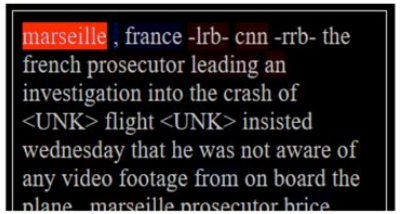



Supported by DARPA XAI program

Hu, B, Tunison, P, Vasu, B, Menon, N, Collins, R, Hoogs, A. XAITK: The explainable AI toolkit. *Applied AI Letters*. 2021; e40. doi:10.1002/ail2.40

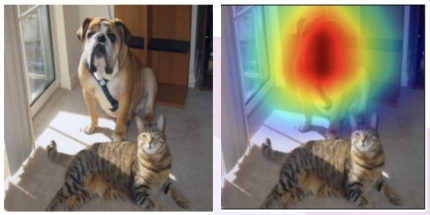
# Saliency Maps as Explanations

Saliency maps are a form of *visual XAI*, overlaying the input with (typically) a heatmap to spatially indicate which areas of the input the AI found "important".




Saliency map for an algorithm on automatic text summarization (Tuckey et al.)

SBSM result showing where the image on the right is similar to the one on the left (Dong et al.)



Grad-CAM result for "Dog" (Selvaraju et al.)

RISE result for importance of "Sheep" (Petsiuk et al.)

https://xaitk-saliency.readthedocs.io/en/latest/introduction.html

# Explainable AI Toolkit (XAI Toolkit)

JATIC
Joint AI Test Infrastructure Capability

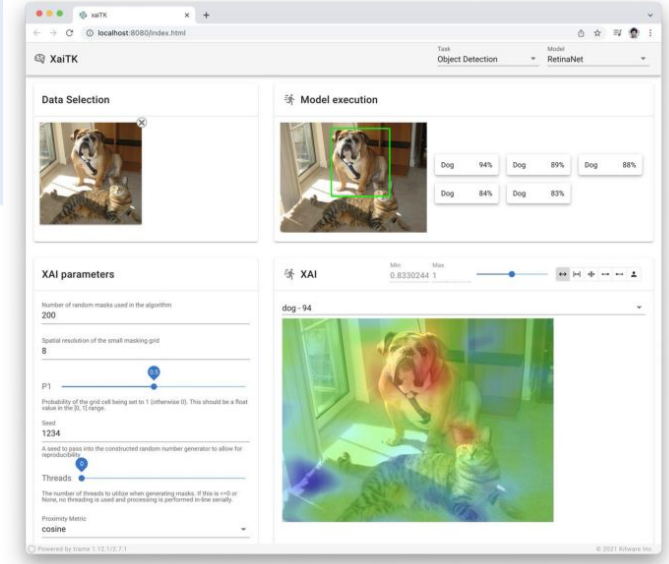
**XAI Toolkit (XAITK)** is a platform for visual saliency algorithms that enable human understanding of complex machine learning algorithms

## Background

- DARPA XAI created a suite of ML techniques that enabled human users to understand and appropriately trust the results of AI systems
- Collaborators included UC Berkeley, UCLA, Carnegie Mellon, UT Dallas, Texas A&M, Rutgers, Oregon State, SRI, Raytheon BBN, and IHMC
- XAITK combines the techniques and research performed throughout the DARPA XAI program into a unified and extensible package

## XAITK Use Cases

- Establish justified confidence in AI models, addressing *DoD Ethical AI Principles* of Traceability and Reliability
- Enable feature understanding and analysis for black-box AI models
- Provide analysis of model failure modes and edge cases




https://www.darpa.mil/program/explainable-artificial-intelligence
https://xaitk.org
https://github.com/XAITK/xaitk-saliency
https://github.com/XAITK/xaitk-saliency-web-demo

