

ECE 498 ICC

SP 2020

Prof. V. Kindratenko

Lab 2 Report

Enabling Edge AI in Raspberry Pi

Name: Yucheng Jin

ID: 3170112253

Name: Heyuan Li

ID: 3160103193

Date: April 19, 2020

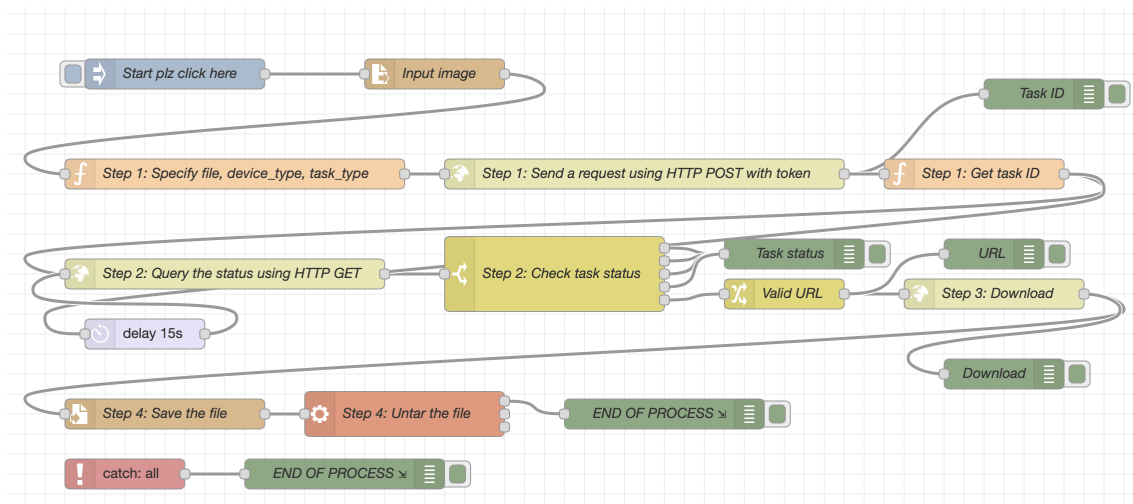
Table of Contents

1	Part I	3
1.1	Check Point I	3
1.2	Check Point II	3
1.3	Check Point III	8
2	Part II	10
2.1	Check Point I	10
2.2	Check Point II	11
2.3	Check Point III	11

1 Part I

1.1 Check Point I

[1 pts] Properly label all nodes within the Remote GPU service (by changing the name fields in their properties panel) and include a screenshot of the Remote GPU service subflow.

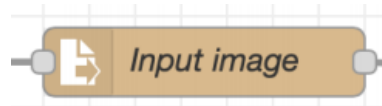


Above is our remote GPU service flow for Part I.

1.2 Check Point II

[2 pts] According to steps listed in the implementation checklist, divide nodes within the Remote GPU service into groups for achieving different goals and comment on each of these groups to justify that your design meet our requirements.

- **Input image**



This node is a *file in* node which inputs the image file.

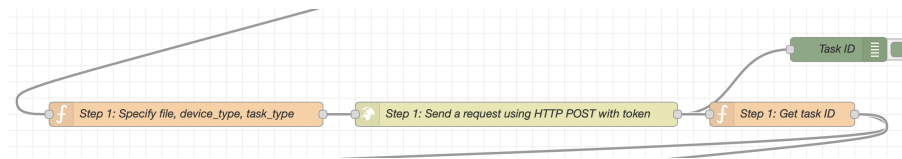
Properties

Filename: /Users/yuchengjin/Desktop/COLLEGE/JUNIOR/SF

Output: a single Buffer object

Name: Input image

- Step 1



There are 4 nodes in this group, including a *function* node which specifies data type, key name for the file, device type, and task type,

Properties

Name: Step 1: Specify file, device_type, task_type

Function

```

1 msg.headers = {
2   'content-type': 'multipart/form-data'
3 };
4
5 var input_image = msg.payload
6
7 msg.payload = {
8   'device_type': 'tx2',
9   'task_type': 'skynet',
10  'file': {
11    'value': input_image,
12    'options': {
13      'filename': msg.filename
14    }
15  }
16 };
17
18 return msg;
  
```

an *HTTP request* node which sends a request to `http://141.142.204.78/uploader` authorized with the token generated by `pyjwt`,

⚙️

Properties

⚙️ 📄 🖨️

☰

Method

POST

⬆️⬆️

🌐

URL

http://141.142.204.78/uploader

☐

Enable secure (SSL/TLS) connection

☒

Use authentication

🔑

Type

bearer authentication

⬆️⬆️

🔒

Token

.....

