



Enhancing the FPGA synthesis service platform "Gofer"

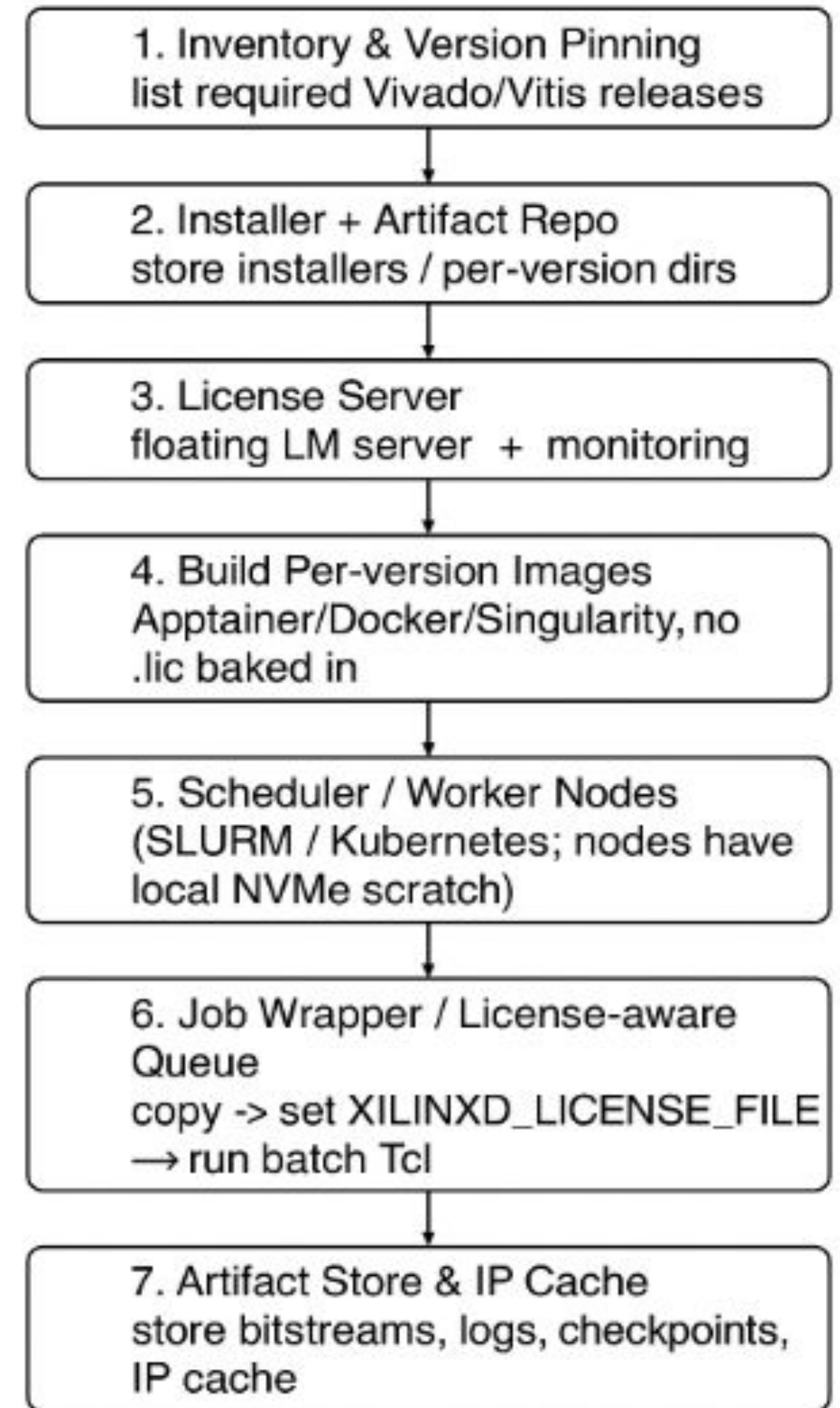
