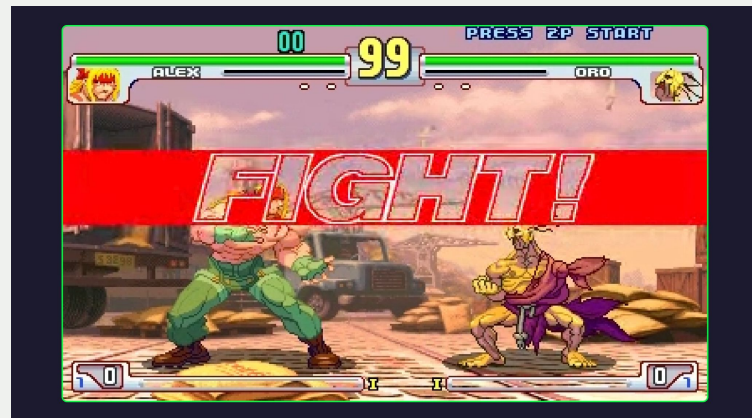


Fighting **RL-trained LLMs** in Street Fighter III: 3rd Strike **on Modal**

Andrew Hinh - DevRel



Outline

01 Web Game

02 Real-Time Inference

03 RL ♥ Sandboxes

04 Demo!

Web Game



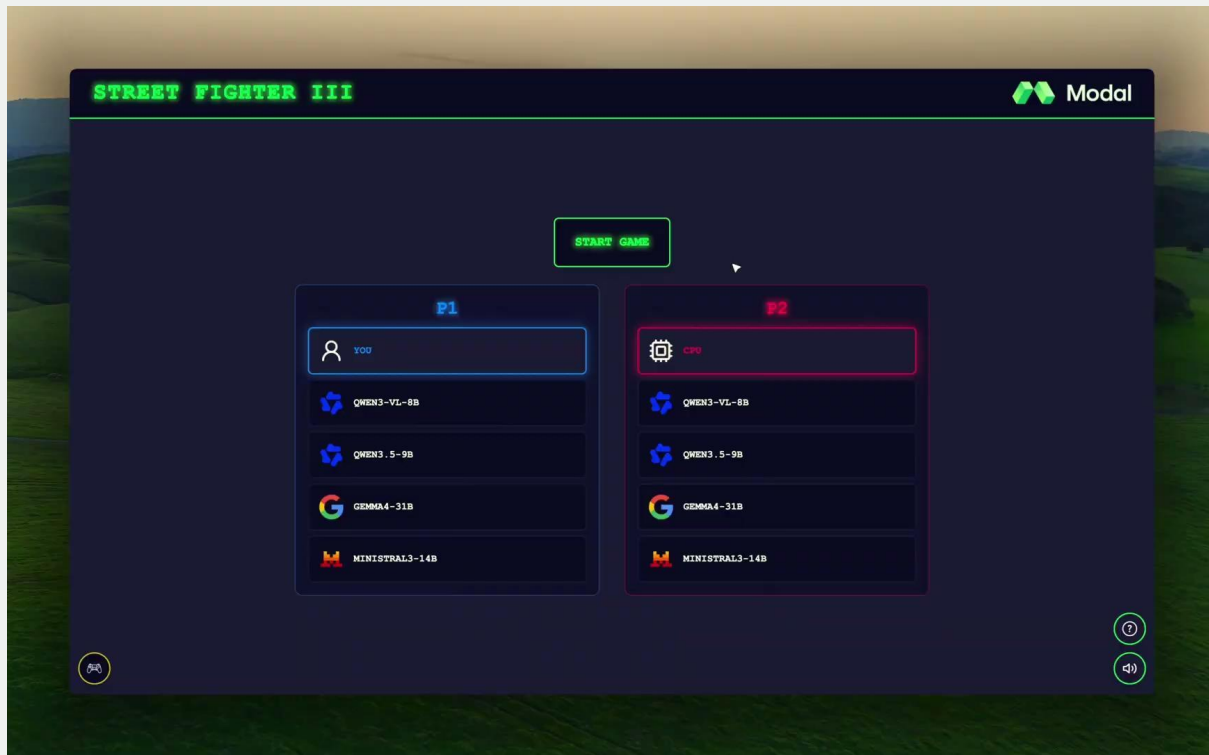
Web Game

```
@app.server(port=GAMEPLAY_PORT, max_concurrency=3, secrets=[turn_credentials])
@modal.sessioned()
class Web:
    @web_app.websocket("/ws/{game_id}")
    async def signaling(websocket):
        pc = RTCPeerConnection(turn_servers)
        pc.addTrack(GameVideoTrack(emulator_frames))
        pc.on("datachannel", lambda channel: channel.on("message", enqueue_input))
        offer = await websocket.receive_json()
        await pc.setRemoteDescription(offer)
        await websocket.send_json(await pc.createAnswer())

@modal.asgi_app(custom_domains=["sf3.modal.dev"])
def launcher():
    session = Web.sessions.start()
    return RedirectResponse(f"{Web.get_url()}/?modal_session_token={session.token}")
```

