TAC ANNUAL REVIEW
24 JUNE 2026

ASWF /* ACADEMY SOFTWARE FOUNDATION

The mission of the Project is to develop a VFX-specific framework for simplifying production and deployment of AI/ML models using well-tracked and rights-cleared datasets to enable clear provenance tracking

*RMTC ASWF Technical Charter

TRACK

- Marking artifacts with licenses & rights
- Asset manager integration
- Injection into existing AI pipelines

TRACE

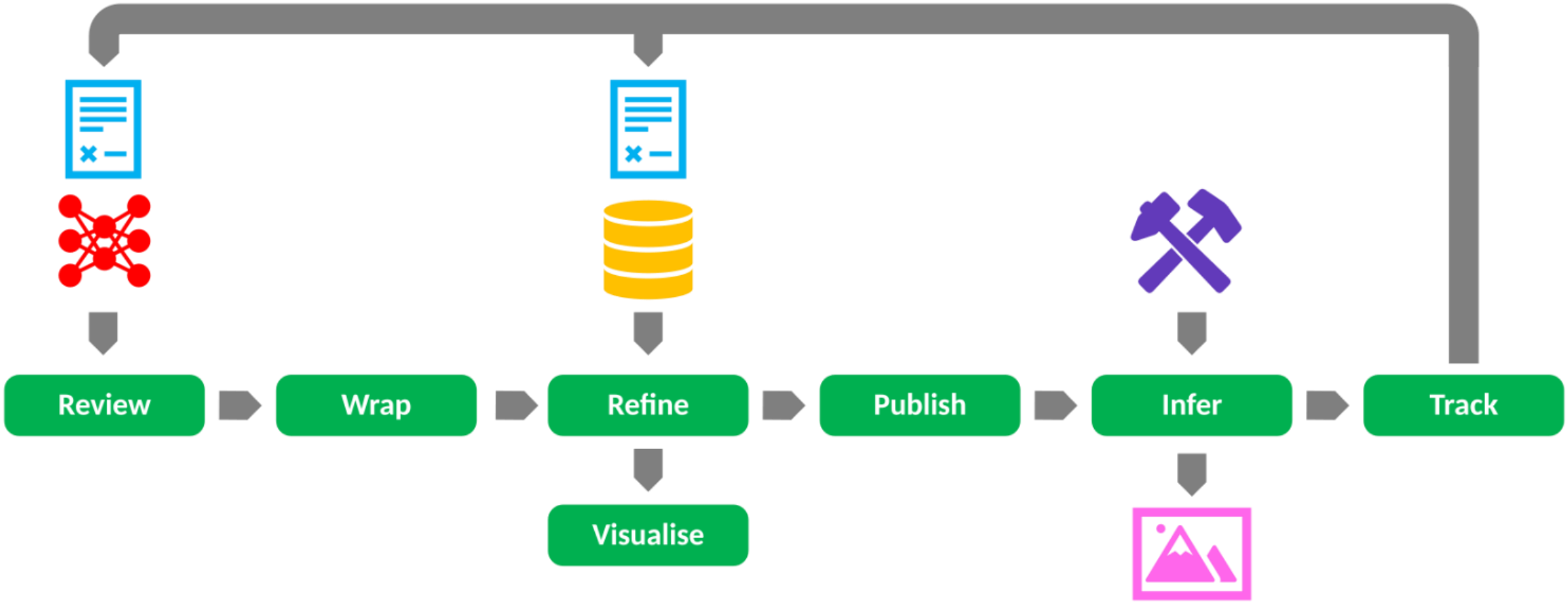
- Tracing across AI & CG pipelines
- Querying artifact sources
- Reporting
- Watermarking
- Non-prod interface

TRAIN

- Training guardrails
- Dataset abstractions
- Simple refinement training
- Automatic tracking

INFER

- Inference guardrails
- VFX Asset types
- DCC integration
- Tensor processing
- Automatic tracking



TSC

Code

Production Testing

Comms & Community

ASWF /* ACADEMY SOFTWARE FOUNDATION

TECHNIAL STEERING COMMITTEE


NVIDIA


RSP


SONY PICTURES
imageworks


wētā FX

TSC

- Since February
- Fortnightly meetings
- Centered around existing develop branch
- Primarily architecture discussions and policy