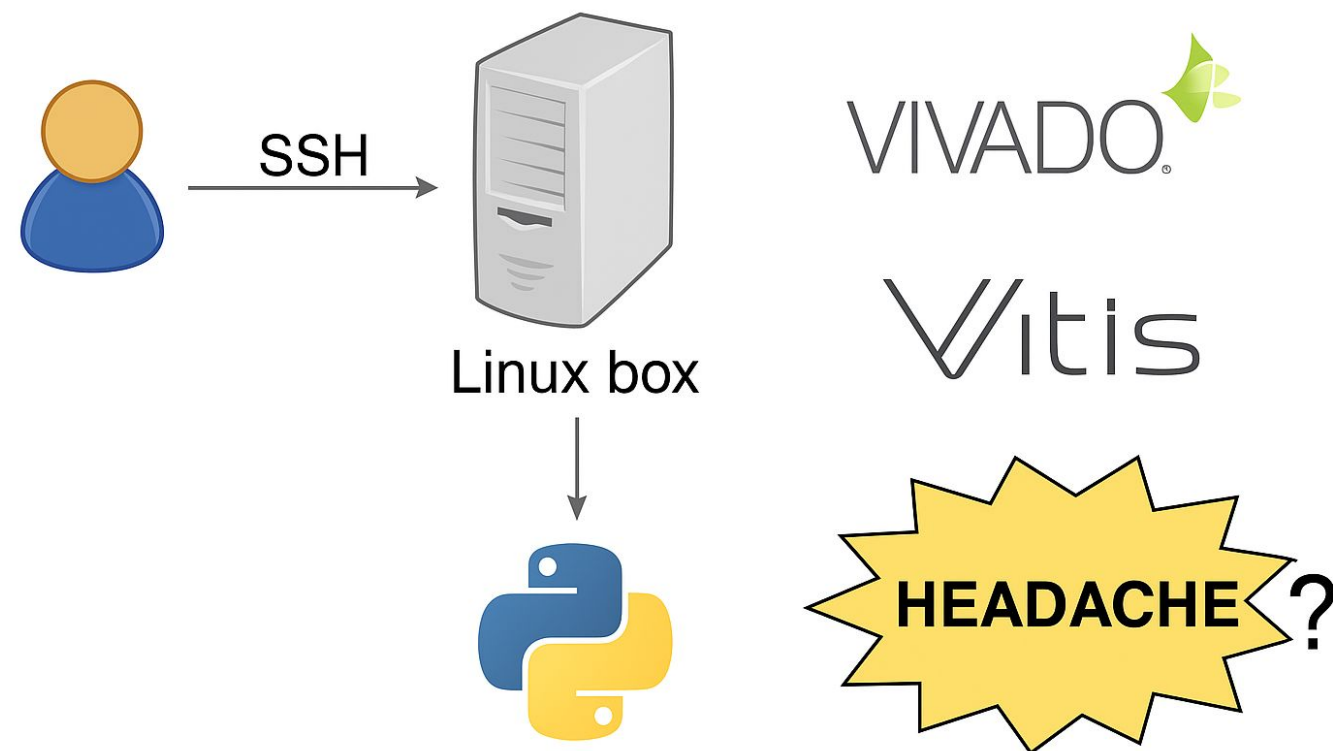
Anushka Bilandani
IIIT Pune

