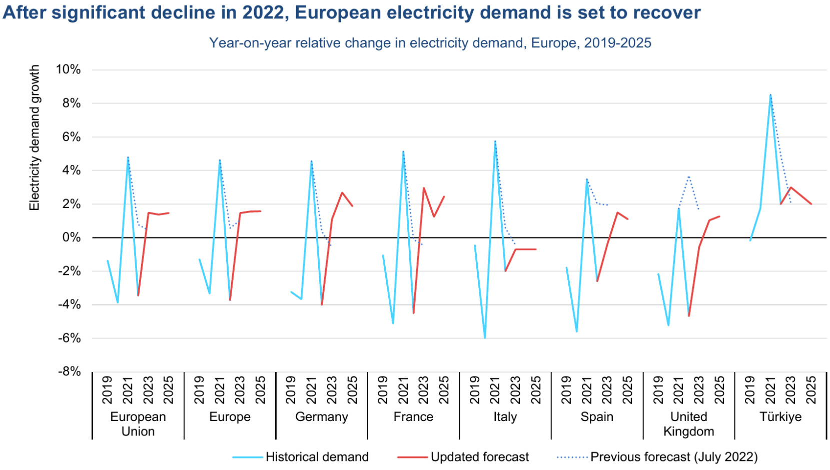
On Sound Experiment Execution with Learning Agents in CPES

ENERGY 2024, Athens, Greece

Eric MSP Veith <eric.veith@uol.de>, Stephan Balduin, Arlena Wellßow, and Torben Logemann , 2023-03-11

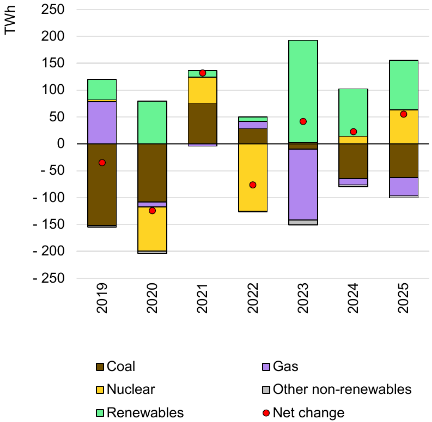
Electricity Demand Rising



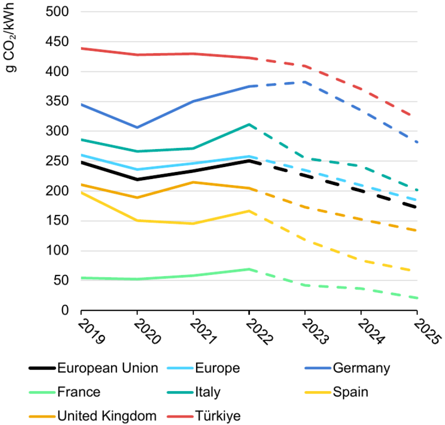
Renewables Are Replacing Fossil Fuels

Following two years of increases, CO_2 intensity starts to decline again from 2023 onward

Year-on-year change in electricity generation, European Union, 2019-2025

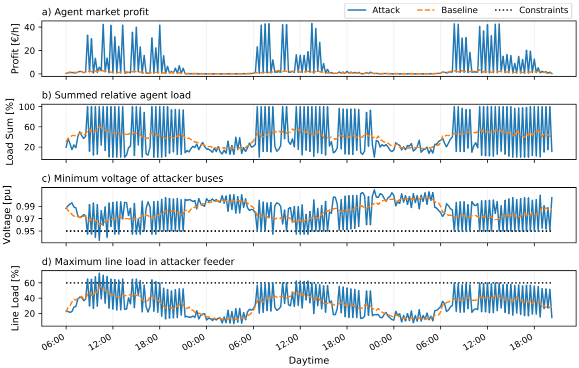


Development of average CO_2 intensity, Europe, 2019-2025



Transactive Energy Can Be Gamed

- Economic and control techniques, based on market standard values
- There is no “sound” market design yet than cannot be gamed
- Worse yet: Agents can find weaknesses & gain market dominance without system knowledge