Juan Buhler
Sony Imageworks

John McCarten
Wētā FX

Jonathan Swartz
NVIDIA

Troy Tobin
RSP

Kimball Thurston
Wētā FX

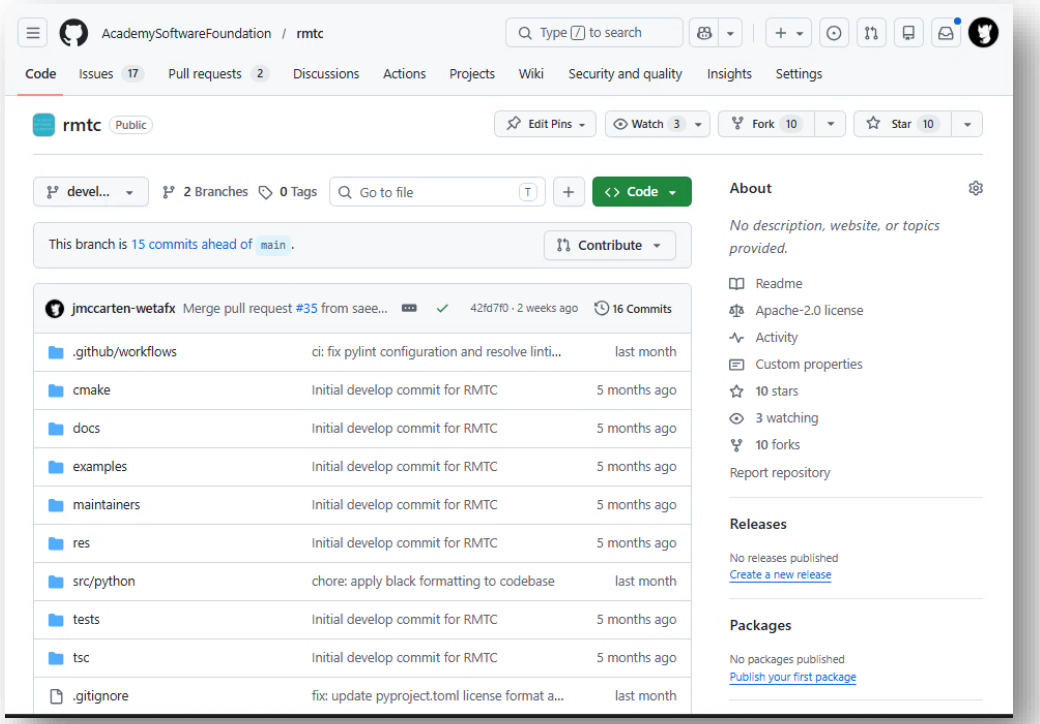
POLICIES & PROCESS

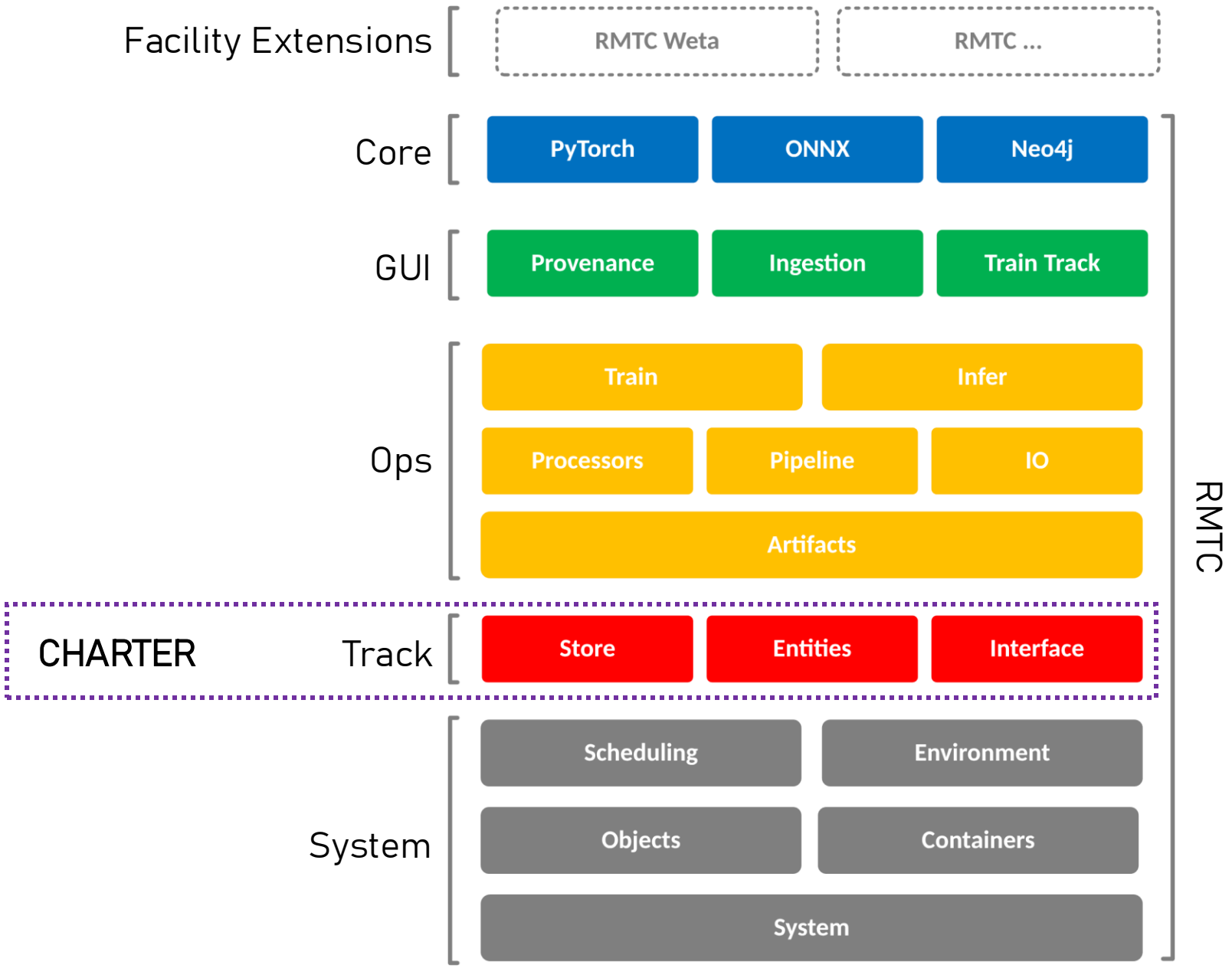
- Follows model project
  - Committers
  - Governance
  - Contributing
  - Security
  - Support
- Novel AI policy document
  - AI definitions
  - Tiered adoption
- Issues Tracker
  - GitHub tickets
  - Coarse level of issues – more depth
- Documentation
  - GitHub wiki
  - TSC notes
- CI/CD
- LF PCC

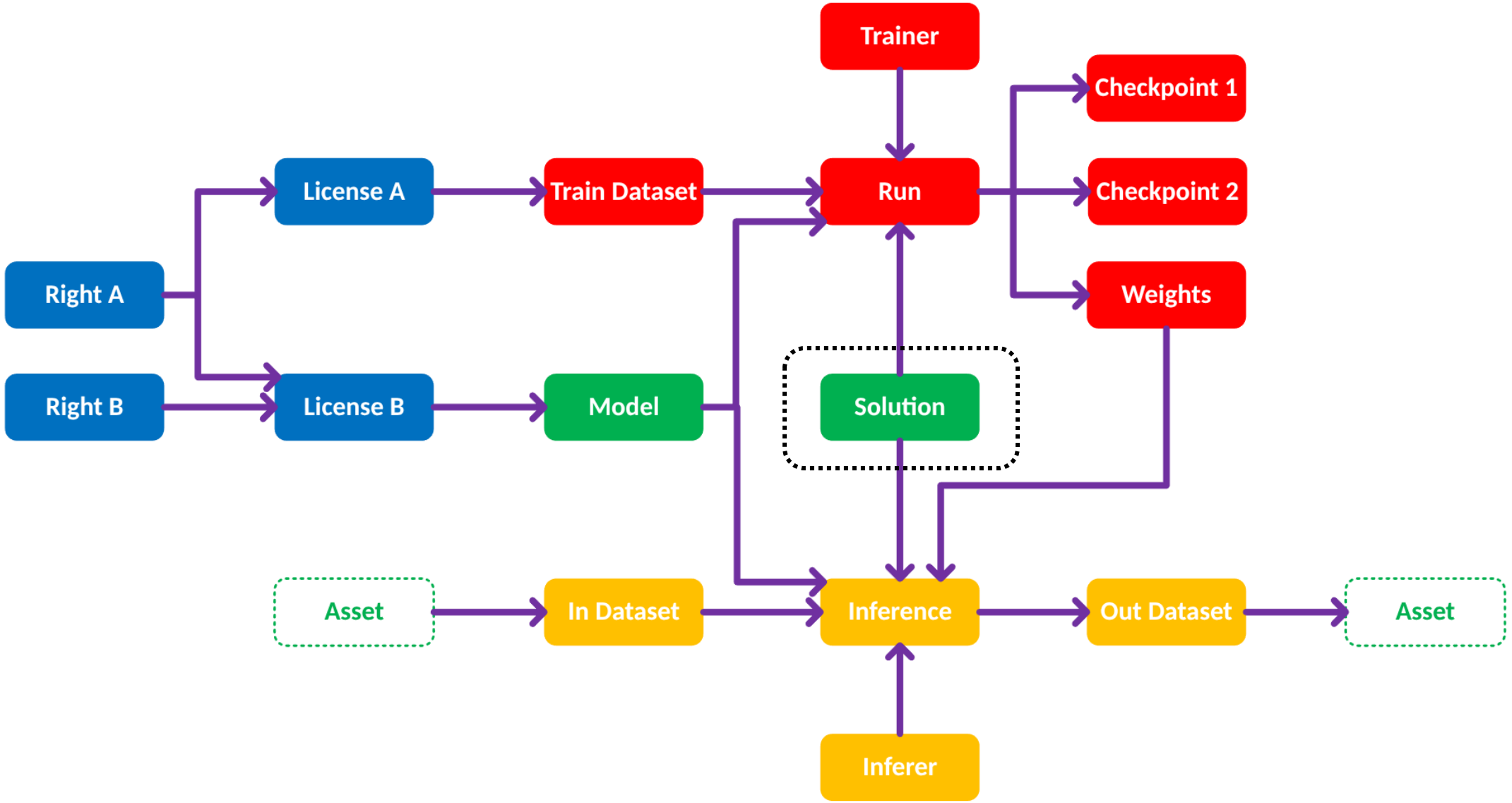
CODE

RMTC/develop

- Branch pushed Jan 28th
- Purpose was for initial collab at TSC level – not a release
- New system in place at Weta to improve cadence
- Moving to main once internal prod use case is met







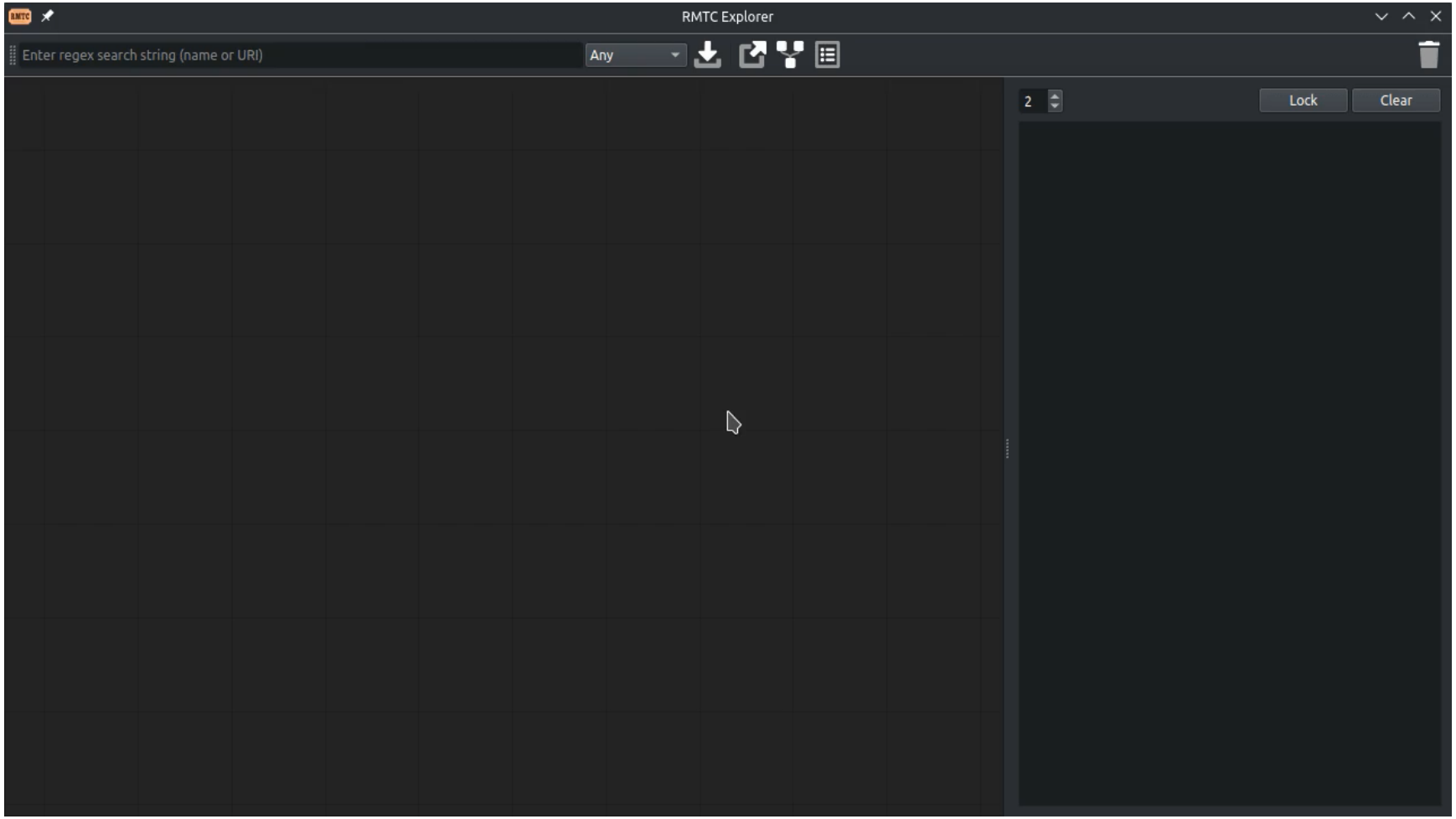
```
rmtc_sys            = System()

# license search
model               = rmtc_sys.get_models("Test Model")[0]
solution            = rmtc_sys.get_solutions("Test Solution")[0]

# training record
weights             = rmtc_sys.create_weights(
    uri             = FileURI("/test/weights.pt")
)
run                 = rmtc_sys.create_run(
    trainer         = trainer,
    solution        = solution,
    name            = "Test Run 1",
    result_weights  = weights,
)

# inference record
result              = rmtc_sys.create_dataset(
    name            = "Result Data",
    uri             = FileURI("/test/result"),
)
inference           = rmtc_sys.create_inference(
    solution        = solution,
    model           = model,
    outputs         = result,
)

# push
rmtc_sys            .push()
```