Supervisors: Vladimir Loncar
Dimitrios Danopoulos

Motivation

- Running **FPGA high-level synthesis** locally requires **installing** Xilinx Vivado/Vitis on a powerful Linux machine with ample **memory (>40 GB)**, along with the appropriate **licenses** for logic synthesis.
- Shared remote machines (via SSH) lead to **multitenancy** issues, version mismatches, and **environment inconsistencies** that require constant management

Without Gofer



Our solution: Gofer

Gofer /'goufər/ - an employee who specializes in the delivery of special items to their superior(s). A labourer who is obliged to do menial work. May also refer to a junior member of an organisation who generally receive the most vexing and thankless work

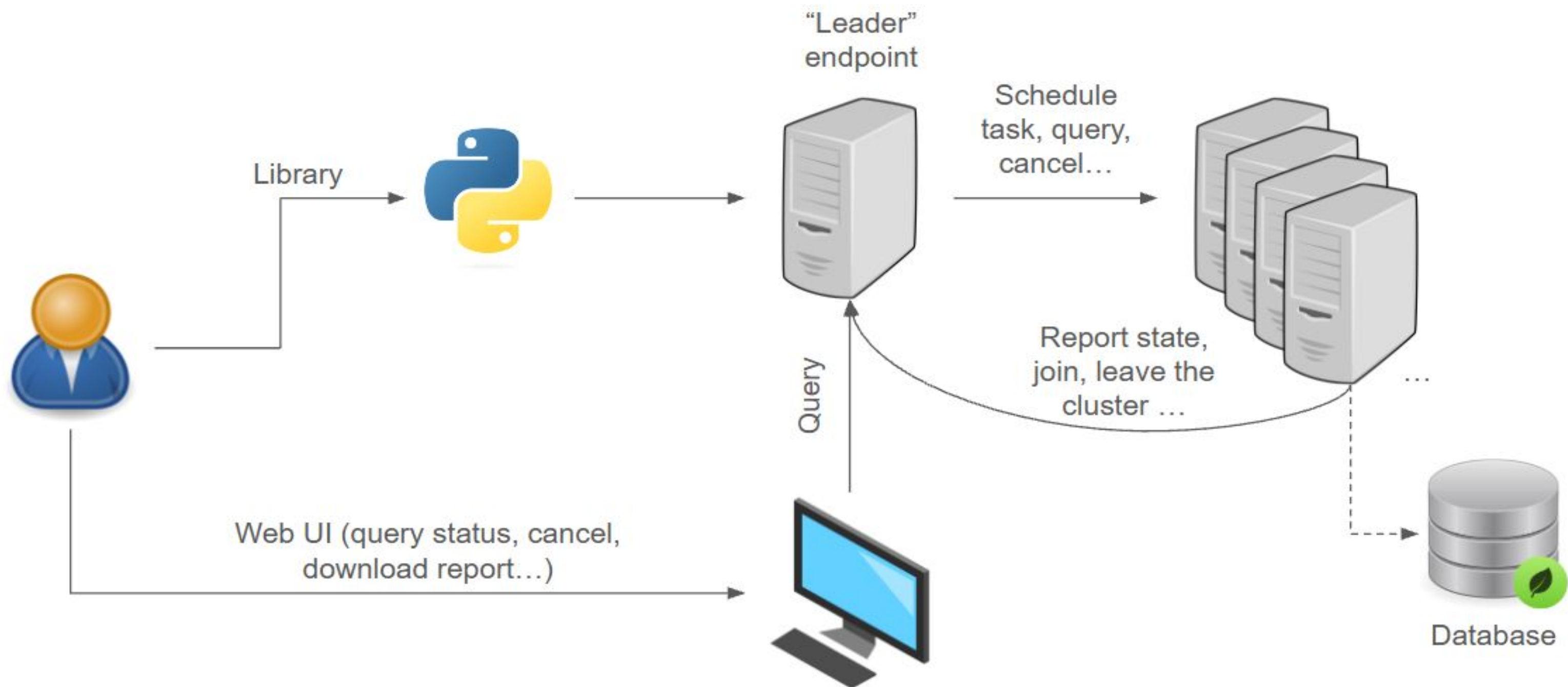
- Gofer is a Python-based synthesis-as-a-service platform that abstracts High-Level Synthesis (HLS) and logic synthesis (e.g., with hls4ml) to scalable cloud backends
- Supports both interactive (“blocking”) and asynchronous (“non-blocking”) execution
- One-line .build() call for single jobs to large-scale synthesis scans, with quotas, access control, and multi-worker scalability.
- Offers a web interface and Python client library for seamless integration into application workflows, enabling easy, reproducible synthesis directly from Python pipelines.

```
{
  "task_id": "129316755e094bf8b7256d7fe7413d12",
  "build_cmd": {
    "command": "vitis_hls -f build_prj.tcl reset=False csim=True synth=True cosim=False",
    "cwd": "\\tmp\\gofer\\tasks\\129316755e094bf8b7256d7fe7413d12",
    "env": null
  },
  "status": "success",
  "task_state": "scheduled"
}
```

```
(gofer-env) [abilanda@ngt-highmem-001 gofer]$ python -c "
from hls4ml_client.api.default_api import DefaultApi
from hls4ml_client.models.build_post_request import BuildPostRequest

api = DefaultApi()
request = BuildPostRequest(
    config_path='/afs/cern.ch/user/a/abilanda/gofer_July10/gofer/test_configs/hls4ml_config.yml'
)

response = api.build_post(request)
base_command = 'vitis_hls -f /afs/cern.ch/user/a/abilanda/gofer_July10/gofer/test_configs/build_prj.tcl'
print(f'{base_command} {options}')ynth=true cosim=false'
vitis_hls -f /afs/cern.ch/user/a/abilanda/gofer_July10/gofer/test_configs/build_prj.tcl reset=false csim=true synth=true cosim=false
```

