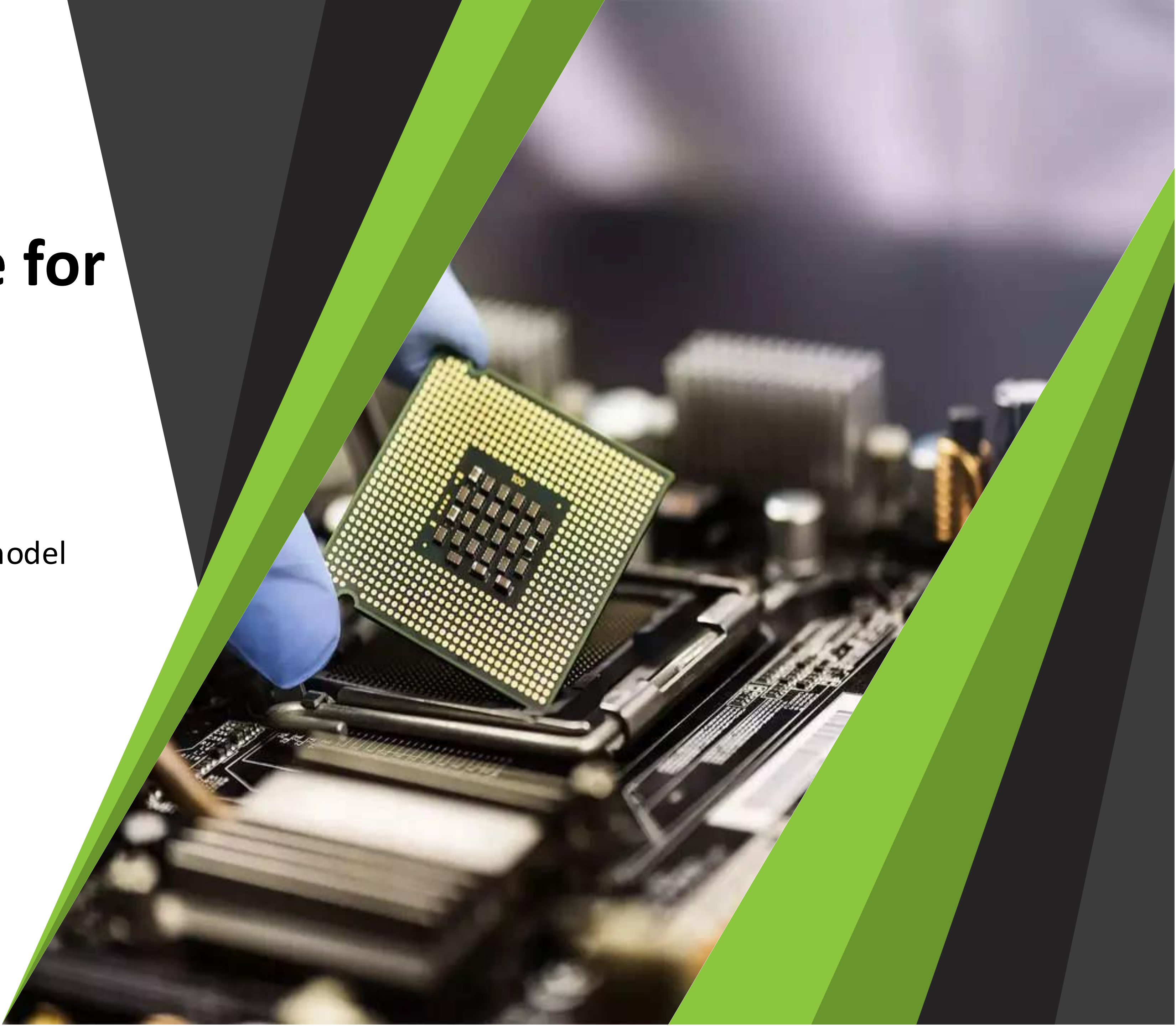
Python-Based Pipeline for Sonic-Dash

A fast, testable, and developer-friendly model



DreamBig SEMICONDUCTOR

Contributors

Project Lead:

- Farhat Ullah

Development Lead:

- Farhan Tariq

Developer(s):

- Muhammad Afaq Younas
- Saad Mazhar (formerly with Dreambig Semiconductors Inc.)