Construct system & get license / solution

Create a weights record and a run against a solution

Create a result dataset and inference record

Push to remote store



UPDATES (JANUARY+)

- Licenses & rights
  - Guardrails
  - License compliance checks
  - Rights control
- Dataset complexity
  - Aggregations
  - Tabled data
- Inference complexity
  - Shares training schedulers
  - Complex inputs & outputs
- Pipeline
  - Nuke – CAT & TS
  - Watermarking
- Assets
  - Cameras
  - Richer IOs
  - Multi-tensor support
- Wrapping
  - Runners
  - Packagers
- Track & Ops
  - Separate modules
- Refactoring
- Bug fixes

TRAIN TRACK TEST

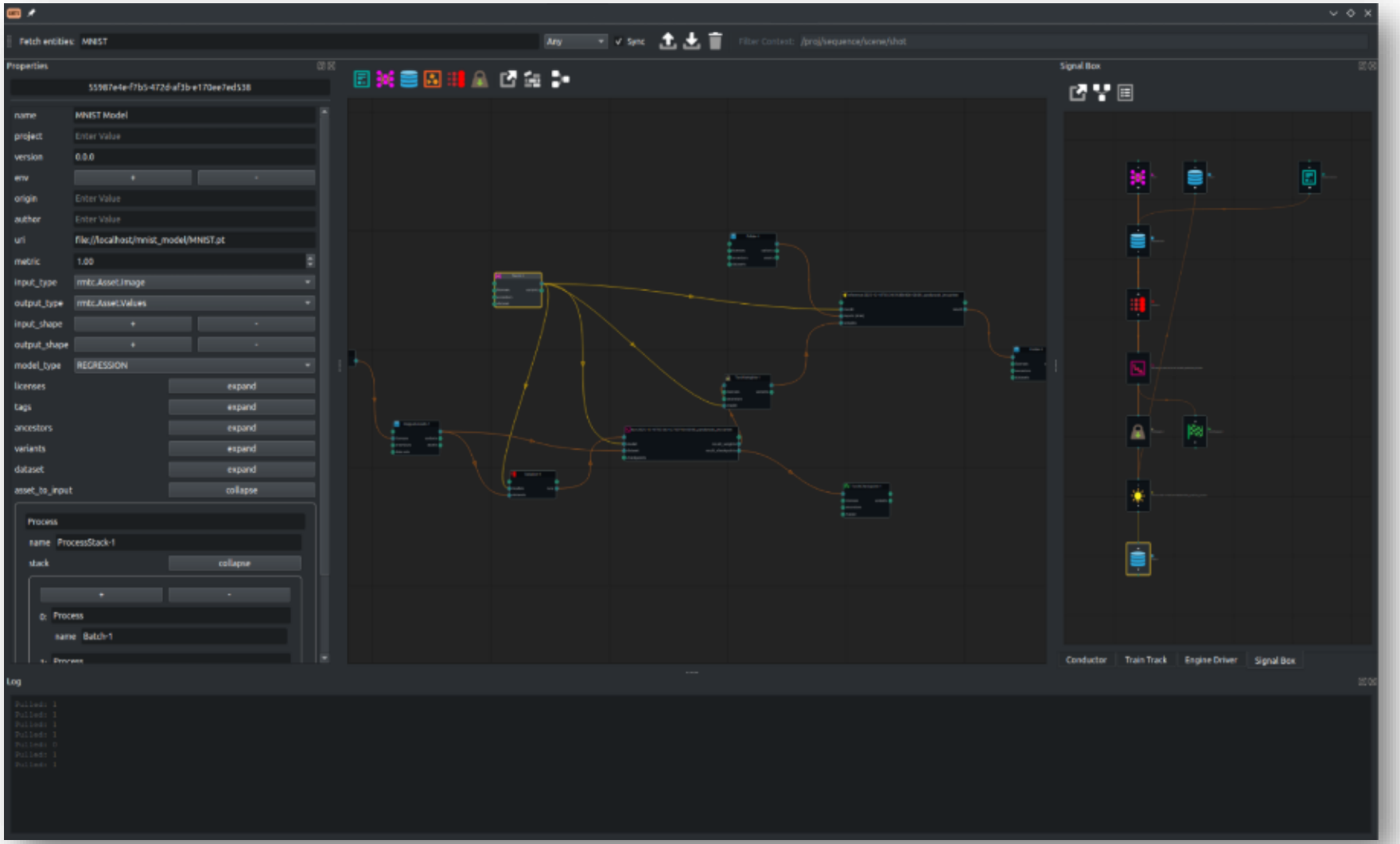
(FEBRUARY- RECAP)