kitware

xaitk

# But where is trame?

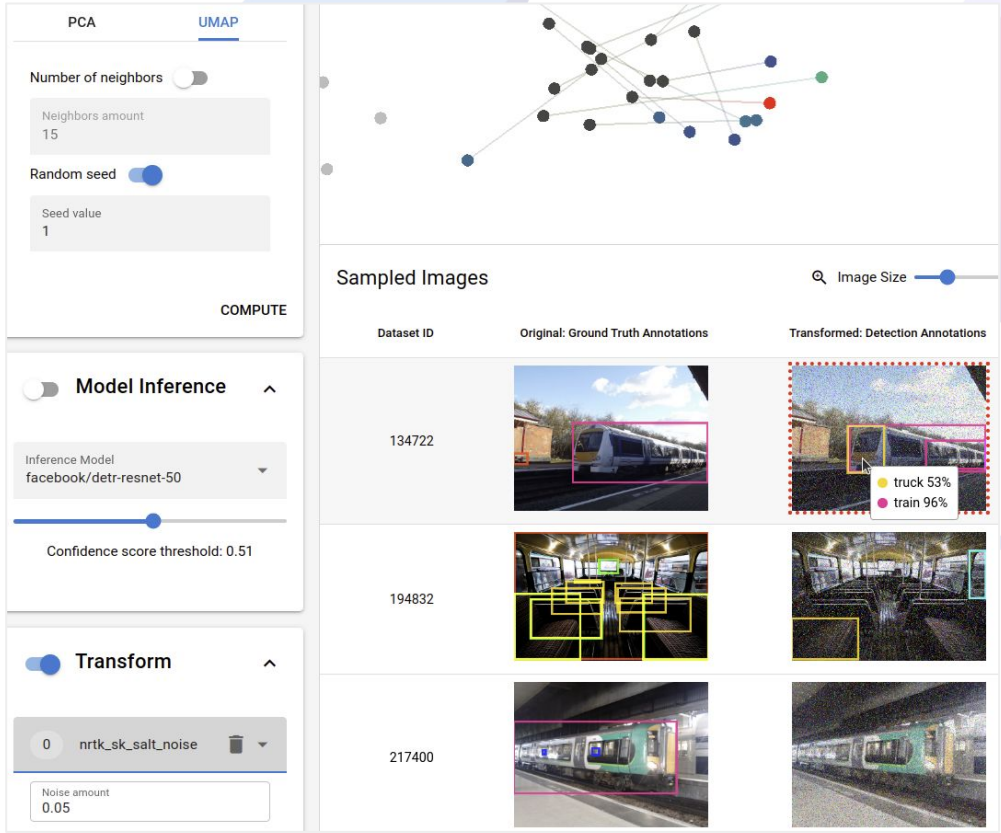
- By default a toolkit is never appealing unless you are a geek and you only work on a terminal.
- But what if you want to interactively try algorithms on your data?
- Trame is ideal for creating flexible GUI that offload the heavy lifting to other libraries.
- With trame we were able to create easy to install tools that can run locally (data privacy) or in cloud services.
- The XAITK demo application only took 1 week to create!
- The NRTK application is quite advanced due to the required generic approach toward perturbator and coco dataset import/export.

# NRTK Explorer App

https://github.com/Kitware/nrtk-explorer

**Dataset and Model Analysis Workflow**

- Load models and datasets
- Group images in low-dimensional embedding space
- Filter by ground truth classifications
- Apply image perturbations
- Evaluate Object Detection and Classification inference results




NRTK
Natural Robustness Toolkit

https://github.com/Kitware/nrtk

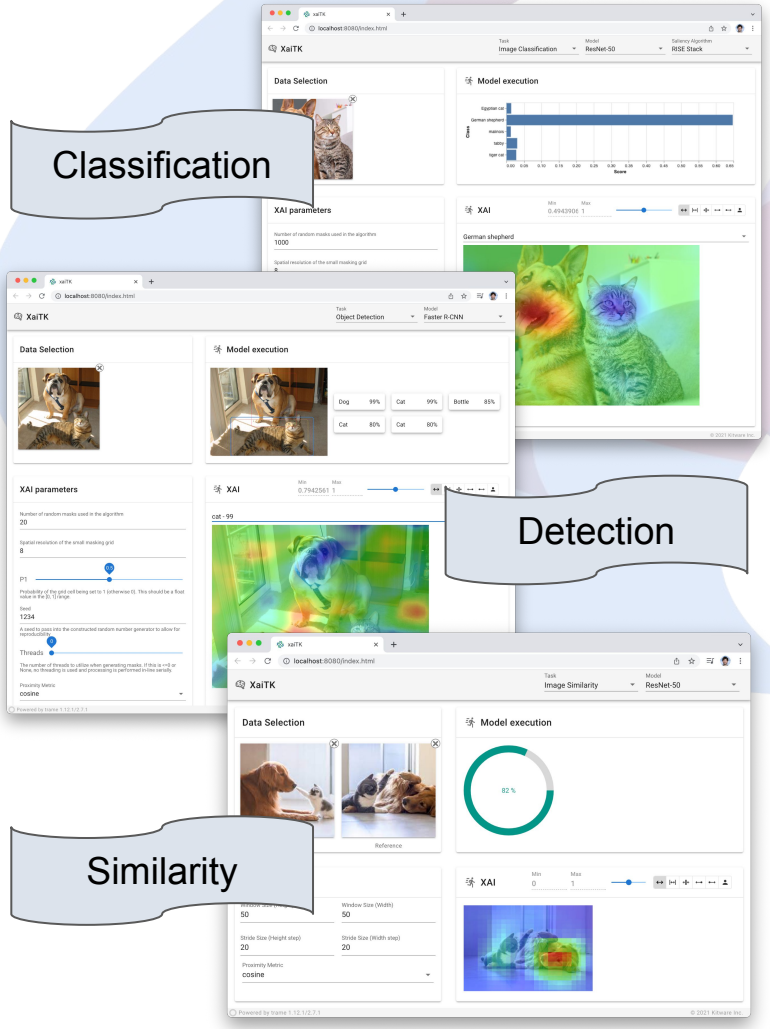
https://github.com/Kitware/nrtk-explorer

# XAITK App

https://github.com/XAITK/xaitk-saliency-web-demo

**Model Explanation Workflow**

- Select task, AI model, XAI algo
- Upload image(s)
- Configure XAI algo
- Interactive XAI visualization




xaitk

https://github.com/XAITK/xaitk-saliency

https://github.com/xaitk/xaitk-saliency-web-demo

# This effort is lead by Brian Hu at Kitware.

brian.hu@kitware.com

Disclaimer: This material is based upon effort sponsored by the U.S. Government under the Tradewind Project Agreement. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government.


THANK
YOU
FOR
YOUR
ATTENTION