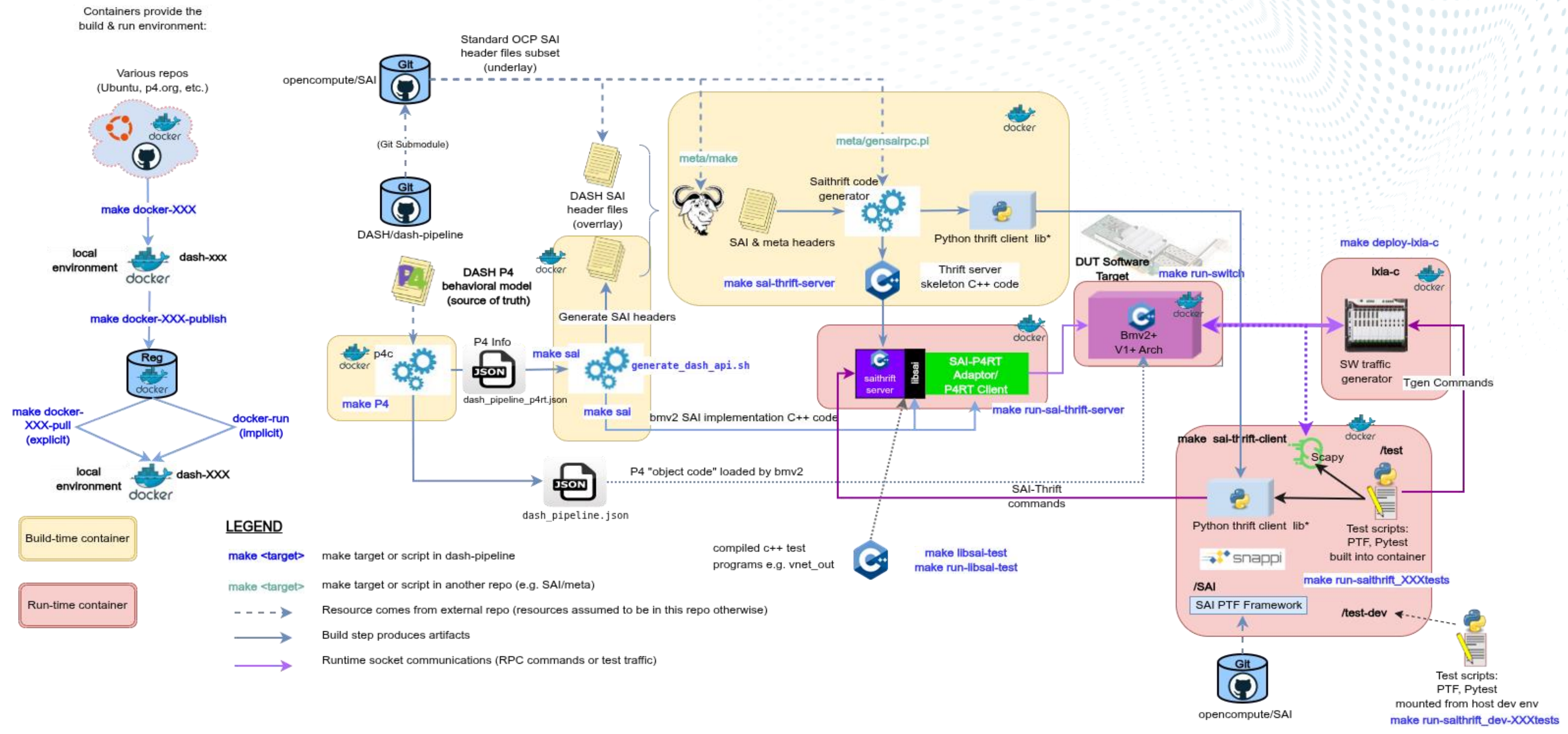
Agenda

- Motivation
- DASH architecture
- Comparison of python model with P4
- Code walk through
- CI/CD pipeline
- Developer flow & running python model
- Demo

Why Python Model?

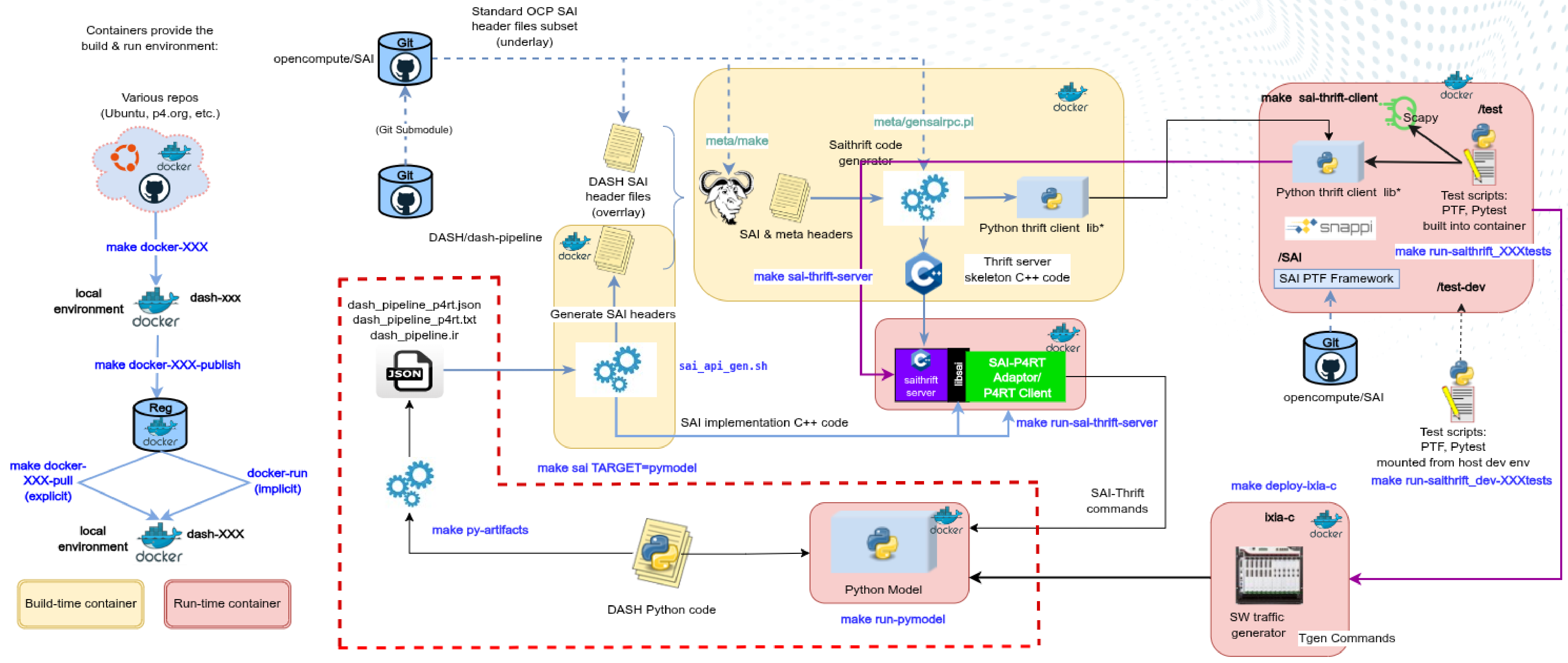
P4 language challenges:	Why Python?
• Complex debugging • Limited feature set • Constraints on stateful flow operations	• Favorite for simulations and quick prototyping • Instant feedback on code changes • No compilation required • Easier and more intuitive debugging • Rich ecosystem of tools and libraries • Easy integration with APIs and external systems • Large and active developer community

