A pythonic journey from scientific data visualization to broadly data science

Wenjun (Walter) Kou
Gastroenterology
BDSD 2020
Outline

• Brief intro: → research route, applications showcase
  • Scientific modeling: python for large-data visualization
  • Clinical data science: python for ML and DL

• Python → How to make it fast?
  • Efficient pipeline:
  • Python vs C (++)

• Python → Visualization analysis
  • Python vs javascript
  • Python way for ‘client-cloud’ visualization
Outline

• Brief intro: research route, applications showcase
  • Scientific modeling: python for large-data visualization
  • Clinical data science: python for ML and DL

• Python → How to make it fast?
  • Efficient pipeline:
  • Python vs C (++)

• Python → Visualization analysis
  • Python vs javascript
  • Python way for ‘client-cloud’ visualization
Background: Esophageal transport

Fig. 1. Esophagus

Fig. 2. Esophagus cross-section
Muscle
Mucosa (folded)

Fig. 3. Myoarchitecture of the esophagus
Circular muscle (CM)
Longitudinal muscle (LM)
Richard J. Gilbert et. al, Cell Tissue Res 2008
Simulation model: Esophageal transport
Clinical device-based modeling $\rightarrow$ data + simulation

Clinical input: geometry of lumen

Outcome: pressure, velocity, wall stress, stiffness

Endoflip

Demo case of axial velocity

Demo outcomes
Produce movies? Python

Data visualization: VisiT

ViSiT data on **NU Quest:**

Slow pipeline:
- Download to local → open GUI → case by case:
  - VERY SLOW FOR MANY CASES

Better pipeline:
- Python script + Bash run (templating)
- Generate image (not movies?) on **Quest**
- ffmpeg to merge image to movie
- Append to submitted job

Large-scale model → Data source

- C++ parallel library (IBAMR)
- Parallel computing on **NU Quest**
- ~5GB data per case
Outline

• Brief intro: research route, applications showcase
  • Scientific modeling: python for large-data visualization
  • Clinical data science: python for ML and DL

• Python → How to make it fast?
  • Python vs C (++)
  • Python ways for efficient pipeline.

• Python → Visualization analysis
  • Python vs javascript
  • Python way for ‘client-cloud’ visualization
Clinical Data Science: Analytics+DL

NMGI: Analytics platform

Start with 4 steps:
1) Choose NMGI application from below:
   - HRIM BFT
   - Pre-Post DCI
   - Flip Dynamics
2) Select file ...
   2a) Setup for auto run (default no)
      - Manual input
      - Auto detect inputs from filename
      - Auto read inputs from input file (must be in the same folder of case file)
   2b) Number of cases for auto run (ignored if choosing Manual input)
      - Only the selected case in step 2)
      - All the cases in the same folder as the selected file
3) Select the output folder ...
   Save output in folder: Not chosen
4) Click to start analysis

DL (Upper: CNN classifier; Lower: VAE)
Python applications:

• Scientific large-scale application
  • Large-scale data visualization → ViSiT: Client-server model + OpenGL

• Biophysical modeling (PDE) → FEniCS support MPI run

• Data science stack:
  • Numpy + matplotlib:
  • Scipy
  • Pandas, Sklearn
  • Keras, Tensorflow
Outline

• Brief intro: → research route, applications showcase
  • Scientific modeling: python for large-data visualization
  • Clinical data science: python for ML and DL

• Python → How to make it fast?
  • Efficient pipeline:
  • Python vs C (++)

• Python → Visualization analysis
  • Python vs javascript
  • Python way for ‘client-cloud’ visualization
How to run *things* efficiently?

Example: build a brick wall

[Image: Building a brick wall]

Source: storage

https://www.tensorflow.org/guide/data_performance

Overlapping construction & transportation

Overlapping Memory-intensive Computation-intensive

Transportation

Vectorization (transport more per time)

Numpy, Tensorflow, matlab

Multiple processing

Multi-thread, multi-process

Construction

Transportation
How to run *things* efficiently?

**Python**

Scripting: interpreter

Data type:
- Variable type: (implicit)
- Run-time changeable
  ```
  a =1; a =‘a’;
  ```

Memory:
- Automatic memory management: Garbage collection (reference counts)

Efficiency: native python is slow
- Difficult to multi-thread
- No vectorization (numpy is from C)

**C++**

Programing: compiler + linker

Data type:
- Variable type: explicit, compiled-time assigned.
  ```
  int a =1; str str_a =‘a’;
  ```

Memory:
- User-controlled (malloc):
- Create, release (stack overflow!!)

Efficiency: fast
- Vectorization (Array)
- MPI, threading is easy
Outline

• Brief intro: → research route, applications showcase
  • Scientific modeling: python for large-data visualization
  • Clinical data science: python for ML and DL

• Python → How to make it fast?
  • Efficient pipeline:
  • Python vs C (++)

• Python → Visualization analysis
  • Python vs javascript
  • Python way for ‘client-cloud’ visualization
Visualization analysis: comparison

### Pythonic way

**Pos:**
- Developer-friendly (debug)
- Efficiency tools (numpy, tensorflow)
- Stack for DS, ML (data slicing, ML)
- Data local: data security & privacy
- Computation local: no server burnout

**Neg:**
- Less user-friendly (installation)
- Visualization interactivity
  - Bokeh, Plotly (js for interactivity)

### Javascript (client-server model)

**Pos:**
- Developer control (server-side control)
- User-friendly (zero installation)
- Cross-platform

**Neg:**
- Non-blocking computing in server + client side (ajax)
  - Multiple clients’ request
  - Multiple servers’ response
- Server burden:
  - Computation: server or client?
  - Javascript: slow for data science
New Pythonic way: client-cloud model

Goal: user-friendly $\rightarrow$ zero installation + double-click to run one-way traffic $\rightarrow$ no data flowout

Client side: (local PC)
- Input/output Data: client side only
- Computation: Analysis + Visualization

Cloud side: (read-only remote drive)
- Static python framework (Anaconda)
- Static application modules

Compiled python code

Start with 4 steps:
1) Choose NMGI application from below:
   - HRIM BFT
   - Pre Post DCI
   - Flip Dynamics
2) Select file...
   - File: Not chosen
2.a) Setup for auto run (default no)
   - Manual input
   - Auto detect inputs from filename
   - Auto read inputs from input file (must be in the same folder of case file)
2.b) Number of cases for auto run (ignored if choosing Manual input)
   - Only the selected case in step 2)
   - All the cases in the same folder as the selected file
3) Select the output folder...
   - Save output in folder: Not chosen
4) Click to start analysis
New JS way: client-only analysis – single webpage

Step 1: upload local file with HTIM5 file reader

Step 2: plug in plotting module: plotly.js, or even tensorflow.js

Efficiency → speed up

- Memory intensive data operation: data slicing (Tips from tensorflow.js)
- Overlapping consumer and producer: callback function, asynchrony
- Computation intensive operation: WebGL (use gpu)
Thanks, Happy programming!!

Acknowledge:

• Northwestern Quest (for both parallel simulation + tensorflow DL)

• NIH grant support!