My work over 9 weeks

What I've worked on

- Mapped tool-specific build features and generated synthesis commands for supported tools like Vivado, Vitis, Quartus etc.
- Designed and implemented a test scheduler to queue jobs and control the number of active concurrent builds.
- Built a Python client to interact with Gofer's REST API, enabling easy job submission, monitoring, and result retrieval.

Purpose

- Automates and accelerates complex firmware builds, repetitive synthesis steps
- Enables multi-user job handling with fair scheduling
- Makes job management and testing easier
- Makes synthesis accessible via web and API tools
- Enables fast interaction through a simple client

My work over 9 weeks

Mapped tool-specific build features and generated synthesis commands for supported tools like Vivado, Vitis, Quartus etc.

POST `/tasks/{tool}/submit` Submit Task

Submit a task for the specified tool

Parameters

Name	Description
tool <small>* required</small> string (path)	<input type="text" value="vitis"/>

Request body required

uploaded_file * required
string(\$binary)

task_name
string

☐ Send empty value

200

Response body

```
{  
  "task_id": "fc65cc61bf1e4c70a8c6589dba9ef9f6"  
}
```

200

Response body

```
{  
  "task_id": "fc65cc61bf1e4c70a8c6589dba9ef9f6",  
  "build_cmd": {  
    "command": "vitis_hls -f build_prj.tcl csim=1 synth=1",  
    "env": {  
      "HOME": "C:\\\\Users\\\\Anushka",  
      "USER": "",  
      "SHELL": "C:\\\\Program Files\\\\Git\\\\usr\\\\bin\\\\bash.exe",  
      "LANG": "en_US.UTF-8",  
      "LC_ALL": "C.UTF-8",  
      "XILINX_VITIS": "C:\\\\Xilinx\\\\Vitis_HLS\\\\2023.1",  
      "XILINX_HLS": "C:\\\\Xilinx\\\\Vitis_HLS\\\\2023.1",  
      "CSIM_DISABLE_GUROBI": "1"  
    }  
  },  
  "status": "success",  
  "task_state": "scheduled"  
}
```


My work over 9 weeks

Designed and implemented a test scheduler to queue jobs and control the number of active concurrent builds.

- Thread Pool Management – Controls parallel task execution using a fixed worker thread limit (e.g., max 5 concurrent builds).
- Task Execution Tracking – Simulates builds via timed TCL scripts and logs timestamps to validate concurrency.
- Test Scenarios – Verifies thread limits, timing accuracy, and shutdown behavior under load.