☐

Enable connection keep-alive

☐

Use proxy

⬅️

Return

a UTF-8 string

⬆️⬆️

🏷️

Name

Step 1: Send a request using HTTP POST with to

a *function* node which gets the taskID,

Properties

Name

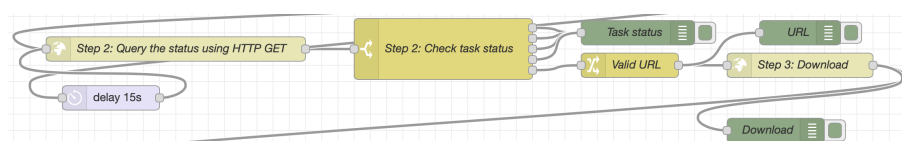
Step 1: Get task ID

Function


1 ▾ msg.headers = {
2 'taskID': msg.payload
3 ^ };
4
5 return msg;




and a *debug* node which shows the taskID.


- Step 2 & Step 3




There are 8 nodes in this group, including an *HTTP request* node which queries the task status,

 **Properties**




 Method GET


 URL

☐ Append msg.payload as query string parameters

☐ Enable secure (SSL/TLS) connection


☒ Use authentication


 Type bearer authentication

 Token


☐ Enable connection keep-alive




☐ Use proxy


 Return a UTF-8 string


 Name



a *switch* node which handles different task statuses,


 **Properties**






 Name


 Property


 == 



→ 1 

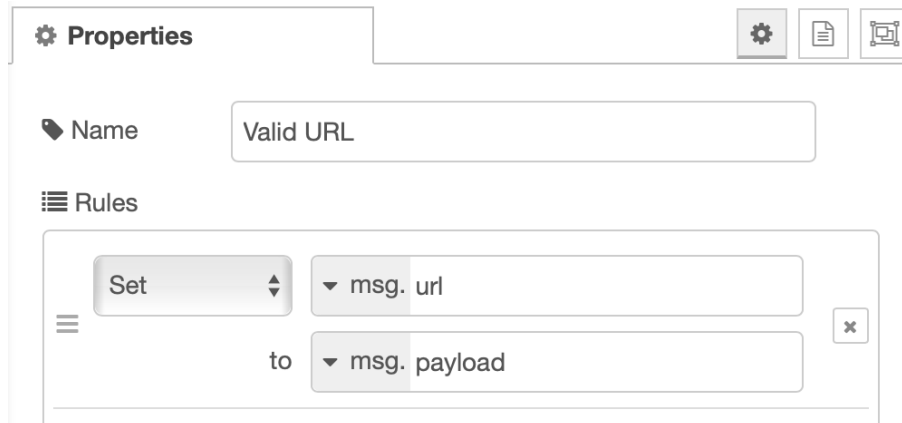
 == 



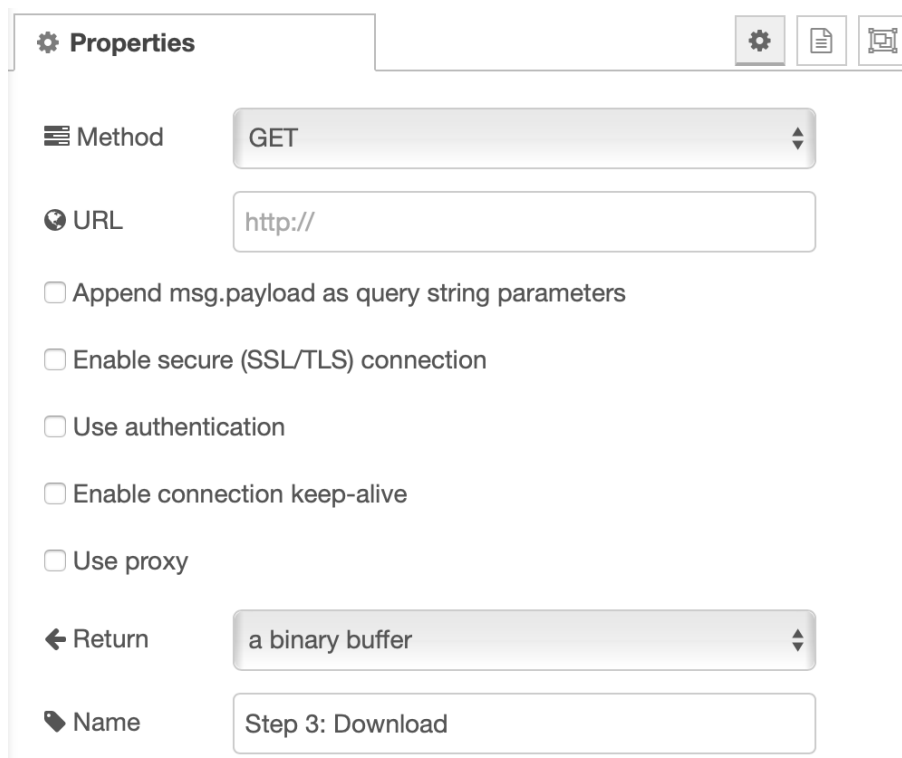
→ 2 

 ==  → 3  ==  → 4  otherwise → 5 

a *change* node which gets the URL,



an *HTTP request* node which sends a request to download the feedback,

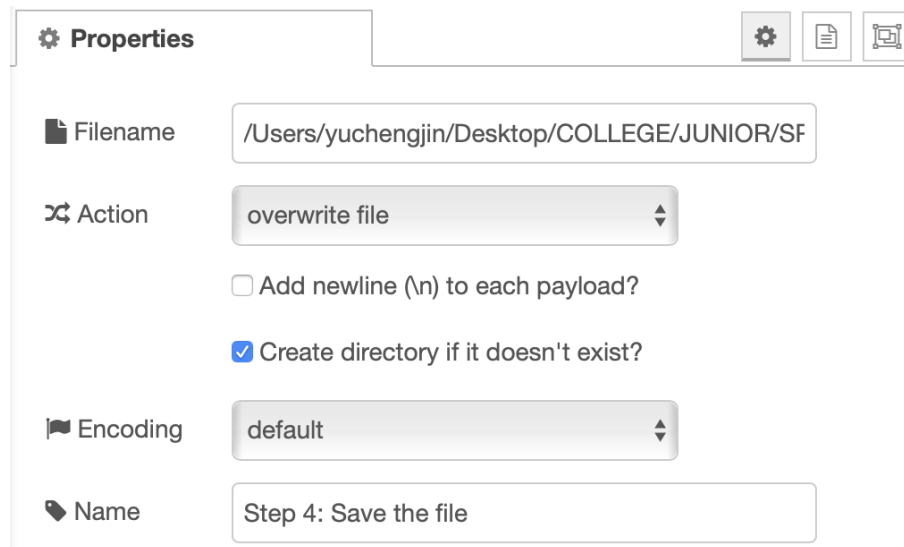


and some *debug* nodes that show task status, URL, and download information.

- **Step 4**



There are 3 nodes in this group, including a *file in* node which saves the feedback,



