SplitTracer: A Cooperative Inference Evaluation Toolkit for Computation Offloading on the Edge

Rowan University
COLLEGE OF SCIENCE & MATHEMATICS
DEPARTMENT OF COMPUTER SCIENCE

(NSF Award #2128341)

Stephen Piccolo
Nicholas Bovee
Gopi Krishna Patapanchala
Suraj Bitla

Shen-Shyang Ho, Ph.D.
Department of Computer Science
hos@rowan.edu

What is Cooperative Inference?

Cooperative inference tasks involve **multiple devices** working together to infer something from data available to at least one device.

Example: Smart Homes

Consider a smart home designed to optimize energy use by starting the air conditioning prior to occupant arrival, turning lights off when nobody is home, etc.

This smart home may incorporate the following devices working cooperatively:

- Smartphones
- Smart thermostats
- Smart light bulbs
- A central "hub"

Each device communicates with the hub to decide (infer) the optimal action to take.

What is Computation Offloading?

Computation offloading involves the **transfer of computationally complex tasks** from a device with limited processing power to one with more capacity.

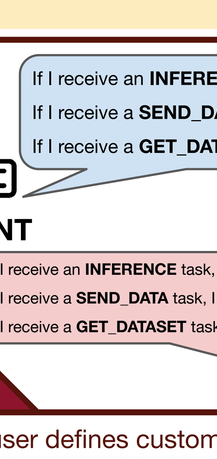
Example: Real-time Language Translation

Consider an app that allows the user to speak into their smartphone, which then plays back translated speech.

The smartphone has limited processing power, so the cloud does the heavy lifting:

- Speech to text
- Translation
- Text to speech

Finally, the cloud sends the results back to the smartphone. This all typically happens in under a second.



What is the Edge?

Edge computing involves **processing data close to where data is generated**, rather than a centralized location like the cloud.

Example: Autonomous Vehicles

Consider a self-driving car that must process vast amounts of data from various sensors in real-time to make decisions instantly.

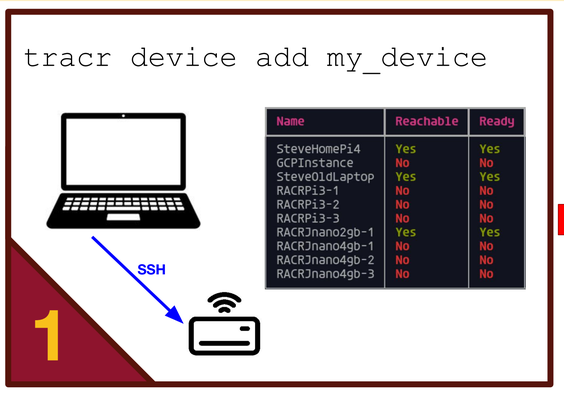
Rather than sending sensor data to the cloud for inference, the car processes data onboard or through nearby edge servers. This allows crucial decisions to be made immediately, without waiting for a response from a distant cloud server.

What does SplitTracer Do?

SplitTracer allows users to deploy edge nodes to their network, tell them how to share resources and make inferences, then record the results.

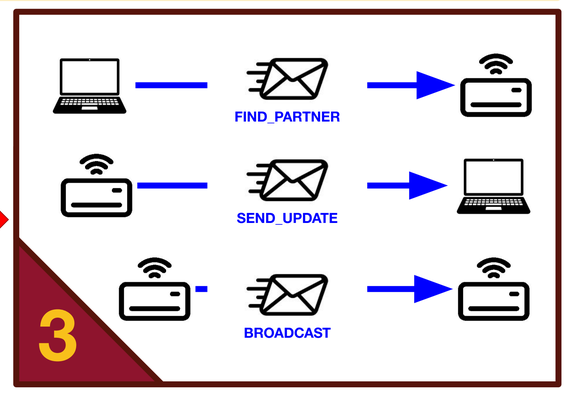
SplitTracer is a Command-Line Interface (CLI) installed and launched from the terminal of the user's desktop or laptop computer (the *controller*). The CLI includes built-in functionality for:

1. **Pairing** remote devices (*participants*) with the controller
2. **Designing custom *roles*** to control how each participant will behave
3. **Specifying custom tasks** that may be sent to a participant for processing
4. **Designing custom experiments** by mapping roles to participants and choosing a sequence of tasks that must be performed in a *manifest* file
5. **Launching multi-node experiments** with a single command
6. **Saving critical performance metrics** recorded automatically during experiment run time

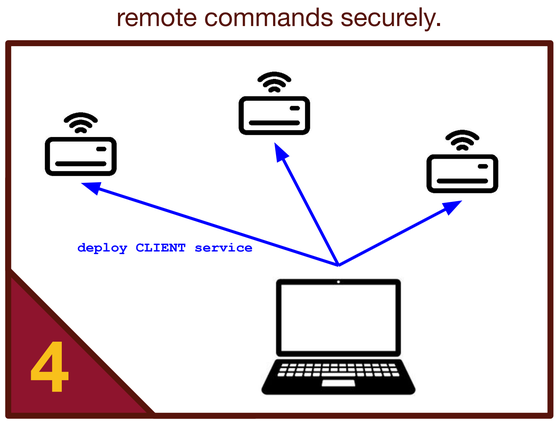


The controller device connects to participants over SSH and saves the connection to send remote commands securely.