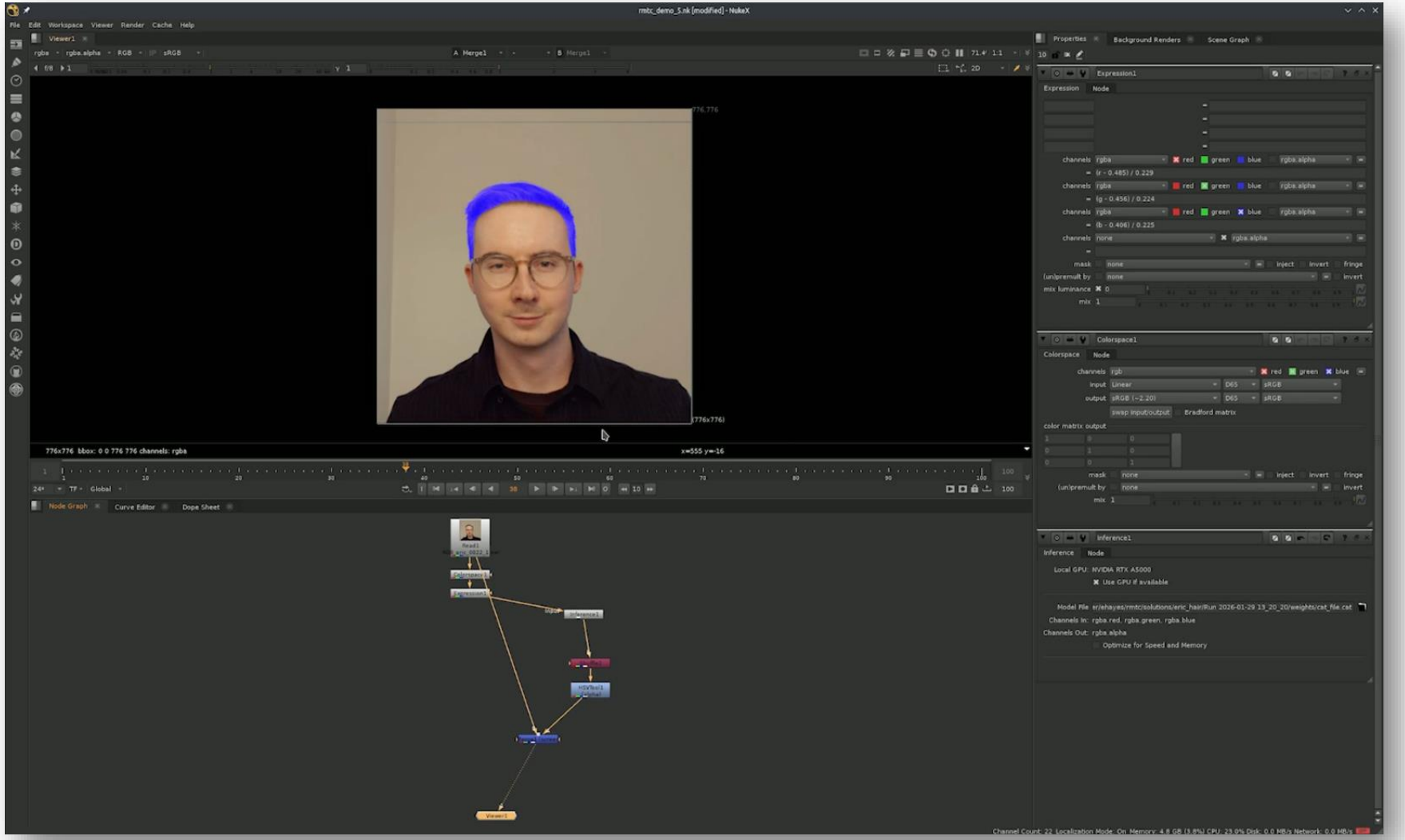
Expose Foundation Model

Refinement Training

Infer in Nuke

Publish & Trace





PRODUCTION TEST

(AUGUST - WIP)

Wrap Complex Model

Guardrails & Production DB

Complex VFX Assets

Time Based Sequences

INPUT EXR SEQUENCE

ESTIMATED DEPTH
w/ CAMERA

```python
# make input image dataset
images              = rmtc_sys.create_dataset(
    artifacts.Dataset,
    name            = "Input Plates",
    columns         = ["plates"],
    uri             = FileURI("/test/plates"),
    io              = folder_io.FolderIO(
        asset_type  = assets.Image,
        asset_io    = image_io.EXR(),
    )
)
depths              = rmtc_sys.create_dataset(
    artifacts.Dataset,
    name            = "Output Depths",
    columns         = ["depths"],
    uri             = FileURI("/test/depth"),
    io              = folder_io.FolderIO(
        asset_type  = assets.Image,
        asset_io    = image_io.EXR(),
    )
)
cameras             = rmtc_sys.create_dataset(
    artifacts.Dataset,
    name            = "Output Cameras",
    columns         = ["cameras"],
    uri             = FileURI("/test/cameras.usda"),
    io              = camera_io.USDCameraSequence()
)
outputs             = rmtc_sys.create_dataset(
    artifacts.Aggregation,
    name            = "Inference",
    order           = Order.COLUMN,
    datasets        = [depths, cameras]
)
```

Construct input dataset – inc. IO

Pre create output datasets for depth & cameras

Construct a column-wise aggregation

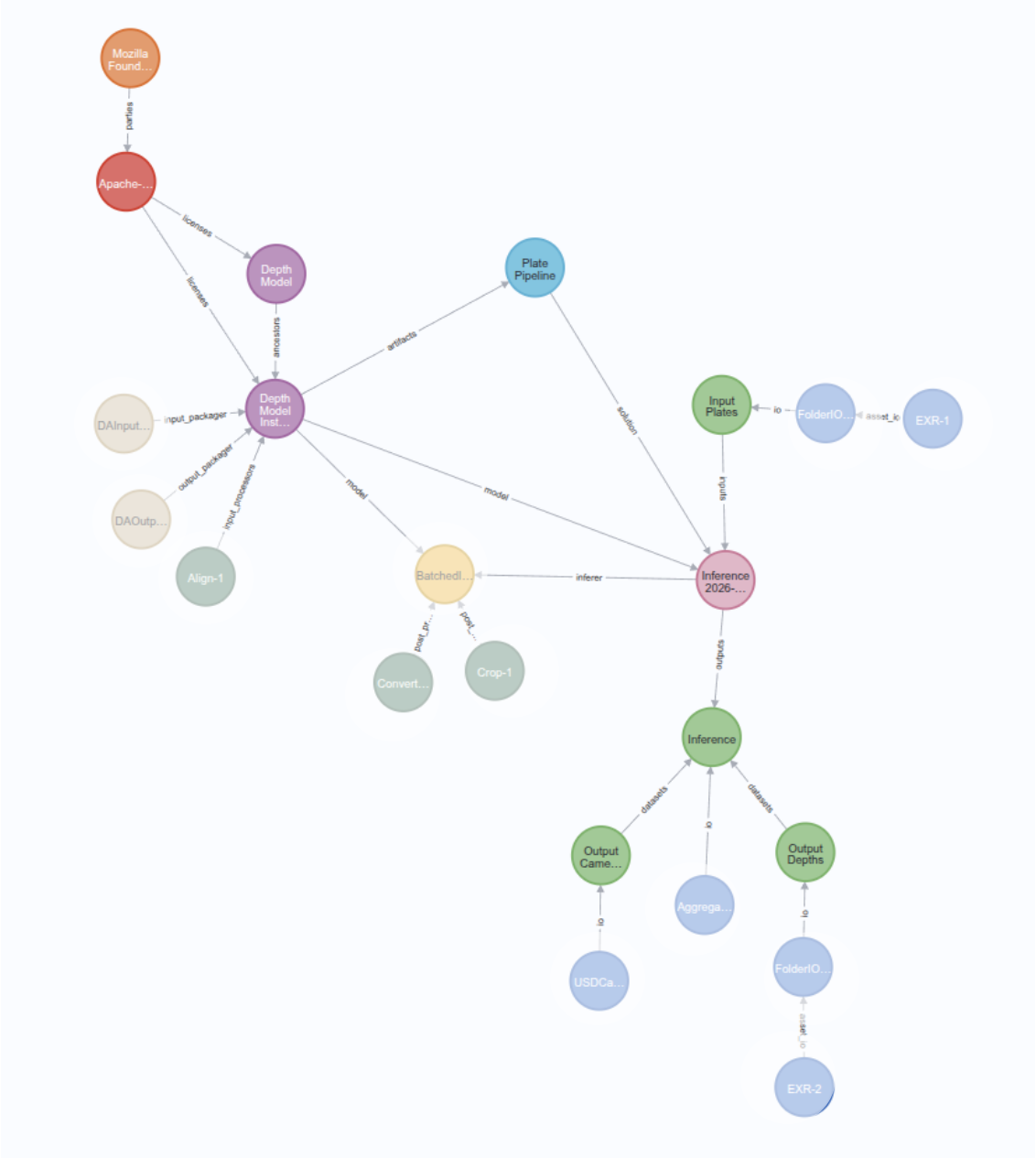
```python
# get solution & best weights
solution            = rmtc_sys.get_solutions("Depth")[0]
weights             = rmtc_sys.get_best_run(solution).result_weights

inferer             = BatchedInferer(
        post_process            = [
            image_procs.Crop(               #depth
                width           = 720,      #px
                height          = 406,      #px
            ),
            camera_procs.ConvertAperture(   #camera
                sensor_width    = 36,       #mm
                sensor_height   = 24,       #mm
                image_width     = 720,      #px
                image_height    = 406,      #px
            ),
        ],
        batch_size  = 5,
        flush_outputs = False,
)

# run a local inference
inference           = rmtc_sys.infer(
    solution        = solution,
    weights         = weights,
    model           = weights.model,
    inputs          = images,
    outputs         = outputs,
)

# write out result and push to DB
rmtc_sys            .write([outputs])
rmtc_sys            .push()
```