Architecture Overview – BMv2


Containers provide the build & run environment:
Various repos (Ubuntu, p4.org, etc.)
docker
make docker-XXX
local environment
dash-xxx
make docker-XXX-publish
Reg
make docker-XXX-pull (explicit)
docker-run (implicit)
local environment
dash-XXX
Build-time container
Run-time container
Git
opencompute/SAI
Standard OCP SAI header files subset (underlay)
(Git Submodule)
DASH/dash-pipeline
DASH P4 behavioral model (source of truth)
p4c
make P4
P4 Info
JSON
dash_pipeline_p4rt.json
make sai
DASH SAI header files (overlay)
Generate SAI headers
generate_dash_api.sh
make sai
bmv2 SAI implementation C++ code
dash_pipeline.json
P4 "object code" loaded by bmv2
meta/make
meta/gensairpc.pl
Saithrift code generator
SAI & meta headers
Python thrift client lib*
Thrift server skeleton C++ code
make sai-thrift-server
saithrift server
libsai
SAI-P4RT Adaptor/ P4RT Client
make run-sai-thrift-server
DUT Software Target
make run-switch
Bmv2+ V1+ Arch
make deploy-ixia-c
ixia-c
SW traffic generator
Tgen Commands
make sai-thrift-client
Scapy
/test
Python thrift client lib*
Test scripts: PTF, Pytest built into container
snappi
make run-saithrift_XXXtests
/SAI
SAI PTF Framework
/test-dev
SAI-Thrift commands
compiled c++ test programs e.g. vnet_out
make libsai-test
make run-libsai-test
opencompute/SAI
Test scripts: PTF, Pytest mounted from host dev env
make run-saithrift_dev-XXXtests
LEGEND
make <target> make target or script in dash-pipeline
make <target> make target or script in another repo (e.g. SAI/meta)
Resource comes from external repo (resources assumed to be in this repo otherwise)
Build step produces artifacts
Runtime socket communications (RPC commands or test traffic)

Credits: Chris Sommers

Architecture Overview – Python Model



LEGEND

make <target>	make target or script in dash-pipeline
make <target>	make target or script in another repo (e.g. SAI/meta)
- - - ➤	Resource comes from external repo (resources assumed to be in this repo otherwise)
➤	Build step produces artifacts
➤	Runtime socket communications (RPC commands or test traffic)
- - - ➤	Python Model

Comparison with BMv2 Model (Similarities)

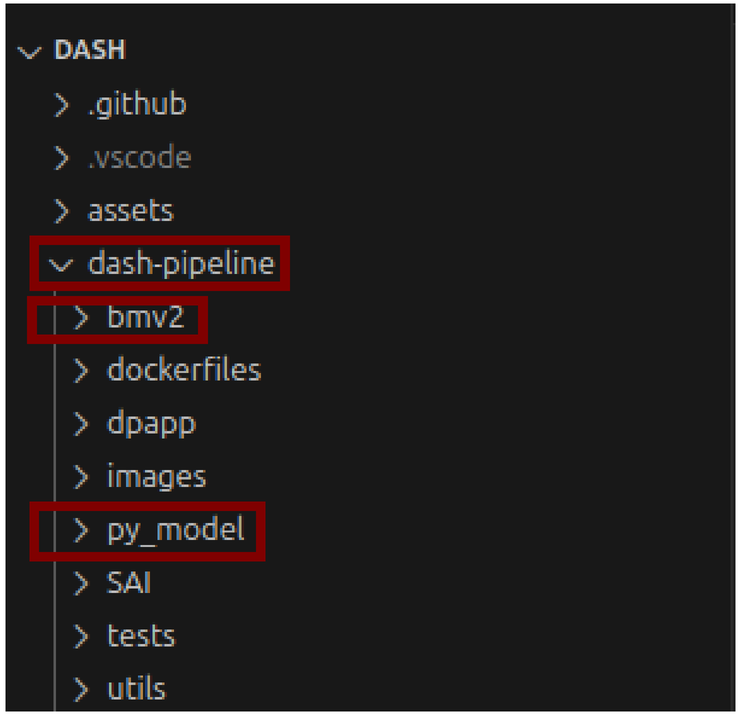
Common features

- Both models implement the same DASH processing pipeline
- Identical table structures and lookup logic
- Consistent routing actions and packet transformations
- Similar P4Info and SAI API generation
- Utilize the same test suites (PTF and Pytest)

Comparison with BMv2 Model (Differences)