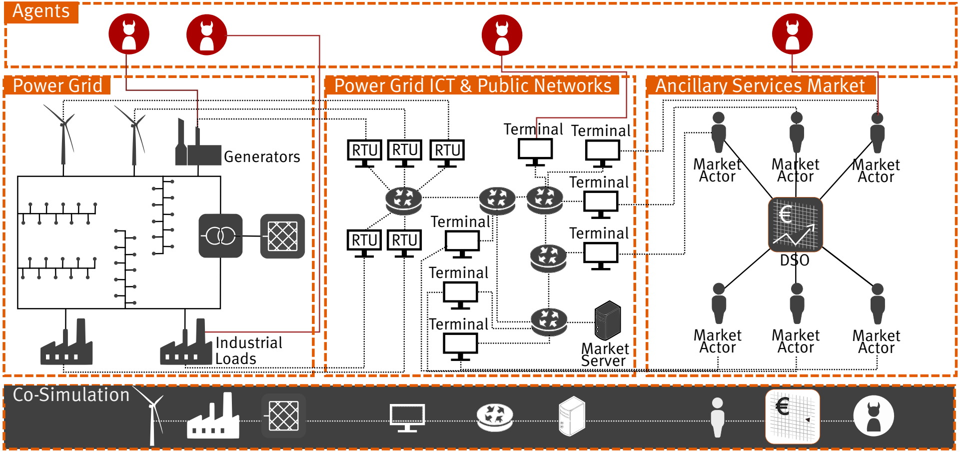
Agents learn to “game” local energy markets

Wolgast, Veith, and Nieße [3]

Attacks from Physical Nodes

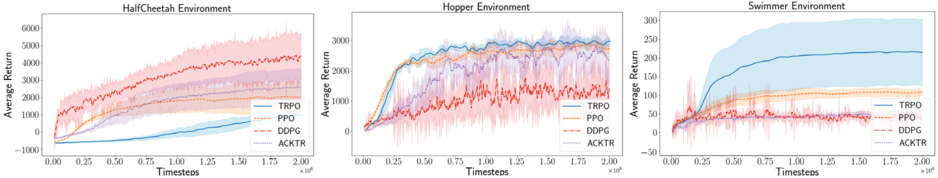
- DRL-based attacker learns to violate voltage controllers
- Leverages control schemes to act against each other

Simulations Have Become Complex



Many influencing factors in DRL alone

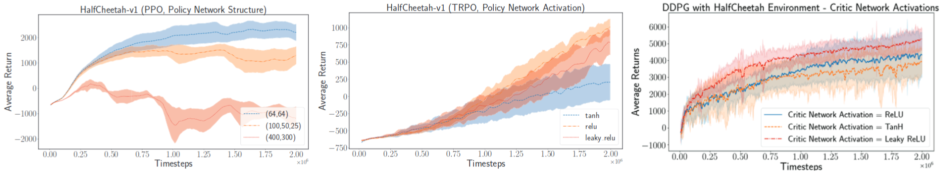
Implementation



Henderson et al. [1]

Many influencing factors in DRL alone

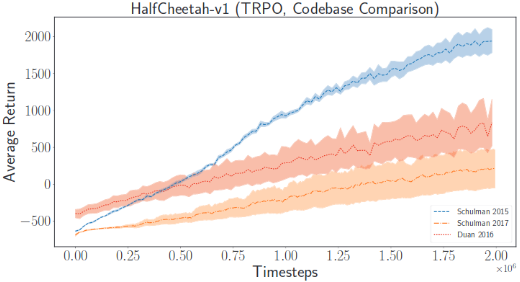
Policy Structure and Activation Functions



Henderson et al. [1]

Many influencing factors in DRL alone

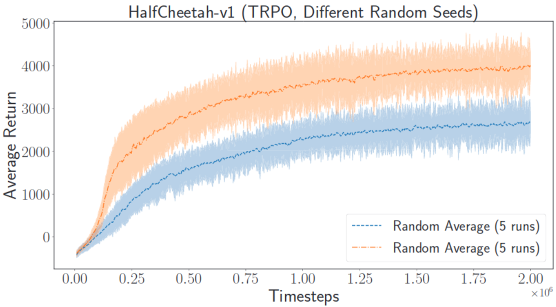
Implementations



Henderson et al. [1]

Many influencing factors in DRL alone

Random Seeds

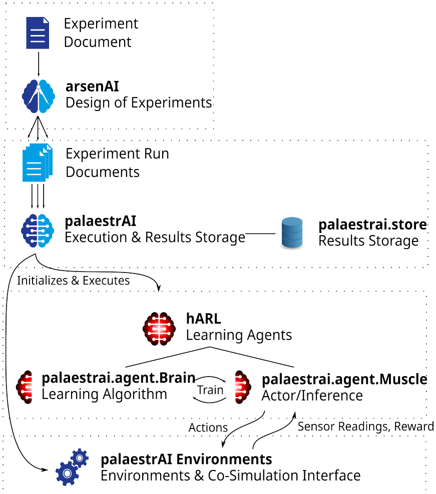


Henderson et al. [1]

Research Question and Main Contribution

How can we guarantee sound execution of experiments in complex, co-simulated Cyber-Physical Energy Systems with learning agents?