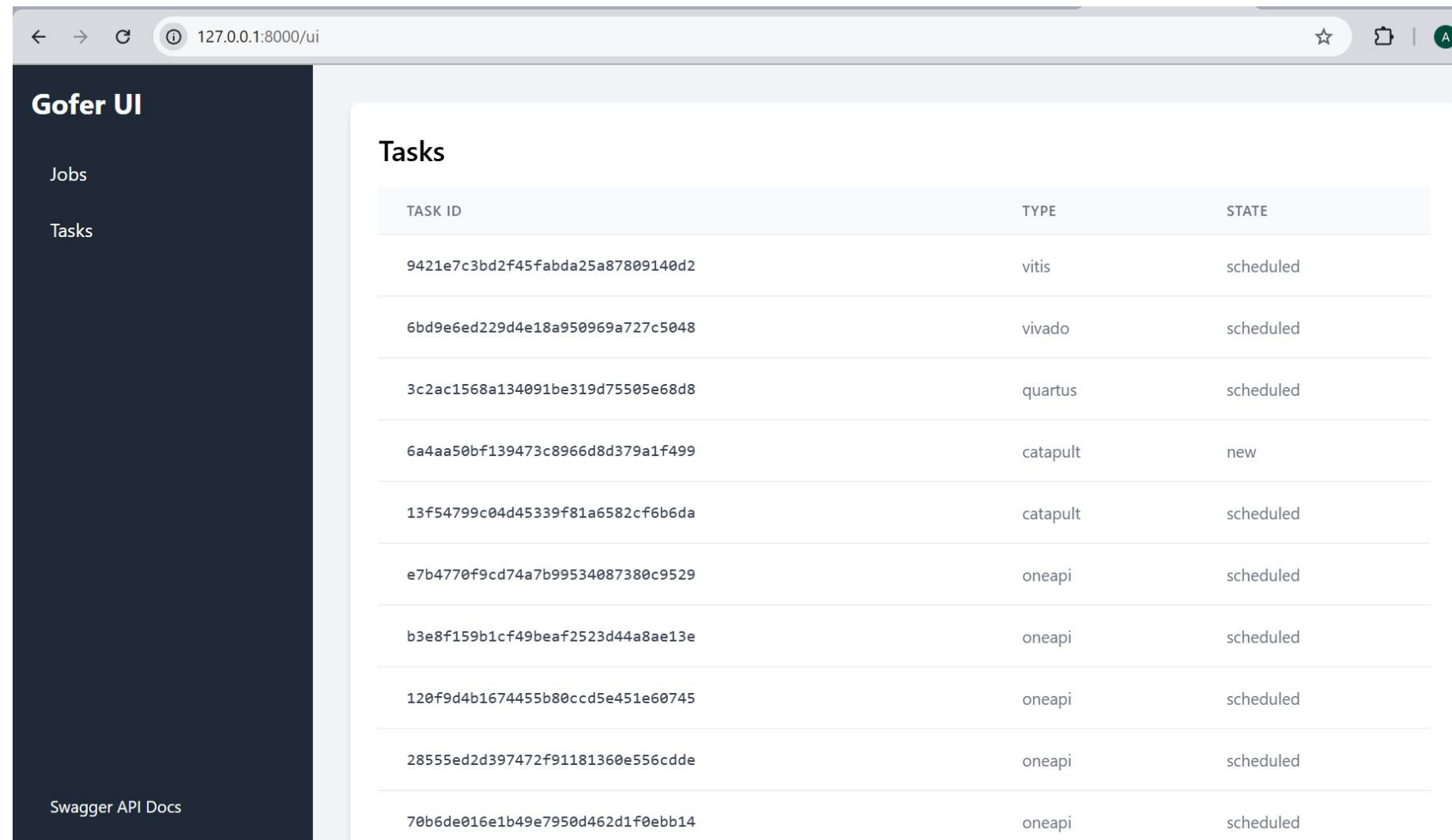
```
> def test_scheduler_waiting(capsys): ...
    set start_time [clock seconds]
    set start_ms [clock milliseconds]
    puts "START task_{i} $start_time.$start_ms"
    flush stdout

    # Simulate work with a simple loop instead of 'after'
    set end_time [expr $start_time + 10]
> while {[clock seconds] < $end_time} {{...
    }}