Web Game

Sticky Sessions!



Real-Time Inference

```
def create_messages(me, opponent, frame, recent_moves):
    system = "\n".join([
        f"You are {me.character} fighting {opponent.character} ...",
        "Moves:",
        *available_moves(me),
        f"Your last moves: {'', '.join(recent_moves[-4:]))}.\"",
        "Respond with only the name of your next move.",
    ])
    return [
        {"role": "system", "content": system},
        {"role": "user", "content": [image(frame), "Your next move is:"]},
    ]
```

Real-Time Inference

```
BUTTONS = ["Left", "Right", "Up", "Down", "LP", "MP", "HP", "LK", "MK", "HK"]
BIT = {name: 1 << i for i, name in enumerate(BUTTONS)}

COMBOS["Ken"]["Hadouken"] = [
    BIT["Down"],
    BIT["Down"] | BIT["Left"],
    BIT["Left"] | BIT["MP"],
] # -> (8, 9, 33)

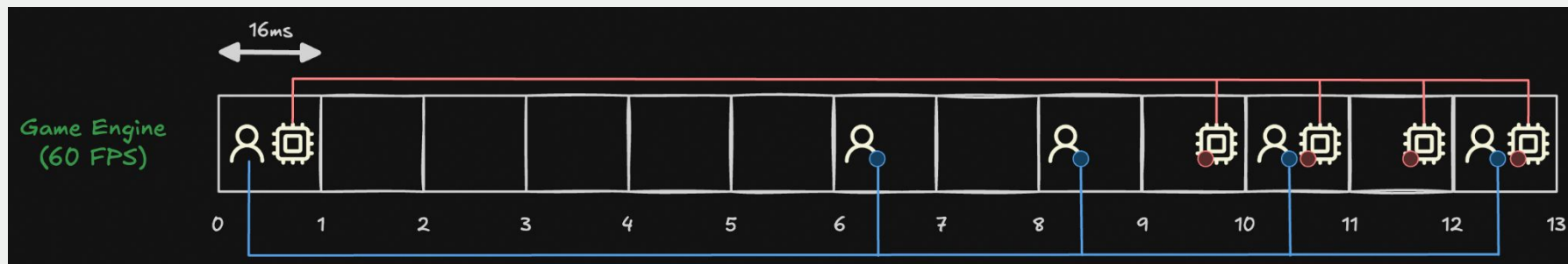
def mirror(frames):
    return [swap_bits(f, BIT["Left"], BIT["Right"]) for f in frames]
```



