Faculty of Engineering

WasmWeaver: A Framework for Runtime-Aware WebAssembly Program Generation with Reinforcement Learning

Kilian Müller[1], Siddharth Mane[1], Peter Wägemann[2], Norman Franchi[1]

March 18, 2026, SANER 2026, Tool Demo Track

[1]Chair of Smart Electronics and Systems

[2]Systems Software Group

WebAssembly (Wasm) at a Glance

- Portable binary instruction format for heterogeneous platforms
- Compilation target for C/C++/Rust and other languages
- Sandboxed execution with strong safety and isolation
- Near-native performance and fast load times
- Use beyond the browser: servers, edge, plugins, embedded systems

WebAssembly Text Format (WAT) Example

```
(module
  (func $add (param $a i32) (param $b i32) (result
i32)local.get $a
    local.get $b
    i32.add
  )
  (export "add" (func $add))
)
```

Motivation

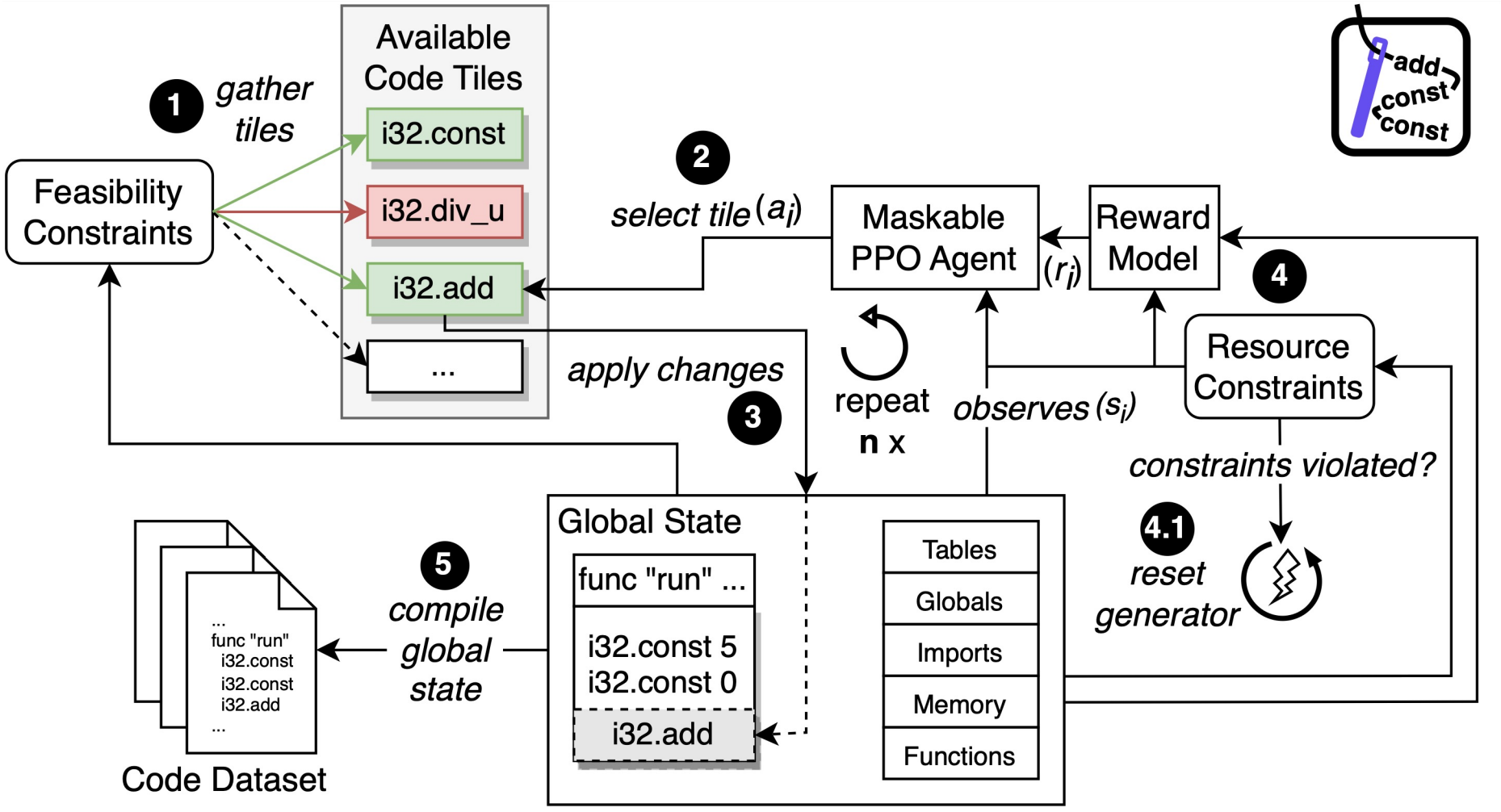
- Currently investigating LLMs for code performance estimation
- Need for pseudo-random, fully executable Wasm programs generated under strict runtime constraints
- Existing fuzzers produce traps, invalid states, or „*unrealistic*“ opcode mixes