Properties

Filename: /Users/yuchengjin/Desktop/COLLEGE/JUNIOR/SF

Action: overwrite file

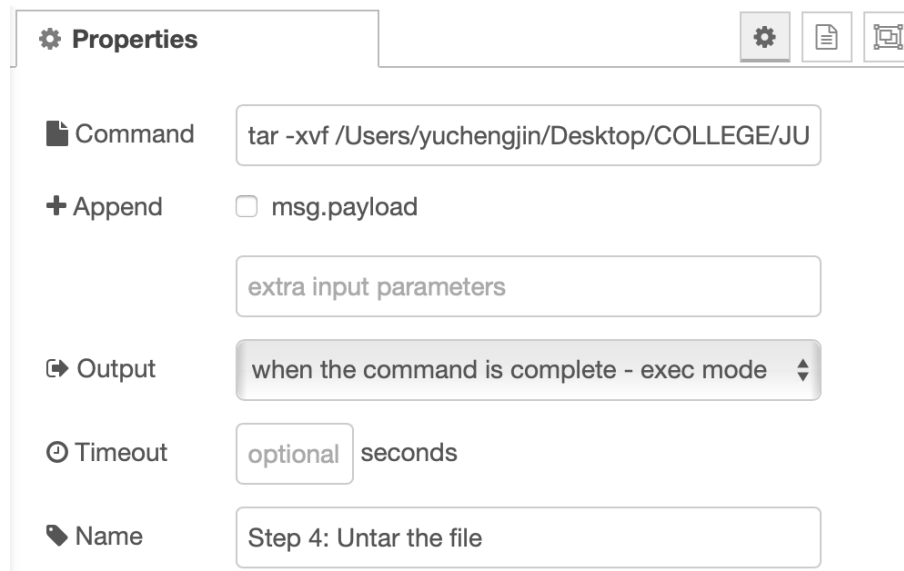
☐ Add newline (\n) to each payload?

☒ Create directory if it doesn't exist?

Encoding: default

Name: Step 4: Save the file

an *exec* node which untars the file,



Properties

Command: tar -xvf /Users/yuchengjin/Desktop/COLLEGE/JU

Append: ☐ msg.payload

extra input parameters

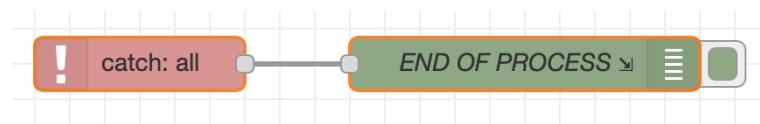
Output: when the command is complete - exec mode

Timeout: optional seconds

Name: Step 4: Untar the file

and a *debug* node which shows the message that the process is finished.

- **Catch**



This group handles with errors.

1.3 Check Point III

[2 pts] Describe at least three challenges you encountered during the development of the Node-RED flow in Part I and how you resolved them.

- Because *function* nodes use JavaScript to specify their functionalities, but we had not learned JavaScript before Lab 2. Therefore, we had to study basic knowledge about JavaScript, and by searching online materials, we successfully learned how to change message header and payload, then finally finished choosing the correct data type, device type, task type, etc.
- At first we did not know how to repeatedly query the task status, and tried several unsuccessful methods. By asking questions in Piazza, we learned that by adding a *delay* node, we could realize the functionality of repeatedly querying the task status.
- For the last step, downloading and untaring the file, we first ended up with damaged file. And we carefully examined every parameter and finally found that we should save a binary buffer instead of an UTF-8 string.

2 Part II

2.1 Check Point I

[1 pts] Collect and report the execution time of all datasets with a table, for basic and tiled MM on both V100 and TX2.