Real-Time Inference

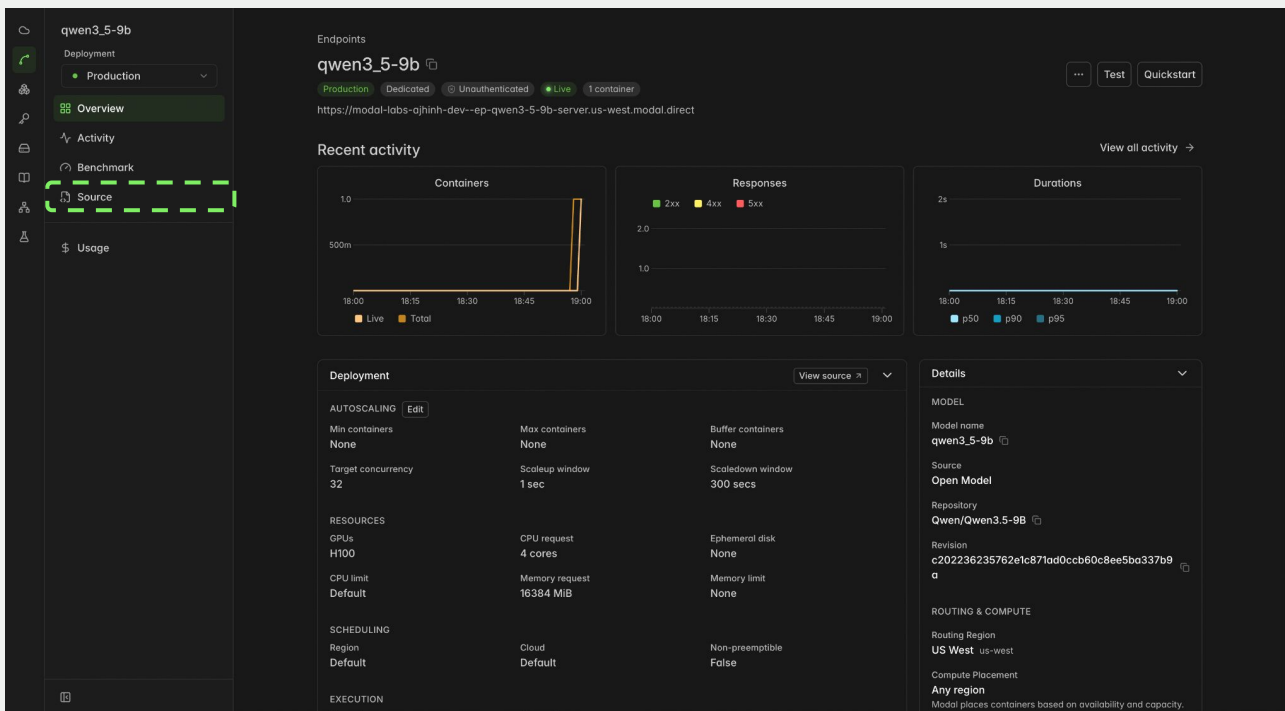
```
env = create_environment(config)
observation, recent = env.reset(), deque(maxlen=4)
while not done:
    move = generate_move(observation["frame"], recent)
    buttons = COMBOS[character][move]
    delay = [0] * model_reaction_frames(buttons, INFERENCE_LATENCY_MS)
    for bitmask in delay + buttons:
        observation, reward, done, info = env.step(bitmask)
    recent.append(move)
```

Real-Time Inference



Real-Time Inference

Endpoints!



Real-Time Inference

```
@app.cls(image=sglang_image, gpu="B200", volumes={"/root/.cache/huggingface": hf_cache})
@modal.concurrent(max_inputs=32, target_inputs=16)
class Qwen35Server:
    ckpt_path: str = modal.parameter(default="")

    @modal.enter()
    def enter(self):
        self.llm = sgl.Engine(model_path=self.ckpt_path or "Qwen/Qwen3.5-9B")

    @modal.method()
    async def chat(self, messages, available_moves):
        regex = move_regex(available_moves)
        return await self.llm.async_generate(messages, sampling_params={"regex": regex})
```

Real-Time Inference

```
Sample 99: 150.90ms, Move Closer, [1, 1, 1, 1]
```

```
-----  
Latency per move percentiles (ms):
```

```
  p50: 150.30ms
```

```
  p90: 244.43ms
```

```
  p95: 246.12ms
```

```
  p99: 296.15ms
```

```
-----
```