  → Therefore, often not suitable for reliable benchmarking
- Current LLM code-reasoning benchmarks lack a precise execution ground truth

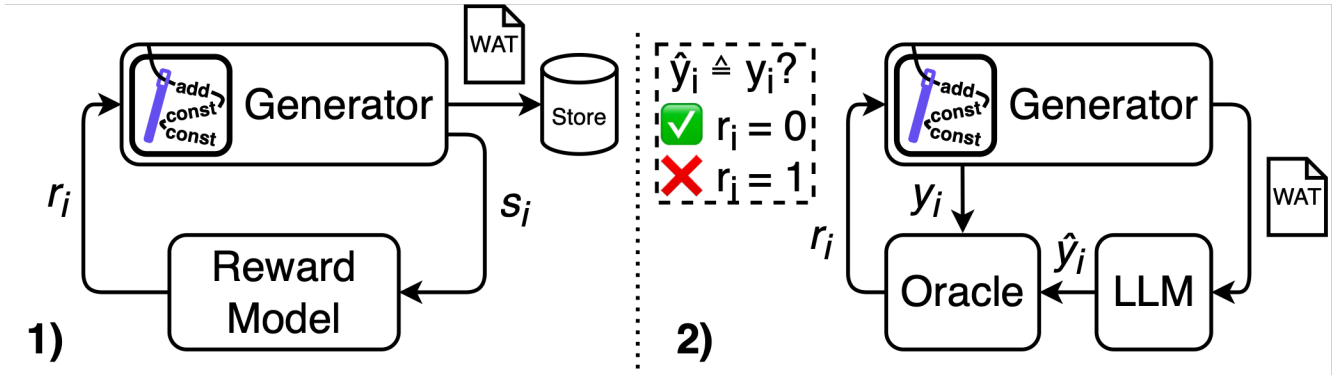
  → Results are easily contaminated or misleading
- Desire for adversarial feedback loops where the generator actively searches for cases that break LLM reasoning

Goal: Systematically expose weaknesses in stack reasoning and control-flow analysis in LLMs