A Software Stack Approach



Structure of an Experiment File

- Global configuraton
  - Name
  - Random seed
- Definitions
  - Agents: Name, algorithm, hyperparameters, utility functions
  - Environments: software, parameters, reward functions
  - Sensor/actuator sets
  - Phase configurations: Training or test mode, number of repetitions
  - Simulation configuration: Execution strategy, termination conditions (invariants to check against)
- Phases
  - Defining possible combinations of agents with sensors/actuators in environments, with particular configurations
  - Establishes DoE sample space

Why Phases

- A phase is a concrete combination of...
  - agent(s) with their (hyber-) parameters set and sensors/actuators assigned
  - Environment(s) (with parameters)
  - Terminiation condition(s)
  - Configuration (such as training or test)
  - Simulation execution strategy (e. g., agents take turns in particular order)
- An experiment (run) can have multiple phases, serially after each other
- Agents with the same name from previous phases are reloaded

Carl von Ossietzky Universität Oldenburg

Experiments, Runs, and Phases: An Example

- Research hypothesis: Agents trained in an autocurriculum setting learn better strategies than agents trained alone
- Idea to verify:
  - Train Agent 1 (A_1) alone
  - train Agent 2 (A_2) alone
  - train variants of those agents in autocurriculum setup: A'_1 vs. A'_2)
  - Test plan: A_1 vs. A_2; A'_1 vs. A_2, A_1 vs. A'_2.
  - Assumption (invariant): $A'_1 \geq A'_1 > A_1 \geq A_2$ (Better: $\sum_i R_i(A)$)
- Many more factors influence outcome, so we need to test different variantions (DoE!), e. g.
  - Start date, end date
  - Hyperparameters
  - Software versions
  - …
- Test all factor combinations

Carl von Ossietzky Universität Oldenburg

Format Chosen

- YAML: Machine readable, but also user-friendly (e. g., anchors)
- “Cascading” phase config (add everything from previous phase, unless explicitly overwritten)

```
function Expand-Schedule(experimentrun)
    schedule = Empty-List
    for phase ∈ experimentrun.phases do
        schedule ← schedule ∪ Deep-Copy(
                          Update-Mapping(schedule, phase))
    return schedule
function Update-Mapping(src, upd)
    for key, value ∈ upd do
        if val isa Mapping then
            entry ← val_key ∨ Empty-Mapping()
            src_key ← Update-Mapping(entry, value)
        else
            src_key ← value
    return src
```

Experiment Definition File: Example I

```yaml
uid: Classic ARL
seed: 2022
version: 3.5.0
output: palaestrai-runfiles
repetitions: 1
max_runs: 300
definitions:
  environments:
    midasmv_tar_ms:
      environment:
        name: MosaikEnvironment
        uid: midas_powergrid
        params: {}
      reward:
        name: ExtendedGridHealthReward
  agents:
    gandalf_sac_single:
      name: Gandalf SAC (single-agent-training)
      brain: &sac_brain
        name: harl:SACBrain
        params:
          fc_dims: [48, 48]
          update_after: 1000
```

Carl von Ossietzky Universität Oldenburg

Experiment Definition File: Example II

```
      batch_size: 500
      update_every: 200
  muscle: &sac_muscle
    name: harl:SACMuscle
    params: {}
  objective: &defender_objective
    name: ArlDefenderObjective
    params: {}
gandalf_sac_ac:
  name: Gandalf SAC (autocurriculum-training)
  brain: *sac_brain
  muscle: *sac_muscle
  objective: *defender_objective
sauron_sac_single:
  name: Sauron SAC (single-agent-training)
  brain: *sac_brain
  muscle: *sac_muscle
  objective: &attacker_objective
    name: ArlAttackerObjective
    params: {}
sauron_sac_ac:
  name: Sauron SAC (autocurriculum-training)
  brain: *sac_brain
```

Carl von Ossietzky Universität Oldenburg

Experiment Definition File: Example III

```yaml
      muscle: *sac_muscle
      objective: *attacker_objective
  sensors:
    all_sensors:
      midas_powergrid: [s1, s2]
  actuators:
    attacker_actuators:
      midas_powergrid: [a1, a2]
    defender_actuators:
      midas_powergrid: [a3, a4]
  simulation:
    vanilla_sim:
      name: TakingTurns
      conditions:
        - name: EnvTerminates
          params: {}
  phase_config:
    train: {mode: train, worker: 1, episodes: 10}
    test: {mode: test, worker: 1, episodes: 3}
schedule:
  - Adversary Single Training:
      environments: [[midasmv_tar_ms]]
      agents: [[sauron_sac_single]]
```