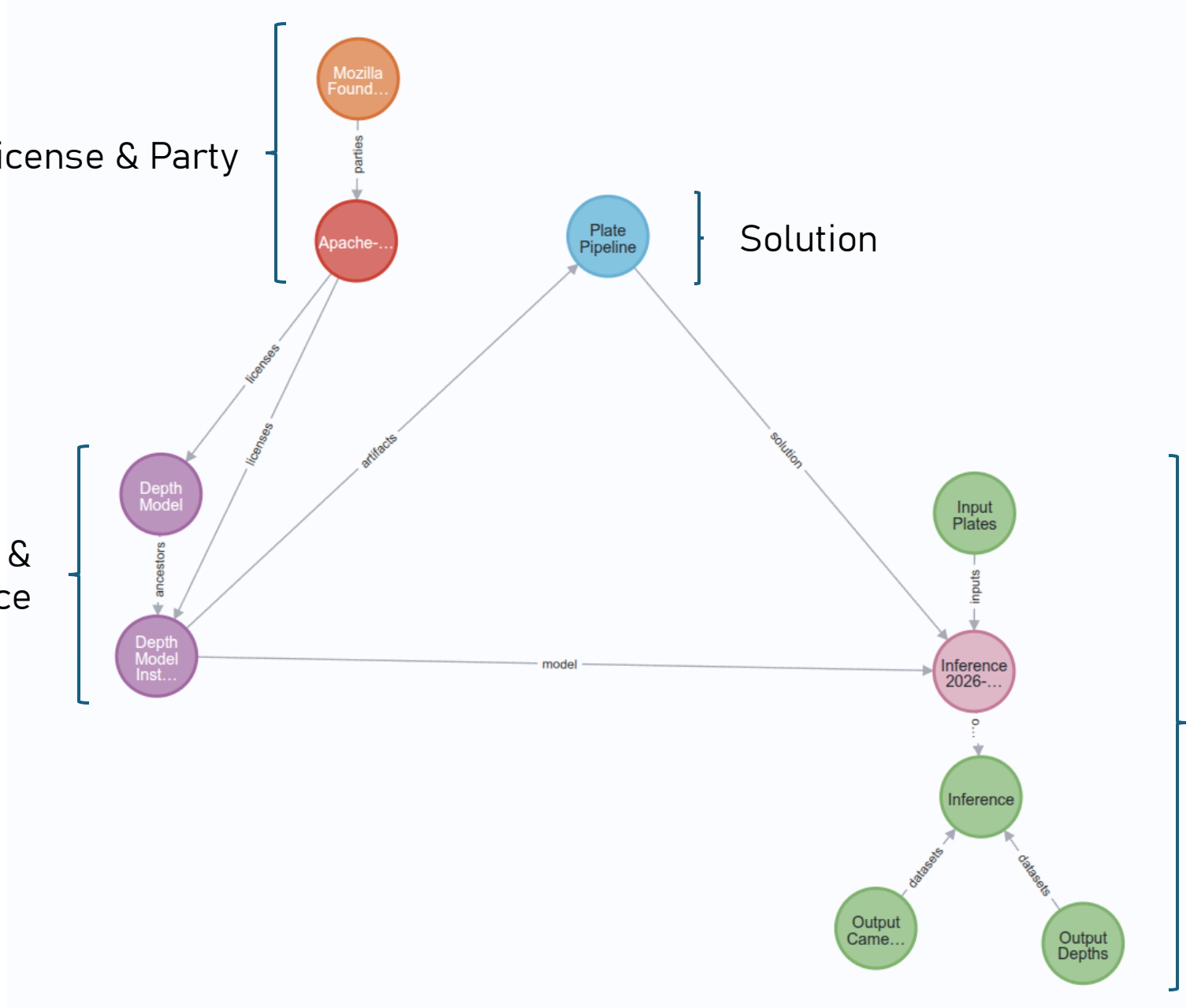
Locate a solution/project & best weights/model

Define how the inference runs
NOTE: this will need to change

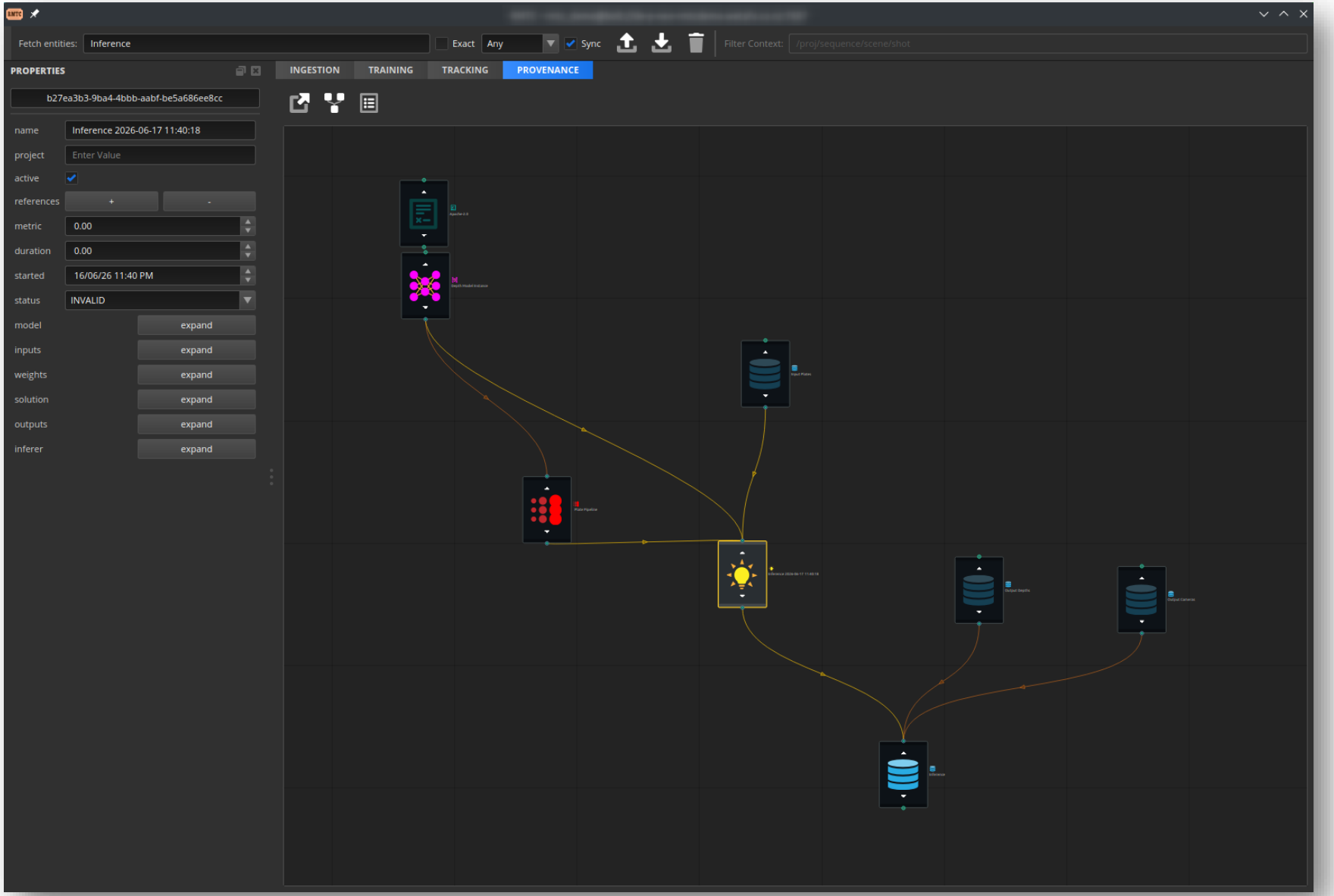
Run inference against inputs, populating outputs