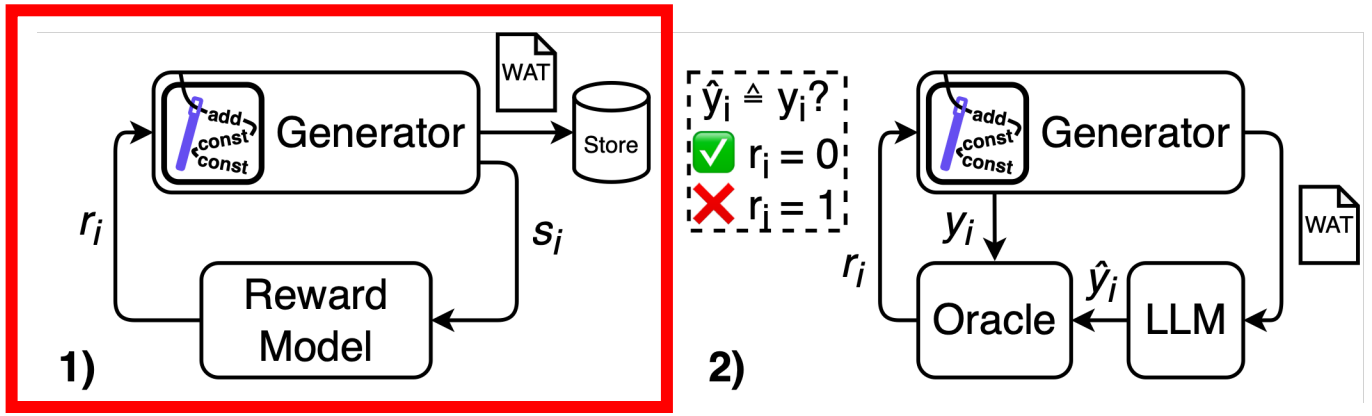
Setup


Available Code Tiles
1
gather tiles
Feasibility Constraints
i32.const
i32.div_u
i32.add
...
2
select tile (a_i)
Maskable PPO Agent
(r_i)
Reward Model
add
const
const
4
Resource Constraints
apply changes
3
repeat n x
observes (s_i)
constraints violated?
4.1
reset generator
Global State
func "run" ...
i32.const 5
i32.const 0
i32.add
Tables
Globals
Imports
Memory
Functions
5
compile global state
...
func "run"
i32.const
i32.const
i32.add
...
Code Dataset

Use Case Examples



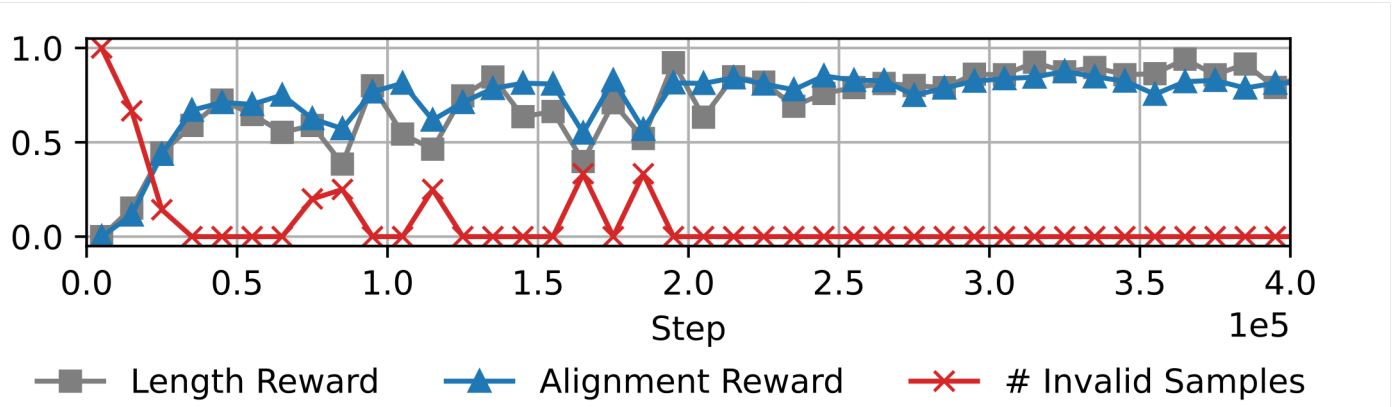
Use Case Examples



$$R_{\text{total}} = \begin{cases} -1, & \text{failed} \\ R_{\text{align}} + R_{\text{length}}, & \text{otherwise} \end{cases}$$

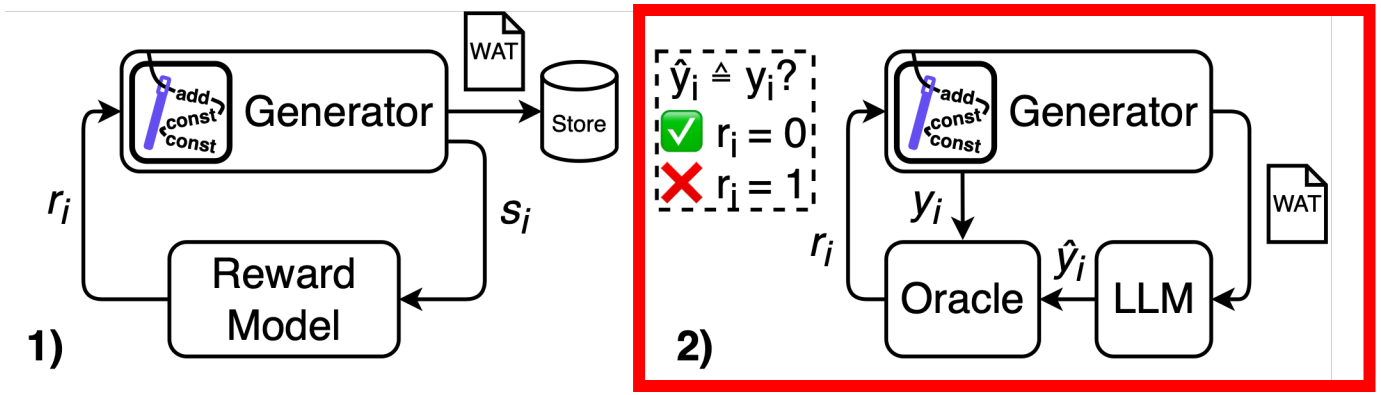
Use Case 1: DRL-Driven Code Generation

```python
import gymnasium as gym
from sb3_contrib import MaskablePPO
from core.environment import WasmWeaverEnv
from drl.rewards import SimpleRewardFunction
from drl.extractor import SimpleFeatureExtractor
from experiments.training.policy import CustomMaskablePolicy
# Register and create the WasmWeaver Gymnasium environment
gym.register(id="gymnasium_env/WasmWeaverEnv-v0", entry_point=W

env = gym.make(
    "gymnasium_env/WasmWeaverEnv-v0",
    constraints=[...],
    reward_function=SimpleRewardFunction(...),
    verbose=True,
)
# Define the PPO policy and feature extractor
policy_kwargs = dict(
    features_extractor_class=SimpleFeatureExtractor,
    net_arch=dict(pi=[512, 256], vf=[512, 256])
)
# Create a Maskable PPO agent
model = MaskablePPO(
    CustomMaskablePolicy,
    env,
    policy_kwargs=policy_kwargs,
    gamma=1.0,
    ...
)
# Train the agent
model.learn(total_timesteps=10_000_000, callback=[...])
model.save("drl_generator_experiment_ppo_wasmweaver")
```