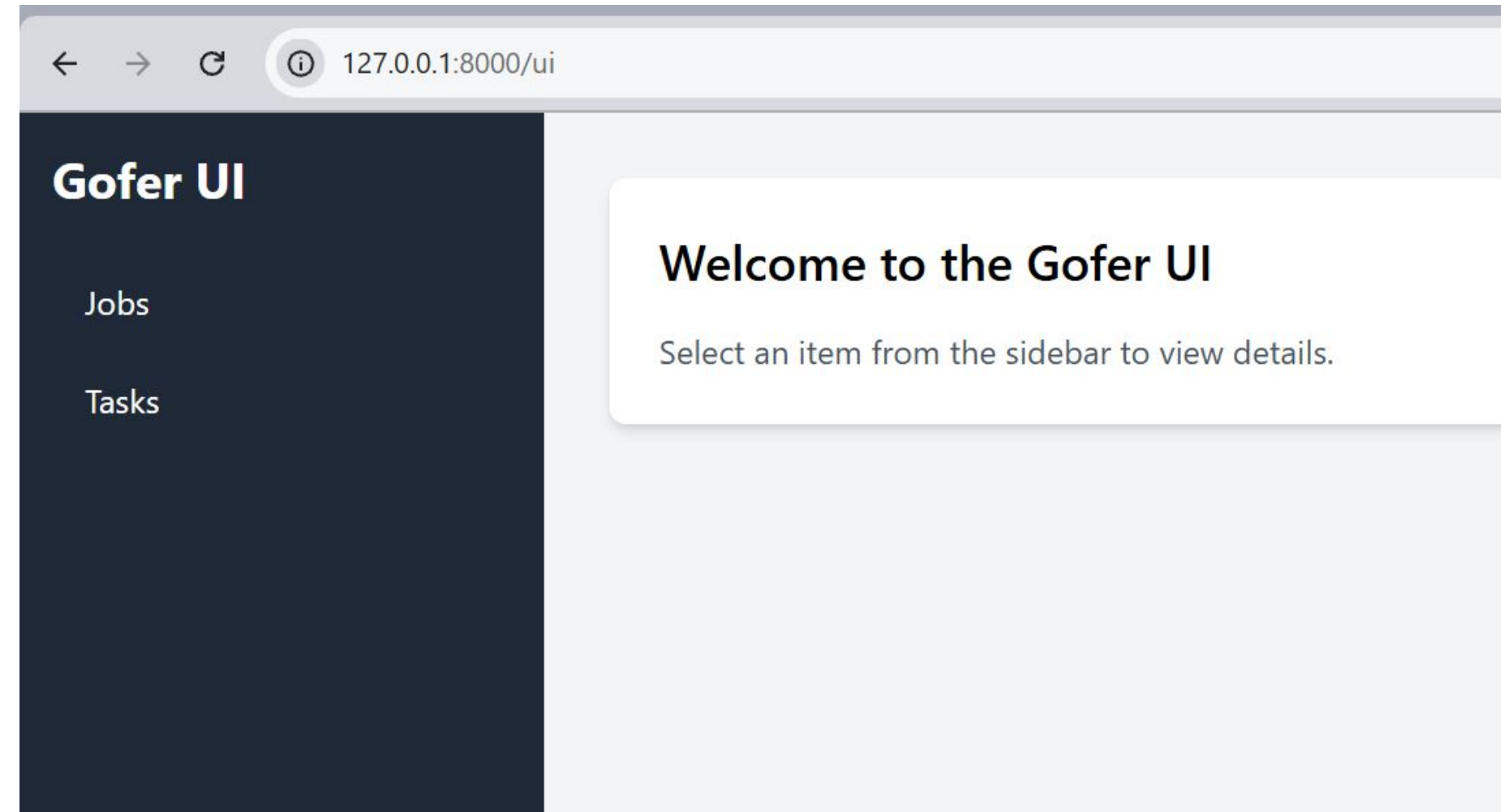
    set end_time [clock seconds]
    set end_ms [clock milliseconds]
    puts "END task_{i} $end_time.$end_ms"
    flush stdout
> ''') ...

> def test_scheduler_with_precise_timing(): ...
    set start_time [clock seconds]
    set start_ms [clock milliseconds]
    puts "TASK_{i}_START=$start_time.$start_ms"
    flush stdout
```