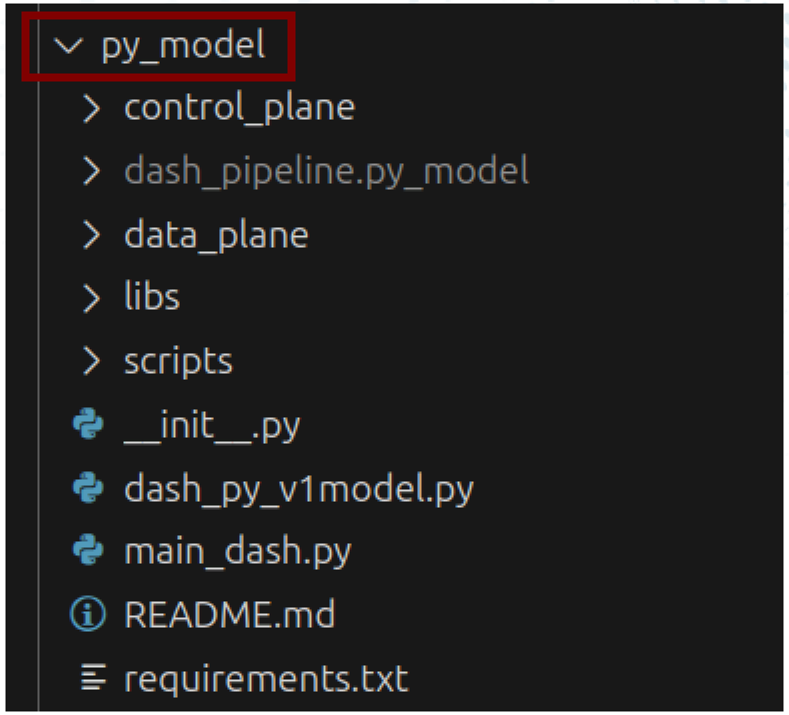
Aspect	BMv2	Python Model
Language	P4 and C++	Python
Compilation	Required	Not needed
Execution	Hardware-centric	Developer-friendly
Debugging	Complex	Simple
Threads	Multi-threaded	Single-threaded

DASH Directory Structure (Code)



```
⌄ DASH
  › .github
  › .vscode
  › assets
  ⌄ dash-pipeline
    › bmv2
    › dockerfiles
    › dpapp
    › images
    › py_model
    › SAI
    › tests
    › utils
```

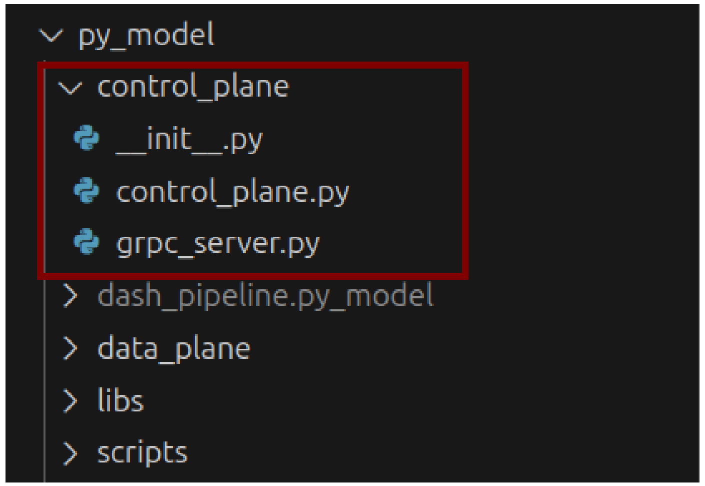
DASH Pipeline



```
⌄ py_model
  › control_plane
  › dash_pipeline.py_model
  › data_plane
  › libs
  › scripts
  __init__.py
  dash_py_v1model.py
  main_dash.py
  README.md
  requirements.txt
```

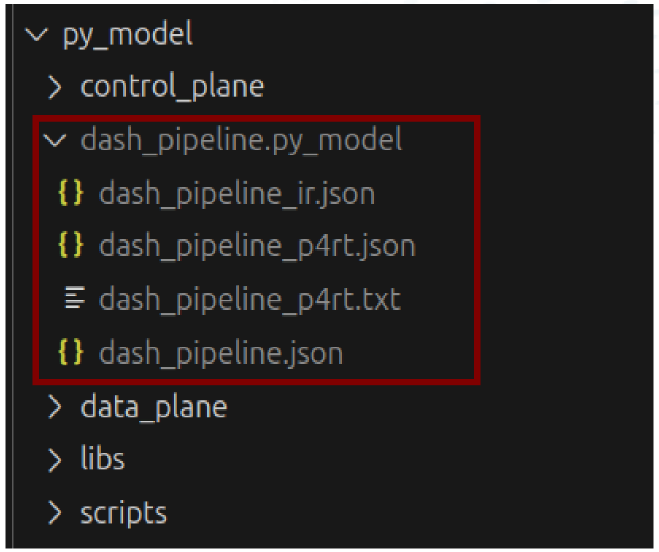
Python model

Python Model Directory Structure (Code)



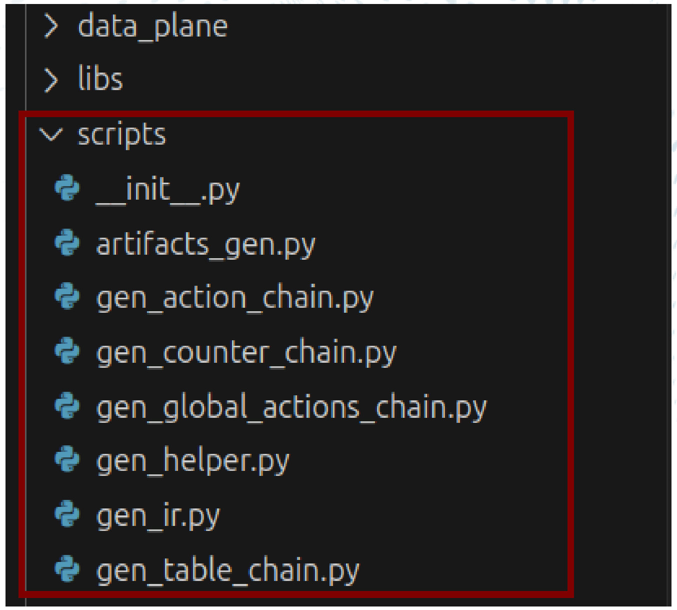
```
py_model
  control_plane
    __init__.py
    control_plane.py
    grpc_server.py
  dash_pipeline.py_model
  data_plane
  libs
  scripts
```