	BasicGEMM, TX2 (in s)									
	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Trial 1	0.204	0.128	0.112	0.108	0.092	0.144	0.200	0.204	0.096	0.188
Trial 2	0.172	0.112	0.096	0.096	0.088	0.132	0.188	0.188	0.084	0.176
Trial 3	0.176	0.116	0.148	0.120	0.100	0.144	0.204	0.200	0.096	0.188
Trial 4	0.160	0.100	0.100	0.100	0.092	0.132	0.188	0.196	0.096	0.184
Trial 5	0.176	0.128	0.116	0.116	0.120	0.172	0.200	0.200	0.096	0.188
	0.178	0.117	0.114	0.108	0.098	0.145	0.196	0.198	0.094	0.185
	BasicGEMM, V100 (in s)									
	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Trial 1	0.212	0.152	0.123	0.141	0.117	0.167	0.215	0.222	0.124	0.206
Trial 2	0.204	0.149	0.127	0.129	0.119	0.161	0.199	0.233	0.129	0.206
Trial 3	0.186	0.145	0.151	0.128	0.122	0.173	0.204	0.23	0.117	0.212
Trial 4	0.199	0.139	0.146	0.124	0.132	0.185	0.21	0.224	0.113	0.22
Trial 5	0.202	0.134	0.125	0.132	0.128	0.179	0.221	0.216	0.113	0.208
	0.201	0.144	0.134	0.131	0.124	0.173	0.210	0.225	0.119	0.210
	TiledGEMM, TX2 (in s)									
	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Trial 1	0.202	0.126	0.111	0.104	0.092	0.156	0.179	0.200	0.099	0.183
Trial 2	0.194	0.109	0.104	0.112	0.092	0.142	0.174	0.200	0.089	0.183
Trial 3	0.156	0.105	0.108	0.098	0.088	0.140	0.171	0.188	0.088	0.181
Trial 4	0.166	0.112	0.111	0.098	0.094	0.151	0.193	0.191	0.088	0.177
Trial 5	0.168	0.117	0.111	0.102	0.097	0.137	0.187	0.199	0.091	0.188
	0.177	0.114	0.109	0.103	0.093	0.145	0.181	0.196	0.091	0.182
	TiledGEMM, V100 (in s)									
	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Trial 1	0.185	0.119	0.118	0.122	0.106	0.152	0.188	0.219	0.119	0.193
Trial 2	0.18	0.114	0.115	0.119	0.109	0.159	0.204	0.221	0.106	0.2
Trial 3	0.201	0.122	0.12	0.119	0.111	0.153	0.202	0.203	0.102	0.202
Trial 4	0.197	0.127	0.115	0.111	0.108	0.153	0.185	0.214	0.105	0.201
Trial 5	0.181	0.118	0.115	0.12	0.103	0.159	0.193	0.205	0.11	0.203
	0.189	0.120	0.117	0.118	0.107	0.155	0.194	0.212	0.108	0.200

Above is our table. For BasicGEMM on TX2, BasicGEMM on V100, TiledGEMM on TX2, and TiledGEMM on V100, we each ran 5 times.

2.2 Check Point II

[2 pts] Compare the execution time and explain how TiledGEMM can reduce global memory access. Calculate and report the maximum theoretical reduction of global memory access with your implementation.

- **Compare the execution time:** from the table above, TiledGEMM runs faster than BasicGEMM, and they both run faster on TX2 compared to V100.
- **How TiledGEMM can reduce global memory access:** TiledGEMM takes the advantage of shared memory. For each tile, Tiled MM copies data from global memory to shared memory and then performs calculations. For global memory, it takes around 500 cycles to access data; but for shared memory, it takes only about 5 cycles to access data. Although it takes some time to copy data from global memory to shared memory, but for each calculation shared memory reduces the number of cycles to access data, so TiledGEMM overall decreases the execution time by reducing global memory access.
- **Calculate and report the maximum theoretical reduction of global memory access:** using 16×16 tiling, we reduce the global memory access by a factor of 16; using 32×32 tiling, we reduce the global memory access by a factor of 32; generally, using $N \times N$ tiling, we reduce the global memory access by a factor of N . It's easy to come up with the conclusion, just think about the multiplication of two $N \times N$ matrices, if we use BasicGEMM, it will access global memory $2 \times N \times N \times N$ times; but for TiledGEMM, using $N \times N$ tiling, it will access global memory $2 \times N \times N$ times, so **using $N \times N$ tiling, we reduce the global memory access by a factor of N .**

2.3 Check Point III

[2 pts] Compare the hardware specifications of V100 and TX2 (e.g. number of SMs, etc.) and briefly explain the factors that causes the performance difference.

	TX2	V100
SM	2	80
CUDA Cores	256	5120
Memory	8 GB 128-bit LPDDR4	32 GB /16 GB HBM2
Bandwidth	59.7 GB/s	up to 900 GB/s

Above table shows the the hardware specifications of V100 and TX2¹.

¹Data from NVIDIA official datasheet, <https://www.nvidia.com/en-us/data-center/v100>, <https://developer.nvidia.com/embedded/jetson-tx2>.

From the hardware specifications we know that compared with TX2, V100 has more SMs and CUDA cores, larger memory and bandwidth. With more SMs and CUDA cores, V100 is able to process tasks at a higher speed; and with larger memory and bandwidth, V100 is able to handle more data and has a significantly higher throughput. So in theory, V100 generally performs better than TX2. But since the matrix multiplication task in our Lab 2 maybe too easy, thus the full resources of V100 were not being taken advantage of, so we did not see V100 outperform this time.