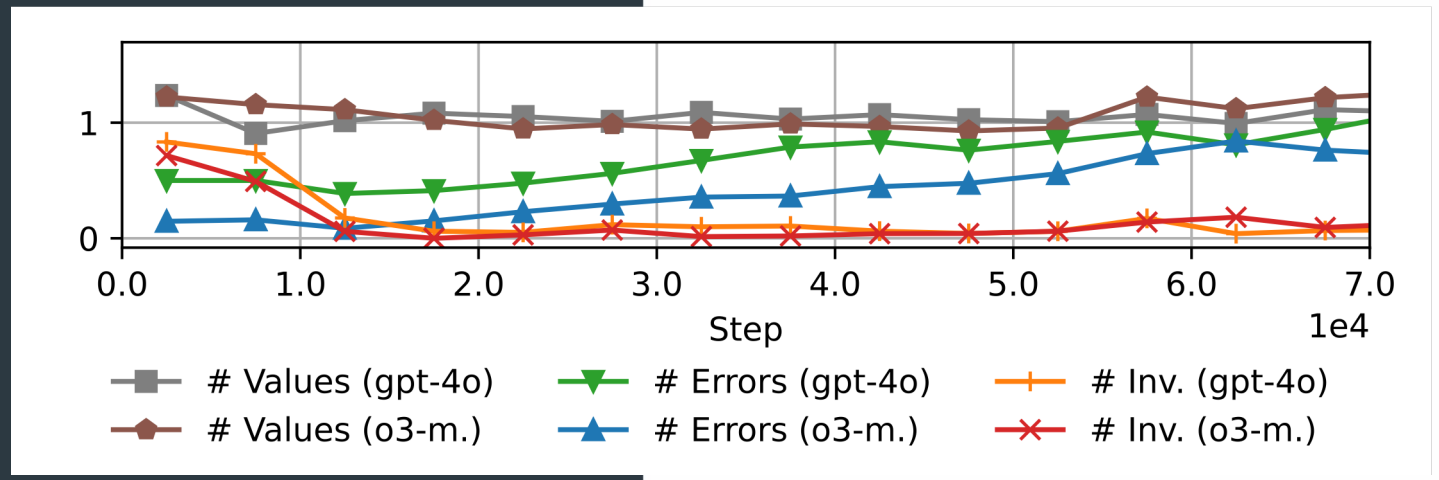

```
    "step": 23685,
    "trace": [
        "START_GlobalGet",
        ...,
        "END_BrTile"
    ],
    "wat_str": "(module\n(type $sig_run (func))\n(import \"env\" \"memory\" (memory 1))...",
    "reward": 1.5618038116194843,
    "length_reward": 0.9013499503829382,
    "used_fuel": 17,
    "target_fuel": 18.97028605652855,
    "module_reward": 0.6604538612365463,
    "good_samples": 92,
    "bucket_reward": 0.9679579299377897,
    "dynamic_depth_reward": 0.4560960551626635,
    "depth_target": 4.652284626691631
}
```