Control plane



```
py_model
  control_plane
  dash_pipeline.py_model
    dash_pipeline_ir.json
    dash_pipeline_p4rt.json
    dash_pipeline_p4rt.txt
    dash_pipeline.json
  data_plane
  libs
  scripts
```

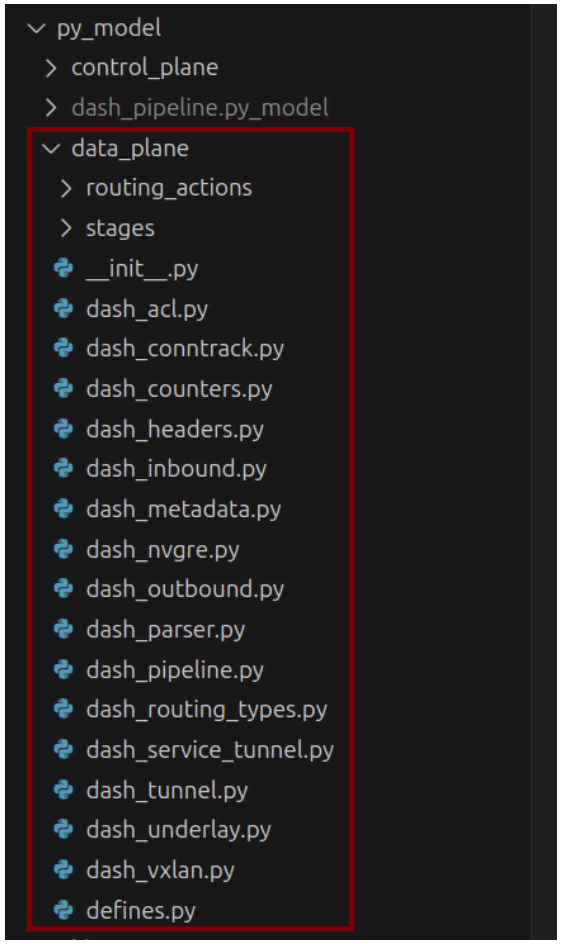
Runtime files



```
data_plane
libs
scripts
  __init__.py
  artifacts_gen.py
  gen_action_chain.py
  gen_counter_chain.py
  gen_global_actions_chain.py
  gen_helper.py
  gen_ir.py
  gen_table_chain.py
```

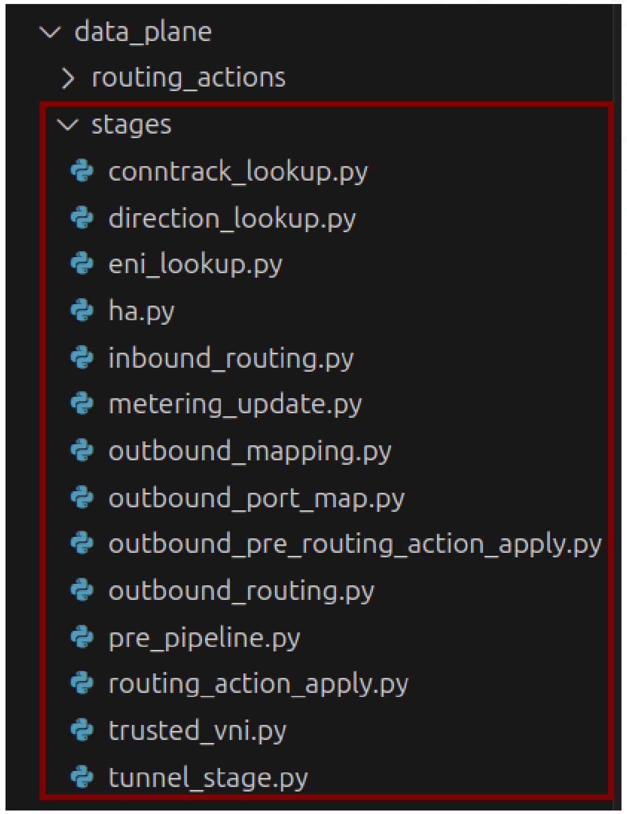
Runtime files generator scripts

Python Model Directory Structure (Code) – Contd.



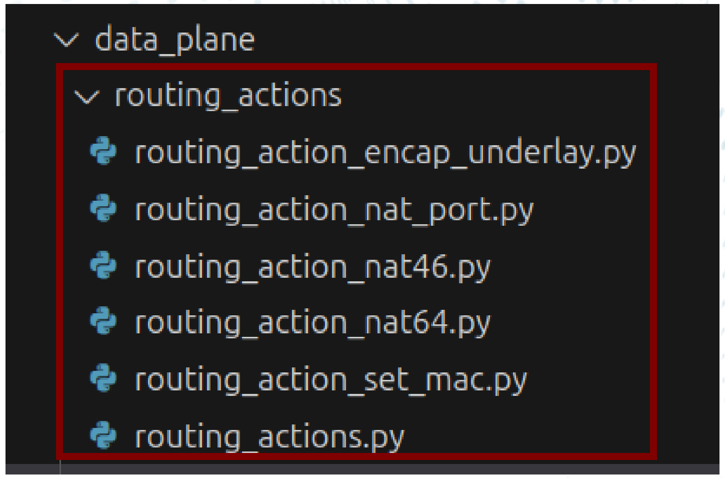
```
py_model
  control_plane
  dash_pipeline.py_model
  data_plane
    routing_actions
    stages
    __init__.py
    dash_acl.py
    dash_conntrack.py
    dash_counters.py
    dash_headers.py
    dash_inbound.py
    dash_metadata.py
    dash_nvgre.py
    dash_outbound.py
    dash_parser.py
    dash_pipeline.py
    dash_routing_types.py
    dash_service_tunnel.py
    dash_tunnel.py
    dash_underlay.py
    dash_vxlan.py
    defines.py
```