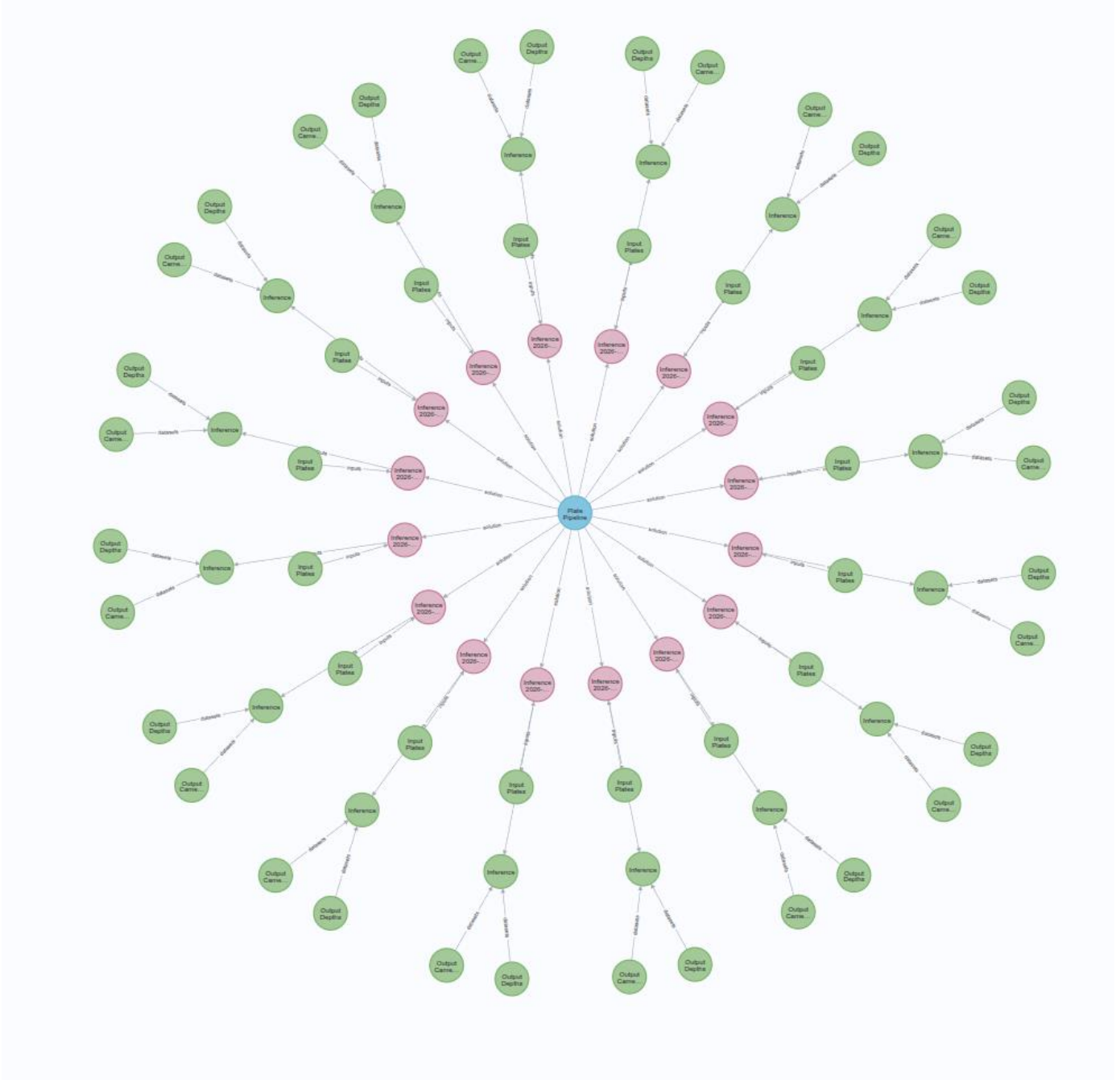
Write out dataset & push to store





ASWF /* ACADEMY SOFTWARE FOUNDATION

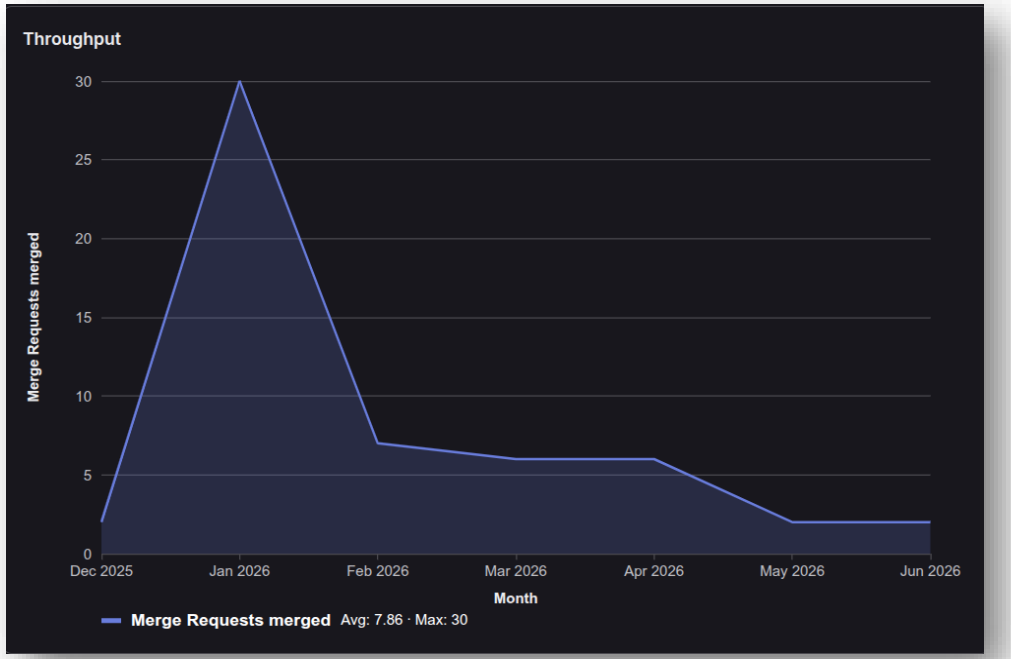




COMMS & COMMUNITY

CONTRIBUTIONS (JANUARY+)

- 14 External PRs
- 27 Internal PRs
- 8 contributors
- Gated on PR update from Weta FX & move to main

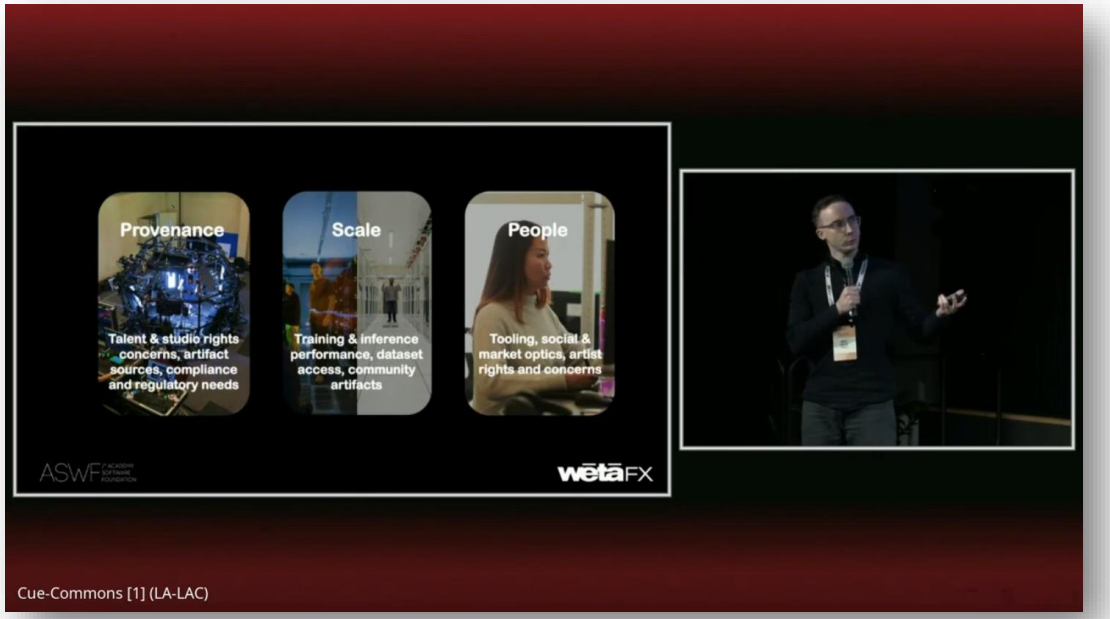


DEV DAYS

- Informal involvement
  - Wiki updated
  - Infrastructure contributions
- Created impetus for ticket review with TSC