RL in Sandboxes

> 難度	易	.	.	.	.	.	.	.	8	難
> 難度	易	1	.	.	.	.	.	.	.	難
> 難度	易	.	.	.	.	.	.	7	.	難
> 難度	易	1	.	.	.	.	.	.	.	難
> 難度	易	.	2	.	.	.	.	.	.	難
> 難度	易	.	.	3	.	.	.	.	.	難
> 難度	易	.	.	.	4	.	.	.	.	難
> 難度	易	.	.	.	.	5	.	.	.	難
> 難度	易	.	.	.	.	.	6	.	.	難
> 難度	易	.	.	.	.	.	.	7	.	難
> 難度	易	.	.	.	.	.	.	.	8	難

RL Sandboxes

Modal Dojo!

```
from modal_dojo import OnlineRollout, Qwen3_VL_8B, Qwen3_VL_8B_Recipe, TrainConfig

recipe = Qwen3_VL_8B_Recipe(
    rollout_function=sf3_rollout,
    custom_generate_function=sf3_generate,
    custom_reward_post_process_function=sf3_reward_post_process,
    actor_num_gpus_per_node=8,
    num_rollout=100,
    rollout_batch_size=32,
)
config = TrainConfig(
    model=Qwen3_VL_8B(), dataset=OnlineRollout(n_rows=32), recipe=recipe
)
config.launch()
```

RL Sandboxes

Sandboxes!

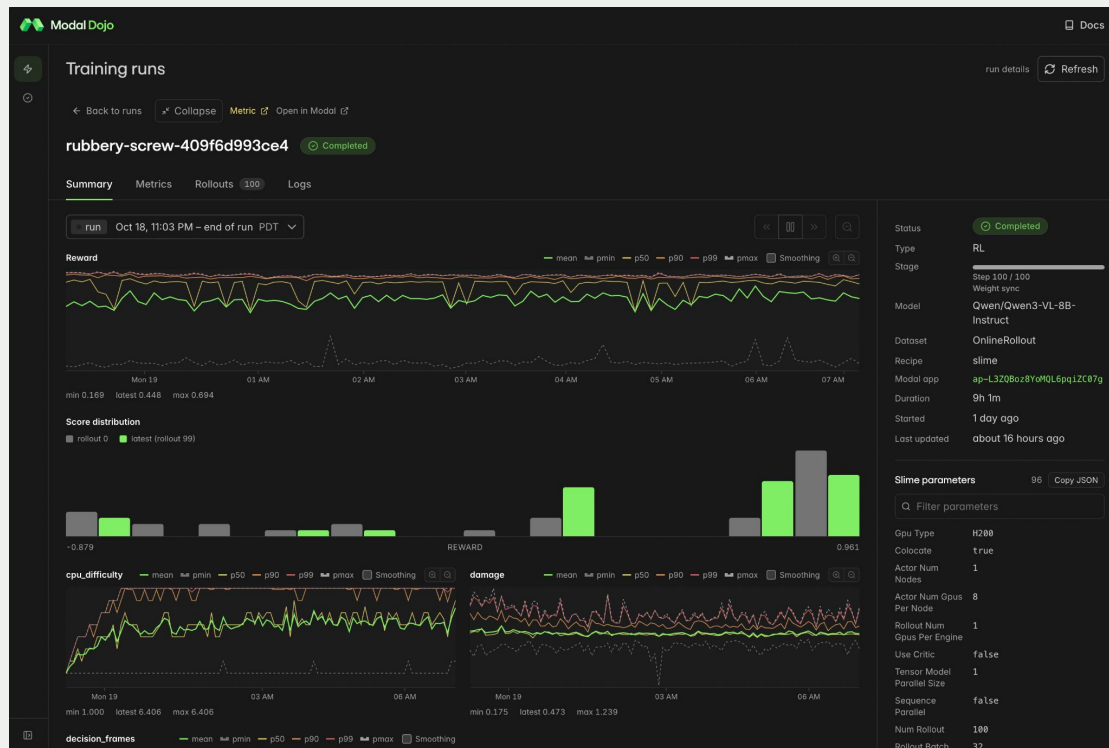
```
async def sf3_generate(args, sample):
    sandbox = await modal.Sandbox.create.aio(
        image=worker_image, runtime="gvisor", cpu=1, timeout=10 * MINUTES
    )
    process = await sandbox.exec.aio("python", "-m", "src.train.worker", config)
    await process.wait.aio()
    return await sandbox.filesystem.read_bytes.aio(RESULT_PATH)

# inside the sandbox
while not done:
    move = requests.post(f"{trainer_url}/move", json={"frame": frame}).json()
    frame, done = env.step(move)
```