Data plane / Pipeline



```
data_plane
  routing_actions
  stages
    conntrack_lookup.py
    direction_lookup.py
    eni_lookup.py
    ha.py
    inbound_routing.py
    metering_update.py
    outbound_mapping.py
    outbound_port_map.py
    outbound_pre_routing_action_apply.py
    outbound_routing.py
    pre_pipeline.py
    routing_action_apply.py
    trusted_vni.py
    tunnel_stage.py
```

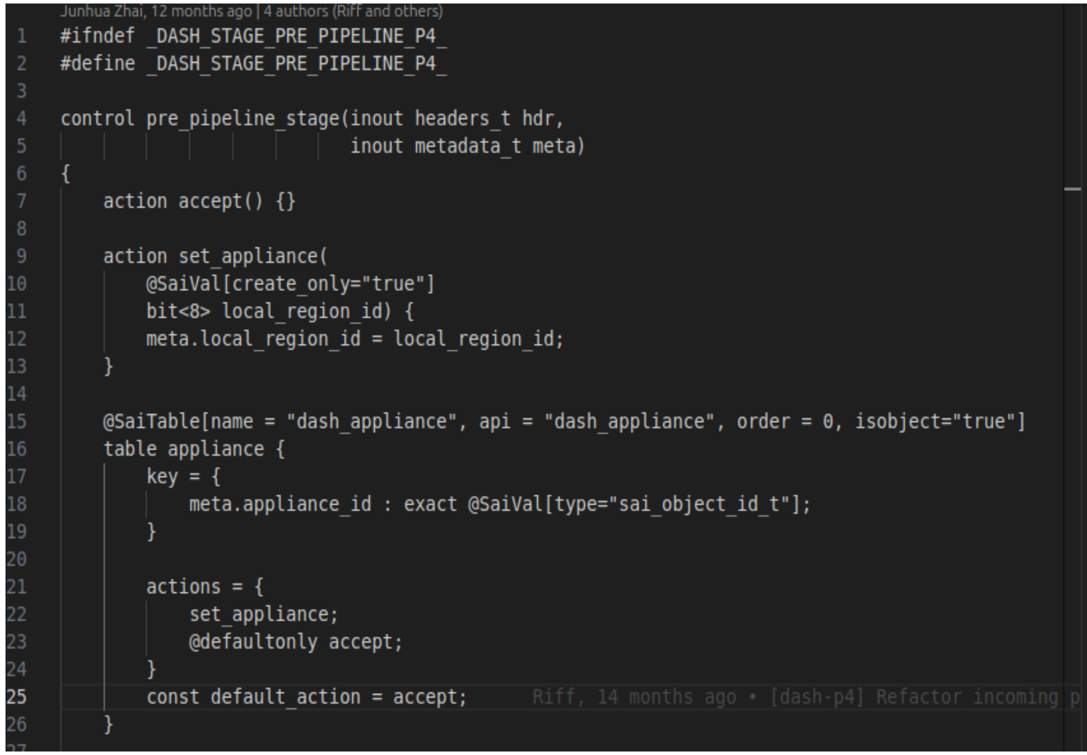
Stages



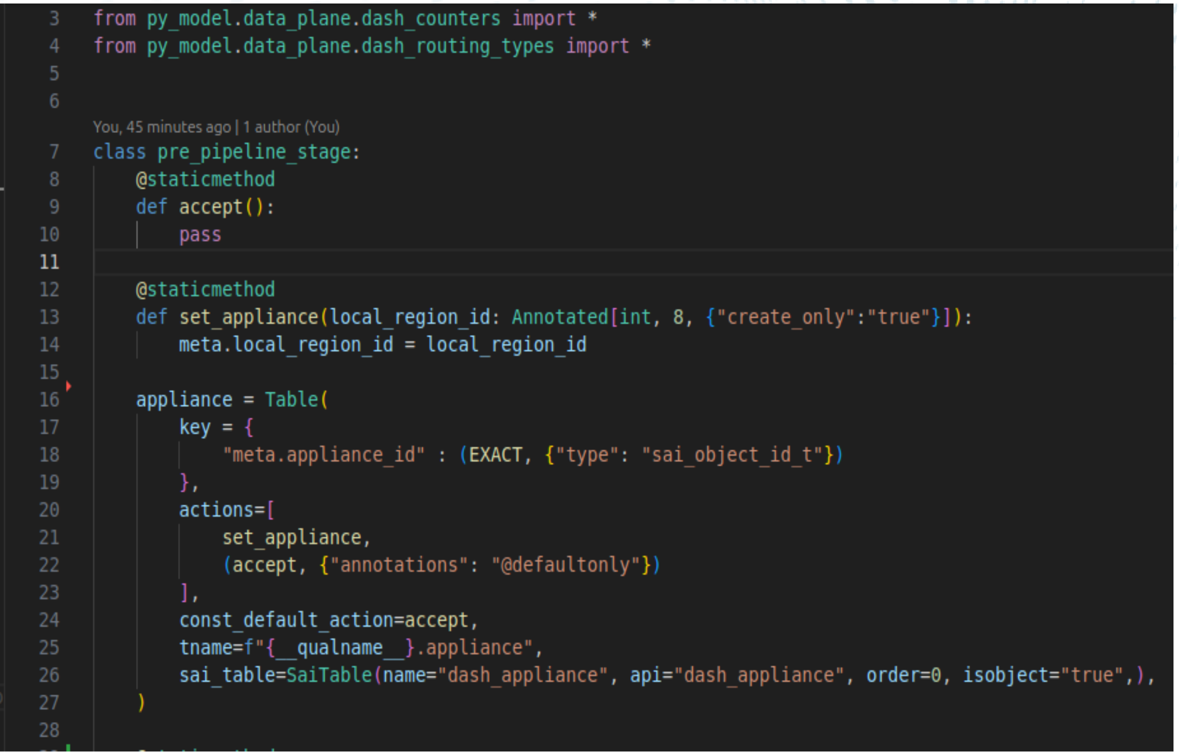
```
data_plane
  routing_actions
    routing_action_encap_underlay.py
    routing_action_nat_port.py
    routing_action_nat46.py
    routing_action_nat64.py
    routing_action_set_mac.py
    routing_actions.py
```