My work over 9 weeks



TASK ID	TYPE	STATE
9421e7c3bd2f45fabda25a87809140d2	vitis	scheduled
6bd9e6ed229d4e18a950969a727c5048	vivado	scheduled
3c2ac1568a134091be319d75505e68d8	quartus	scheduled
6a4aa50bf139473c8966d8d379a1f499	catapult	new
13f54799c04d45339f81a6582cf6b6da	catapult	scheduled
e7b4770f9cd74a7b99534087380c9529	oneapi	scheduled
b3e8f159b1cf49beaf2523d44a8ae13e	oneapi	scheduled
120f9d4b1674455b80ccd5e451e60745	oneapi	scheduled
28555ed2d397472f91181360e556cdde	oneapi	scheduled
70b6de016e1b49e7950d462d1f0ebb14	oneapi	scheduled



Challenges & Problem Solving

- Figuring out how to map hls4ml build features to the service
- Debugged unexpected build failures and input/output mismatches from user-submitted jobs
- Mapping diverse HLS and logic synthesis workflows (Vivado, Vitis, hls4ml, others) into a uniform backend interface while handling differences in input formats, build options, and output artifacts.
- Managing issues across different parts of the system (scheduler, API, client)
- Ensuring the web API and Python client remain in sync as features evolve, while supporting different usage modes

GOT A DIFFERENT ERROR



```
Uninstalling gofer-0.1:
Successfully uninstalled gofer-0.1
Successfully installed gofer-0.1
(gofer-env) [abilanda@ngt-highmem-001 gofer]$ python -m pytest tests/test_scheduler.py -v
===== test session start =====
platform linux -- Python 3.9.23, pytest-8.4.1, pluggy-1.6.0 -- /data/abilanda/miniconda3/
cachedir: .pytest_cache
rootdir: /afs/cern.ch/user/a/abilanda/gofer_July10/gofer
configfile: setup.cfg
plugins: anyio-4.9.0, timeout-2.4.0
collected 3 items

tests/test_scheduler.py::test_scheduler_waiting PASSED
[ 33%]
tests/test_scheduler.py::test_scheduler_with_precise_timing PASSED
[ 66%]
tests/test_scheduler.py::test_scheduler_shutdown PASSED
[100%]

===== 3 passed in 62.24s (0:
(gofer-env) [abilanda@ngt-highmem-001 gofer]$
```


Outcomes & Learnings

- Gained end-to-end understanding of the build flow — from client input to tool-specific command generation and job simulation
- Developed modular components ready for integration into Gofer's production system
- Built a functional job scheduler with concurrency control for active builds
- Created a Python client to submit and track synthesis jobs via REST API
- **Next steps:** strengthen the Python client (e.g., better error handling, retry logic, user feedback)
- Plan to explore a web-based UI to display job status and history
- Improved understanding of scalable service design, multitenancy challenges, and build pipeline abstraction

Thank you!

Email | anushka.vb4@gmail.com