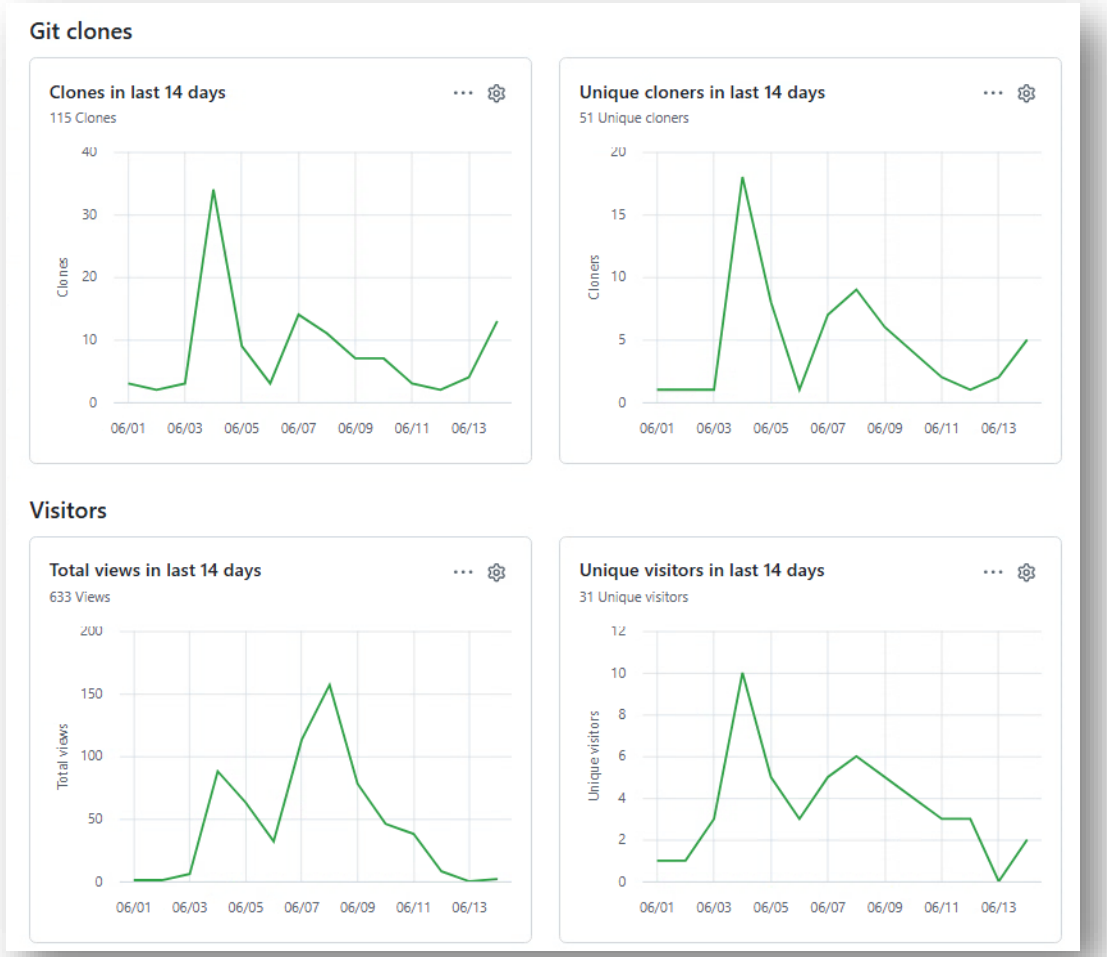
COMMS

- Open Source Forum
  - Received well
  - Opened additional conversations
- Slack
  - Project & TSC channels
  - Discussion limited to TSC agenda
  - Still burgeoning



REPOSITORY

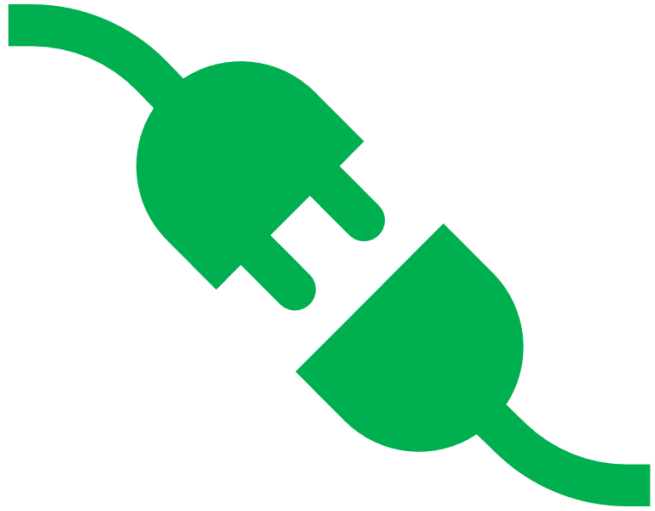
- Main branch not yet merged
- PRs have been on non-systemic work
  - CI/CD
  - Linting/Formatting
  - Documentation
  - Git setup



NEXT

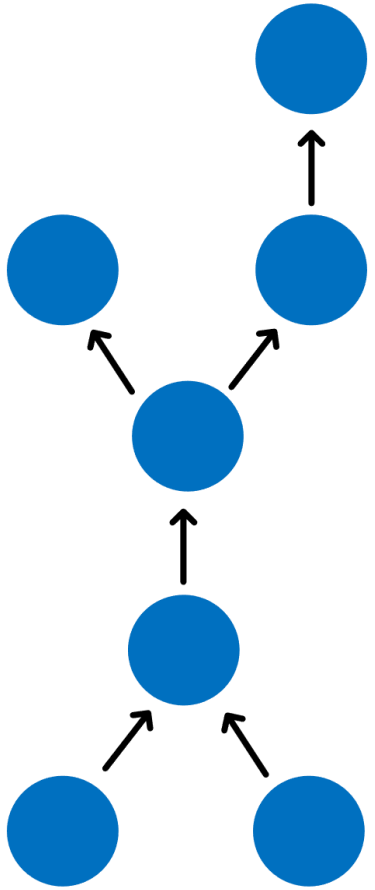
ADOPTION

- Internal integration at Wētā FX
  - 3 ongoing use cases
- Aoption by TSC organizations
- Seeking other adoptees



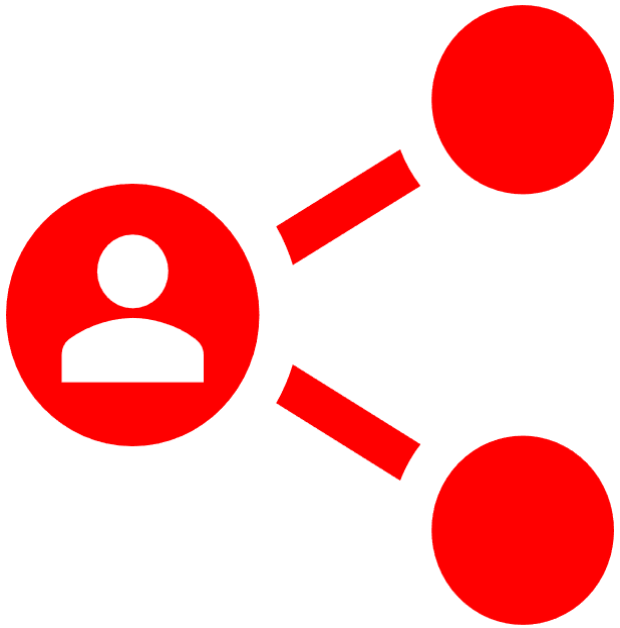
TRACING

- Watermarking
  - Integration of C2PA
  - USD / Apple
- Improved reporting
- Pipeline to RMTC bridge
  - RMTC **does not** trace through CG pipeline



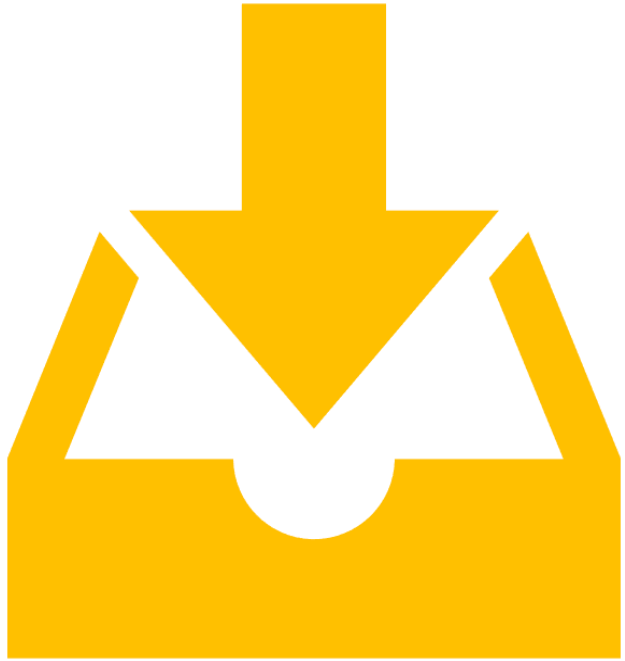
COMMS

- SIGGRAPH
  - Informal presence (Kimball)
- SIGGRAPH Asia
  - TBC
  - Technical Comms / BoF
- WG-ML
  - Talk
  - Juan Buhler representing RMTC



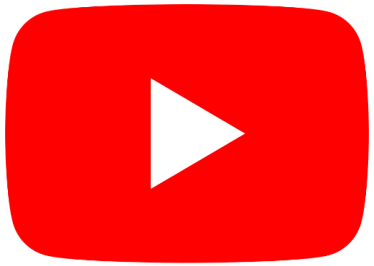
FEATURES

- OpenCue/Deadline scheduler
- Separate tracking & ops DB
- Workflow/DCC plugins
- Time
- Web UI & Server
- Website/Logo