Routing actions

P4 vs Python - Code



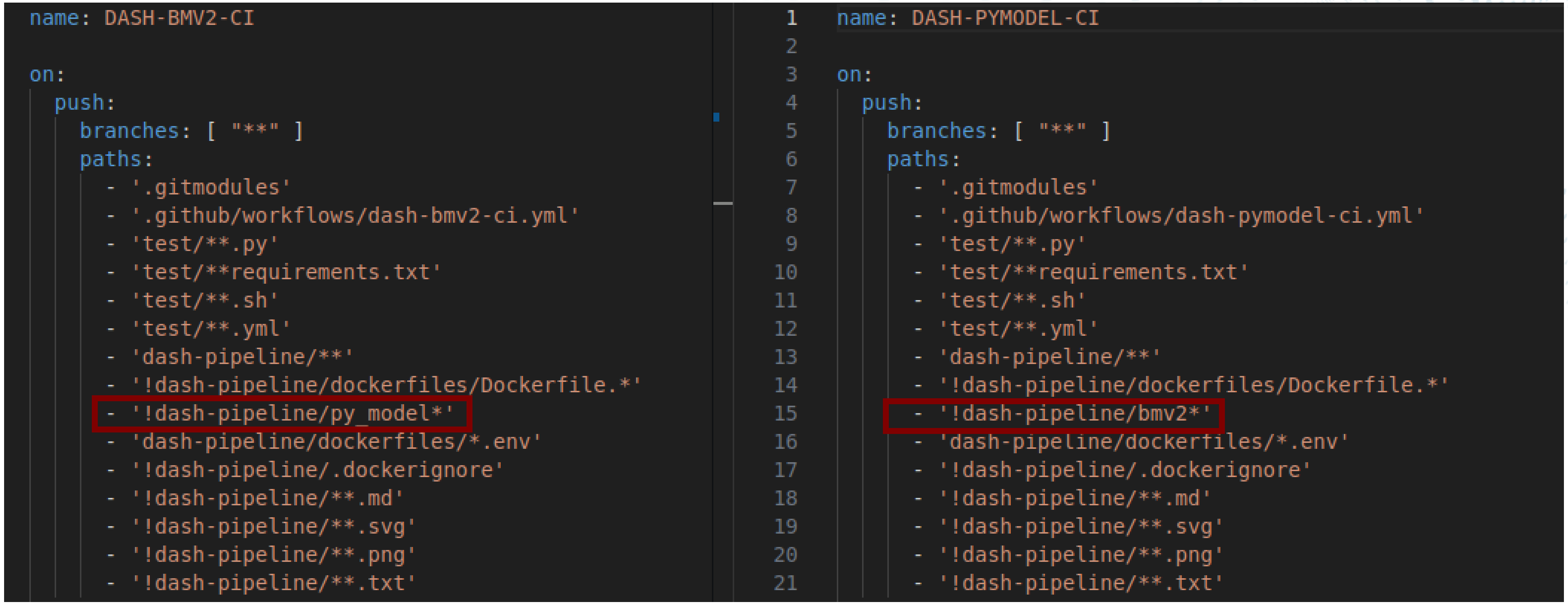
```
Junhua Zhai, 12 months ago | 4 authors (Riff and others)
#ifndef _DASH_STAGE_PRE_PIPELINE_P4_
#define _DASH_STAGE_PRE_PIPELINE_P4_

control pre_pipeline_stage(inout headers_t hdr,
                           inout metadata_t meta)
{
    action accept() {}

    action set_appliance(
        @SaiVal[create_only="true"]
        bit<8> local_region_id) {
        meta.local_region_id = local_region_id;
    }

    @SaiTable[name = "dash_appliance", api = "dash_appliance", order = 0, isobject="true"]
    table appliance {
        key = {
            meta.appliance_id : exact @SaiVal[type="sai_object_id_t"];
        }

        actions = {
            set_appliance;
            @defaultonly accept;
        }
        const default_action = accept;      Riff, 14 months ago • [dash-p4] Refactor incoming p
    }
```



```python
from py_model.data_plane.dash_counters import *
from py_model.data_plane.dash_routing_types import *


You, 45 minutes ago | 1 author (You)
class pre_pipeline_stage:
    @staticmethod
    def accept():
        pass

    @staticmethod
    def set_appliance(local_region_id: Annotated[int, 8, {"create_only":"true"}]):
        meta.local_region_id = local_region_id

    appliance = Table(
        key = {
            "meta.appliance_id" : (EXACT, {"type": "sai_object_id_t"})
        },
        actions=[
            set_appliance,
            (accept, {"annotations": "@defaultonly"})
        ],
        const_default_action=accept,
        tname=f"{__qualname__}.appliance",
        sai_table=SaiTable(name="dash_appliance", api="dash_appliance", order=0, isobject="true",),
    )
```

P4 Code (left), Python Code (right)