Experiment Definition File: Example IV

```yaml
    simulation: [vanilla_sim]
    phase_config: [training]
    sensors: &sensors_single
      sauron_sac_single: [all_sensors]
      gandalf_ddpg_single: [all_sensors]
      gandalf_sac_single: [all_sensors]
    actuators: &actuators_single
      sauron_sac_single: [attacker_actuators]
      gandalf_sac_single: [defender_actuators]
  - Operator Single Training:
    environments: [[midasmv_tar_ms]]
    agents: [[gandalf_sac_single]]
    simulation: [vanilla_sim]
    phase_config: [training]
    sensors: *sensors_single
    actuators: *actuators_single
  - Autocurriculum Training:
    environments: [[midasmv_tar_ms]]
    agents: [[sauron_sac_ac, gandalf_sac_ac]]
    simulation: [vanilla_sim]
    phase_config: [training]
    sensors: &sensors_ac
      sauron_sac_ac: [all_sensors]
```

Carl von Ossietzky Universität Oldenburg

Experiment Definition File: Example V

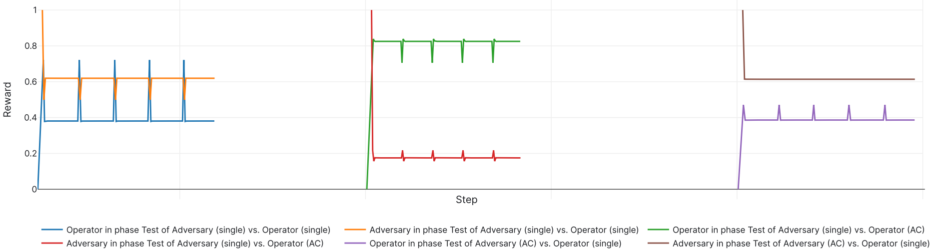
```
      gandalf_sac_ac: [all_sensors]
    actuators: &actuators_ac
      sauron_sac_ac: [attacker_actuators]
      gandalf_sac_ac: [defender_actuators]
  - Adversary (S) vs. Operator (AC) Test:
    environments: [[midasmv_tar_ms]]
    agents: [
      [sauron_sac_single, gandalf_sac_ac]]
    simulation: [vanilla_sim]
    phase_config: [test]
    sensors:
      <<: [*sensors_ac, *sensors_single]
    actuators:
      <<: [*actuators_ac, *actuators_single]
  - Adversary (AC) vs. Operator (S) Test:
    environments: [[midasmv_tar_ms]]
    agents: [
      [sauron_sac_ac, gandalf_sac_single]]
    simulation: [vanilla_sim]
    phase_config: [test]
    sensors:
      <<: [*sensors_ac, *sensors_single]
    actuators:
```

Carl von Ossietzky Universität Oldenburg

Experiment Definition File: Example VI

```yaml
      <<: [*actuators_ac, *actuators_single]
  - Adversary (AC) vs. Operator (AC) Test:
      environments: [[midasmv_tar_ms]]
      agents: [[sauron_sac_ac, gandalf_sac_ac]]
      simulation: [vanilla_sim]
      phase_config: [test]
      sensors: *sensors_ac
      actuators: *actuators_ac
```

Result of the Investigation



Conclusion

- Need for reproducible, sound execution of experiments with
  - Learning agents (DRL)
  - Complex CPES Models
  - Co-simulation
- Approach: Software stack (palaestrAI) and designated DoE tool suite (arsenAI)
- Format to define Experiments: Generate experiment runs
- Includes agents, sensor/actuator sets, environments, termination conditions, invariants, etc.
- Blue prints for actual runs
- Future work
  - Install required software automatically from defintion
  - Discover dependencies as much as possible

Bibliography I

[1] Peter Henderson et al. “Deep reinforcement learning that matters”. In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI’18/IAAI’18/EAAI’18. New Orleans, Louisiana, USA: AAAI Press, Feb. 2, 2018, pp. 3207–3214. isbn: 978-1-57735-800-8. (Visited on 02/26/2024).

[2] Eric Veith, Arlena Wellßow, and Mathias Uslar. “Learning new attack vectors from misuse cases with deep reinforcement learning”. In: *Frontiers in Energy Research* (2023).

[3] Thomas Wolgast, Eric MSP Veith, and Astrid Nieße. “Towards Reinforcement Learning for Vulnerability Analysis in Power-Economic Systems”. In: *DACH+ Energy Informatics 2021: The 10th DACH+ Conference on Energy Informatics*. Freiburg, Germany, Sept. 2021, pp. 1–20.