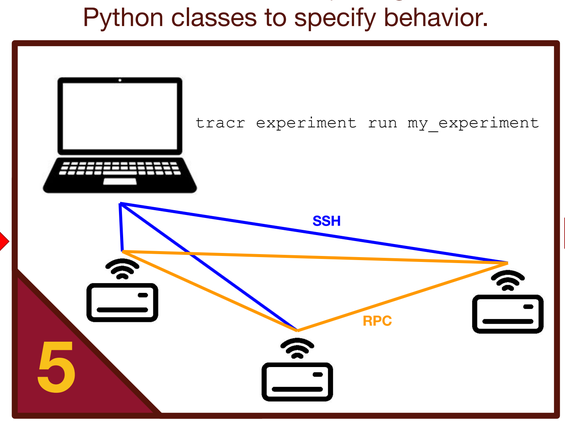
The user defines custom roles (such as "CLIENT" and "SERVER") using extensible Python classes to specify behavior.



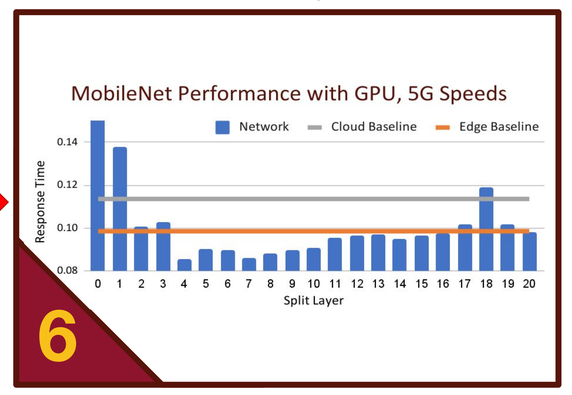
The user can also define custom tasks, again using extensible Python classes included with SplitTracer.



Experiments are launched by deploying RPC services to each participant, corresponding to the behavior specified by the user.



Experiments can be named, saved, and invoked later with a single command.

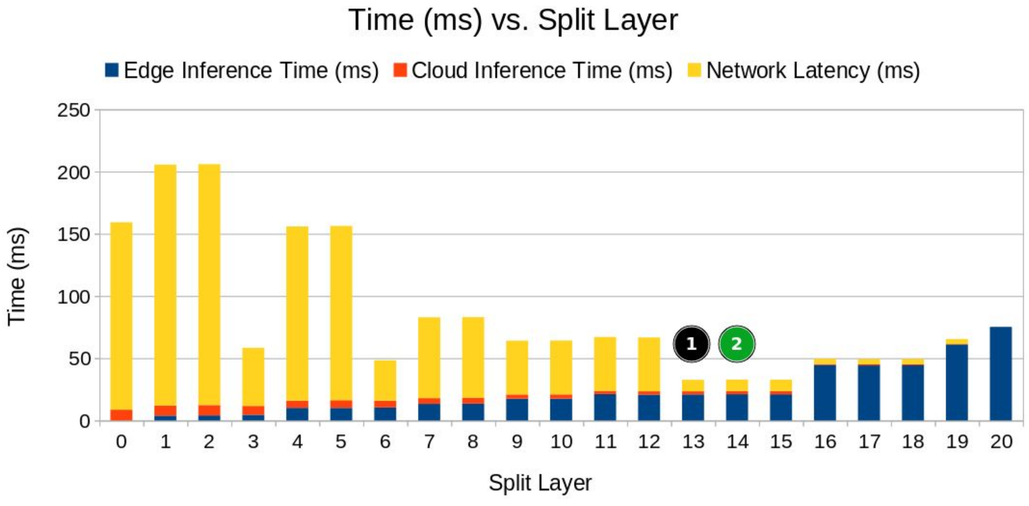


SplitTracer automatically tracks and records metrics, then saves them for the user to analyze and visualize

Replicating a Paper in Minutes with SplitTracer:

Can your phone do some of the work before sending inference tasks to the cloud?

- This is the main question addressed by *Neurosurgeon*.
  - If the mobile device does all the work, inference takes a long time because it has low compute power, & If the whole task is sent to the cloud, large file size causes network transmission to increase
- The authors of *Neurosurgeon* answered this question by setting up an experiment and comparing the total inference latency for each possible split-point (layer) of a neural network.

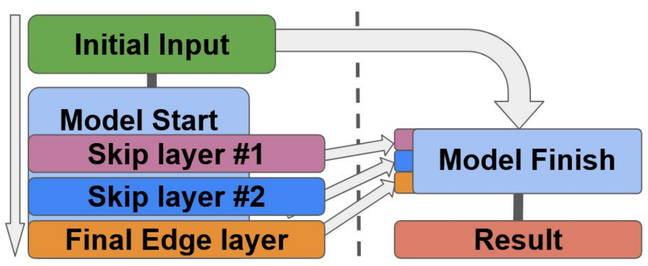


- We replicated these results (in the current modern implementation of AlexNet, the network used at the time)
  - Results are shown concisely with a stacked bar chart: **inference is faster when the mobile device handles the initial layers before sending the rest to the cloud.**
- **Using SplitTracr to configure the same experiment, we were able to successfully replicate the results from *Neurosurgeon* with fewer than 90 lines of Python code and a single command (*left*).**

Handling Yolo/Skip Connections

SplitTracer was also configured in a manner to allow it to process skip style connections dynamically, or by hand tuning.

- By parsing the internal structures of *Yolo*, the skip connections can be mapped & split appropriately



- Dynamic mapping is not perfect and leaves us open to model types that may not then be compatible.
- For these we can allow for custom tuning and specification of a models behavior through yaml files.

Reference:

[1] Kang, Yiping, et al. "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge". ACM SIGARCH 2017.

[2] Hu, Bao, et al. "Dynamic Adaptive DNN Surgery for Inference Acceleration on the Edge". IEEE Conference on Computer Communications 2019.

[3] Official gRPC website (grpc.io). "About gRPC". Accessed 25 July, 2023.

[4] Official Blosc website (blosc.org). "What is Blosc?". Accessed 25 July, 2023.

[5] He et al. "Deep Residual Learning for Image Recognition". IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2016.