Use Case Examples



Use Case 2: LLM-in-the-Loop (Stack Reasoning)

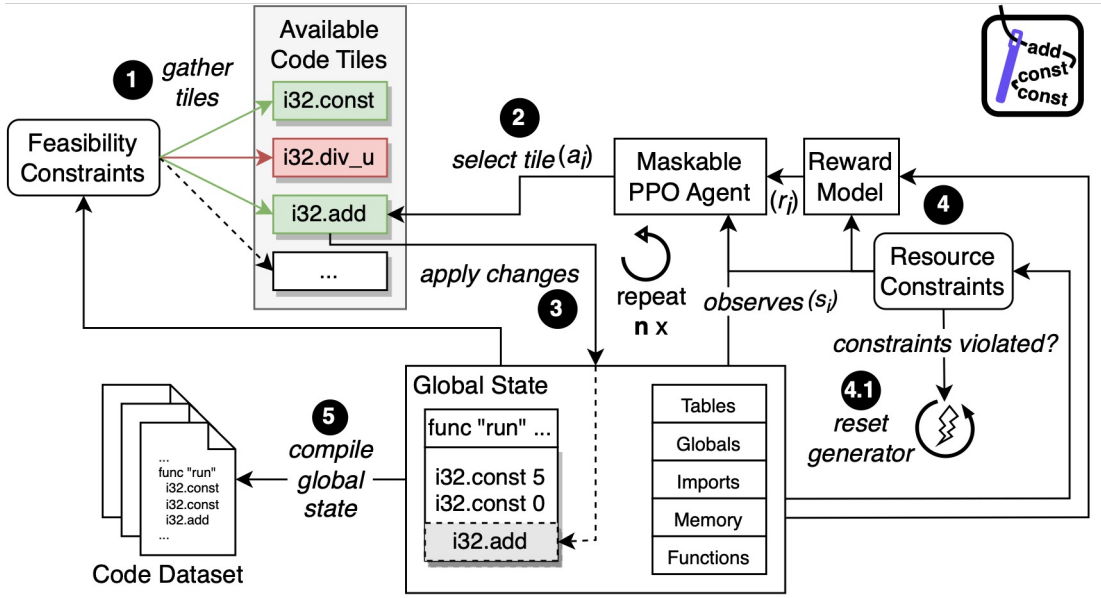
```python
...

# Register the WasmWeaver environment
gym.register(id="gymnasium_env/WasmWeaverEnv-v0",
             entry_point=WasmWeaverEnv)
# Environment configured for stack-effect prediction via LLM
env = gym.make(
    "gymnasium_env/WasmWeaverEnv-v0",
    constraints=[ByteCodeSizeConstraint(0, 1000),
                 FuelConstraint(0, 50)],
    post_processor_types=[StackInspectorPostProcessor],
    forbidden_instruction_name_tokens=[...],
    reward_function=SimpleRewardFunction(
        "STACK_EFFECT_EXPERIMENT_samples",
        stack_reward=True,
        flag_reward=False,
        model=O3Mini(),   # LLM performing stack-value prediction
    ),
    verbose=True,
)
# PPO feature extractor and network definition
policy_kwargs = dict(
    features_extractor_class=SimpleFeatureExtractor,
    net_arch=dict(pi=[512, 256], vf=[512, 256]),
)
# Maskable PPO agent
model = MaskablePPO(
    CustomMaskablePolicy,
    env,
    policy_kwargs=policy_kwargs,
    gamma=1.0,
    ent_coef=1e-3,
    device="mps",
    ...
)
# Train against the LLM
model.learn(total_timesteps=500_000, callback=[...])
model.save("STACK_EFFECT_EXPERIMENT_ppo_wasmweaver")
```




```
(module
  (func (export "run")
    i32.const 2
    i32.const 3
    i32.add
    i32.const 2
    i32.mul
    f32.const 1.5
    ;;INSPECT i32(10), f32(1.5)
    f32.const 2.5
    f32.add
    drop
    drop
  )
)
```

Summary

- Generates fully executable Wasm under hard fuel/size bounds
- Produces *„realistic“* and controllable program structures via DRL
- Enables LLM-in-the-loop adversarial stress testing of code reasoning
- Produces unlimited, leak-free, fully labeled custom benchmarks



Q&A