RL in Sandboxes

```
def reward(move):  
    return (  
        discounted(round_won)  
        + 0.1 * discounted(damage_dealt - damage_taken)  
        - penalty * move.invalid  
    )  
  
def sample_cpu_difficulty():  
    unlocked = levels_up_to(hardest_level_with(win_rate >= 0.5) + 1)  
    weights = [win_rate(d) * (1 - win_rate(d)) for d in unlocked]  
    return random.choices(unlocked, weights)[0]
```

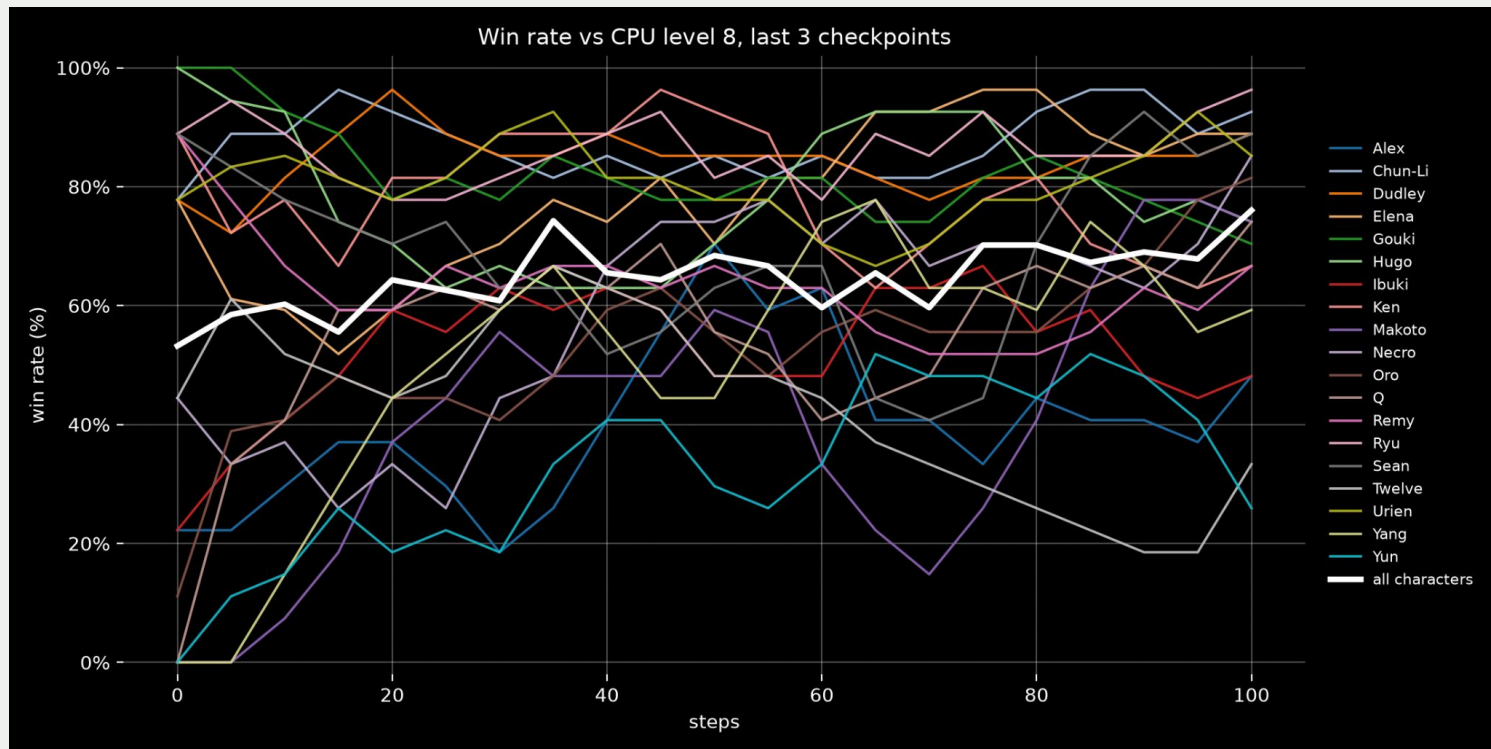
RL in Sandboxes



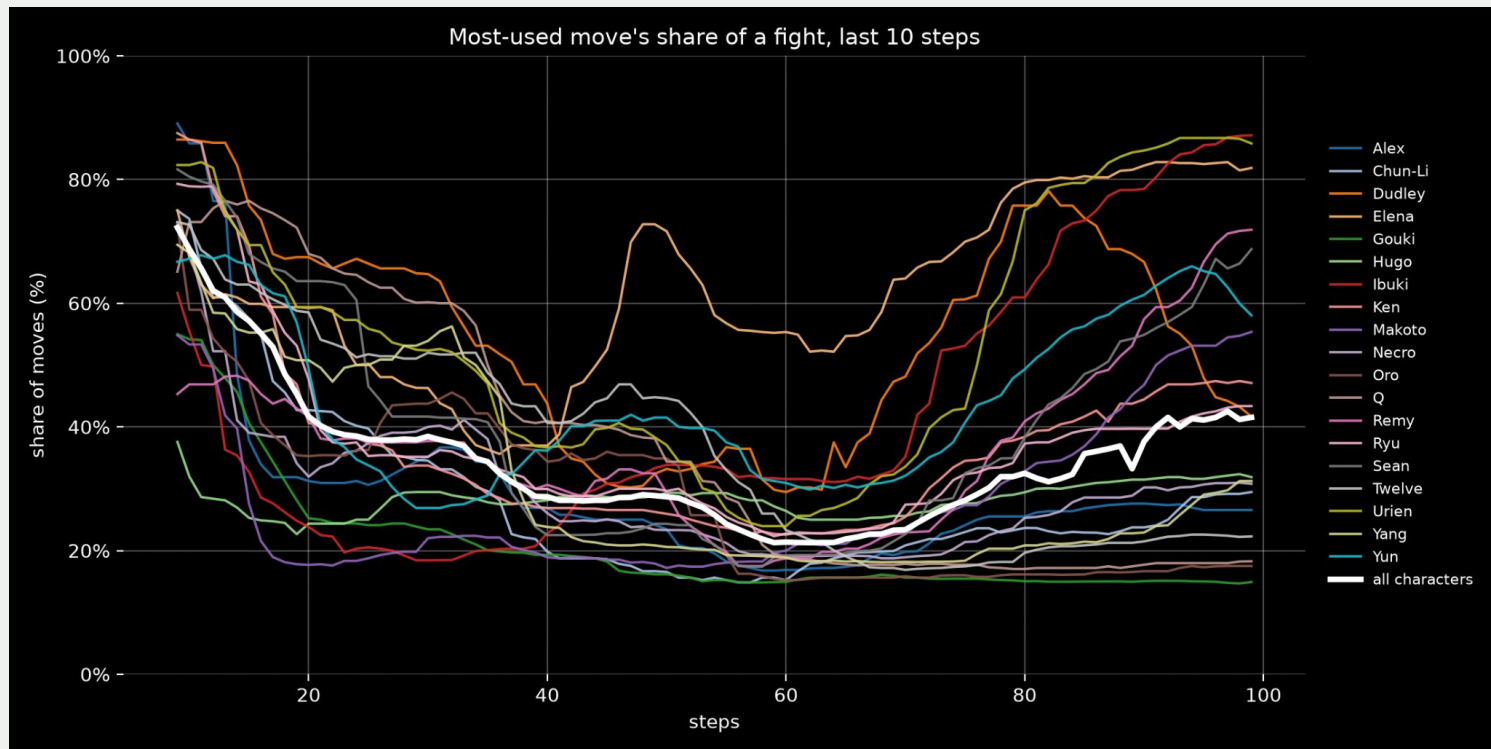
RL ♥ Sandboxes



RL ♥ Sandboxes



RL ♥ Sandboxes



Demo!



sf3.modal.dev

github.com/modal-projects/sf3