Q&A

APPENDIX



youtube.com/watch?v=dDTYIiJxtOY

```python
# make system
rmtc_sys                = rmtc.System()

# find license
apache2                 = rmtc_sys.get_license("Apache-2.0")

# wrap model
model                   = rmtc_sys.create_model(
    DepthModel,
    name                = f"Depth ({args.name})",
    model_name          = args.name,
    uri                 = FileURI(args.weights),
    version             = Version("1.0.0"),
    licenses            = [apache2],
    packages            = [
        Package("depth-1.0.0"),
    ]
)

# create the solution
solution                = rmtc_sys.create_solution(
    name                = "Demo Pipeline",
    version             = Version("1.0.0"),
    input_types         = model.input_types,
    output_types        = model.output_types,
    description         = "Infers depth & camera from images",
)
solution                .add_artifacts([model])

# push to DB
rmtc_sys                .push()
```

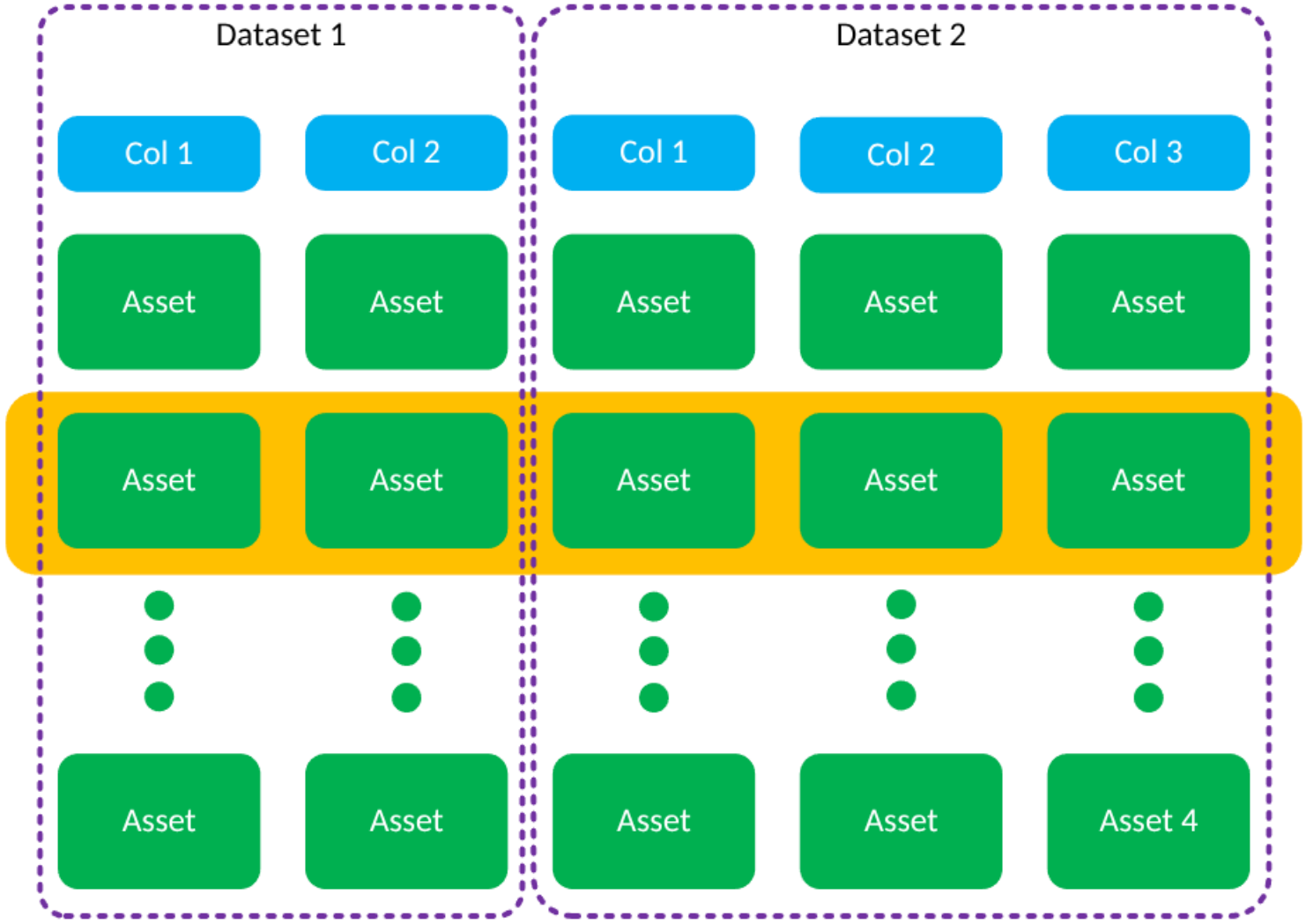
Construct system & get license

Expose custom model – Depth Model derives from RMTC Model, overrides 'run'

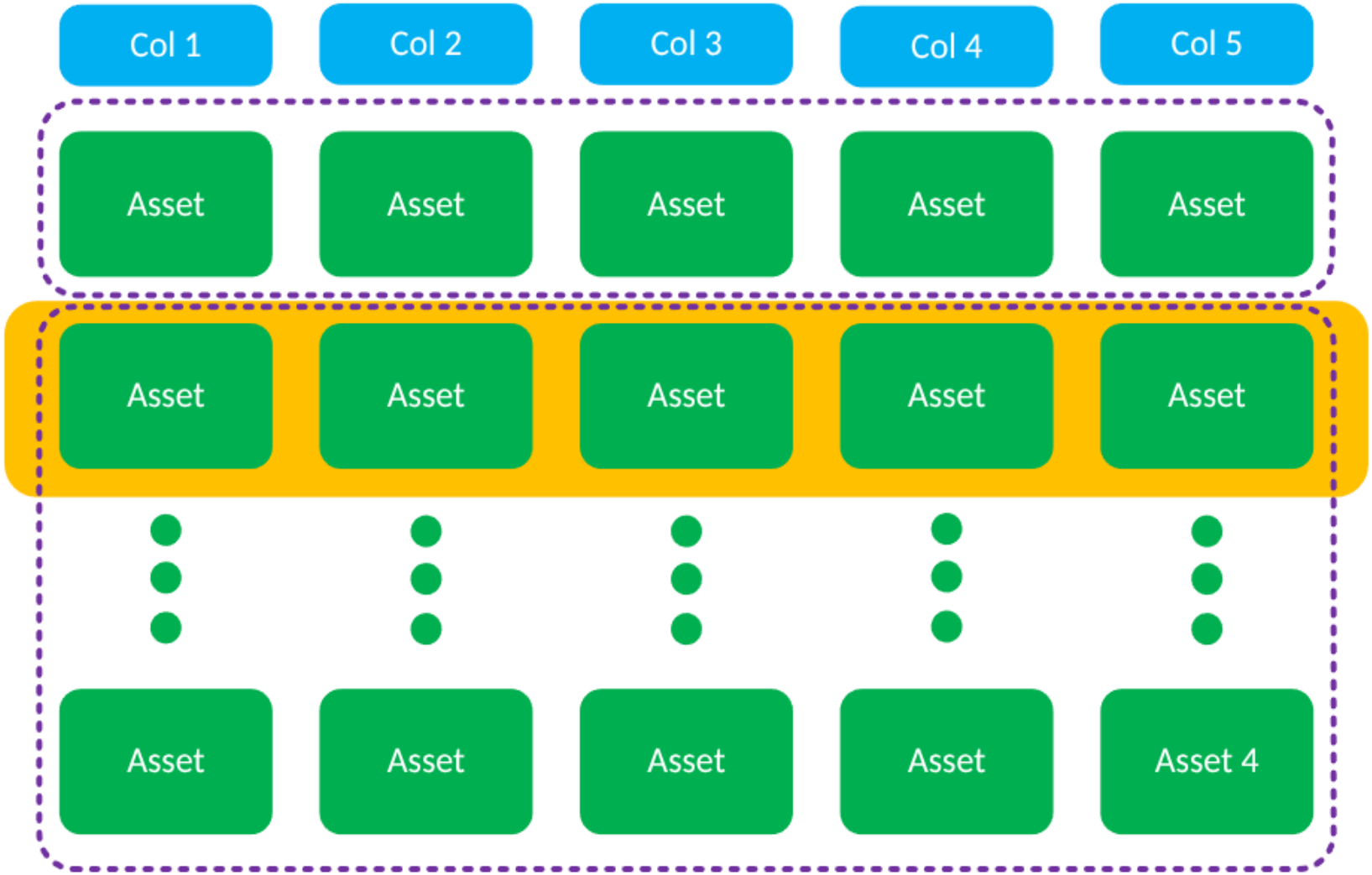
Create 'solution' and attach model

Push to remote store

COLUMN aggregates allow you to combine datasets into virtual tables with rows sampled across



ROW aggregates create a virtual dataset of datasets
with rows populated from a specific dataset




RMTC
The Rongotai Model Train Club
ASWF /* ACADEMY SOFTWARE FOUNDATION
wētāFX