P4 vs Python - CI Pipeline

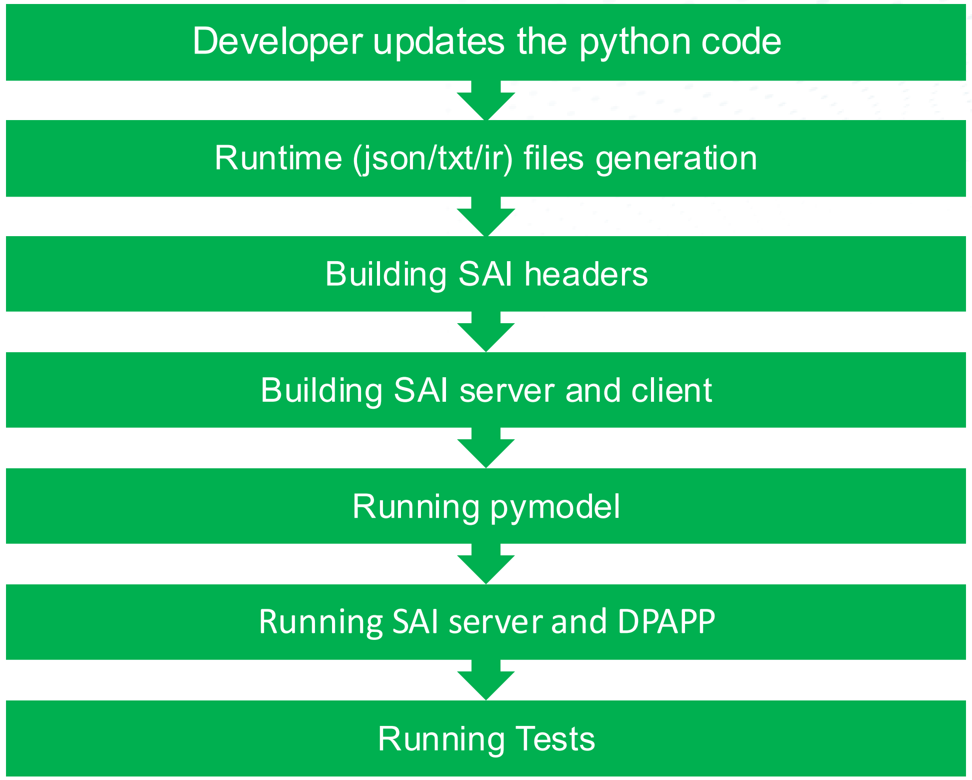


```yaml
name: DASH-BMV2-CI

on:
  push:
    branches: [ "**" ]
    paths:
      - '.gitmodules'
      - '.github/workflows/dash-bmv2-ci.yml'
      - 'test/**.py'
      - 'test/**requirements.txt'
      - 'test/**.sh'
      - 'test/**.yml'
      - 'dash-pipeline/**'
      - '!dash-pipeline/dockerfiles/Dockerfile.*'
      - '!dash-pipeline/py_model*'
      - 'dash-pipeline/dockerfiles/*.env'
      - '!dash-pipeline/.dockerignore'
      - '!dash-pipeline/**.md'
      - '!dash-pipeline/**.svg'
      - '!dash-pipeline/**.png'
      - '!dash-pipeline/**.txt'
```

```yaml
name: DASH-PYMODEL-CI

on:
  push:
    branches: [ "**" ]
    paths:
      - '.gitmodules'
      - '.github/workflows/dash-pymodel-ci.yml'
      - 'test/**.py'
      - 'test/**requirements.txt'
      - 'test/**.sh'
      - 'test/**.yml'
      - 'dash-pipeline/**'
      - '!dash-pipeline/dockerfiles/Dockerfile.*'
      - '!dash-pipeline/bmv2*'
      - 'dash-pipeline/dockerfiles/*.env'
      - '!dash-pipeline/.dockerignore'
      - '!dash-pipeline/**.md'
      - '!dash-pipeline/**.svg'
      - '!dash-pipeline/**.png'
      - '!dash-pipeline/**.txt'
```

BMv2 CI Pipeline (left), Python CI Pipeline (right)

Developer Workflow for Python Model


Developer updates the python code
Runtime (json/txt/ir) files generation
Building SAI headers
Building SAI server and client
Running pymodel
Running SAI server and DPAPP
Running Tests

Documentation: `dash-pipeline/py_model/README.md`

Status

Completed

- Core DASH pipeline (routing, VNet, ENI)
- Packet parsing/deparsing
- Table lookups and actions
- Data plane application (dpapp)
- SAI Thrift integration
- P4Info/SAI artifact generation
- PTF test and Pytests execution

Future work

- SAI Challenger
- Fast Path
- Multi-threading

Documentation: `dash-pipeline/py_model/README.md`

Running Pymodel

Follow the below steps to test the Python model using PTF and Py Tests:

Terminal	Purpose	Commands(s)
Terminal 1	Build artifacts and to run the python model	1. \`make py-artifacts\` 2. \`make sai TARGET=pymodel\` 3. \`make dpapp TARGET=pymodel\` 4. \`make saithrift-server\` 5. \`make docker-saithrift-client\` 6. \`sudo make run-pymodel HAVE_DPAPP=y\`
Terminal 2	Run DPAPP	1. \`make run-dpapp TARGET=pymodel\`
Terminal 3	Run SAI Thrift Server	1. \`make run-saithrift-server TARGET=pymodel\`
Terminal 4	Run Tests	1. \`make run-saithrift-ptftests\` 2. \`make run-saithrift-pytests\`

Documentation: \`README-dash-pymodel-workflows.md\`

Thank You

Contact: ftariq@dreambigsemi.com

GitHub PR: https://github.com/sonic-net/DASH/pull/687

Documentation: `README-dash-pymodel-workflows.md`