Machine learning based design of a supercritical CO_2 concentrating solar power plant

Tahar Nabil, Yann Le Moullec, Adrien Le Coz

EDF R&D China Center

3rd European Conference on Supercritical CO2

September 19, 2019

Introduction

Motivation (1/2)

A case study: Shouhang 10 MW SCO2-CSP demonstrator

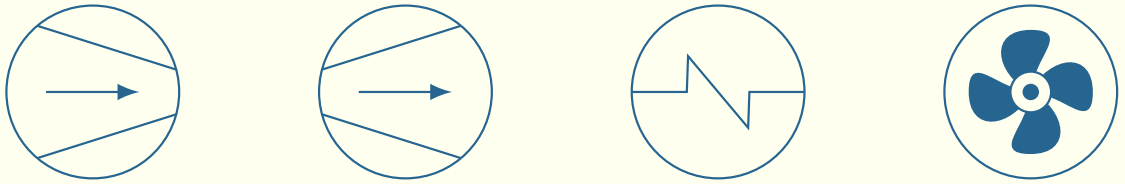
- from steam Rankine to supercritical CO_2 (SCO2) Brayton
- **Open question**: can the SCO2 cycle improve the performances of the existing steam Rankine cycle?

- SCO2 is thought to bring cost reduction, increased efficiency as well as environmental impact reduction

Conceptual design of a SCO2 power cycle

How to design an optimal process?

- Main unit operations in a power cycle:

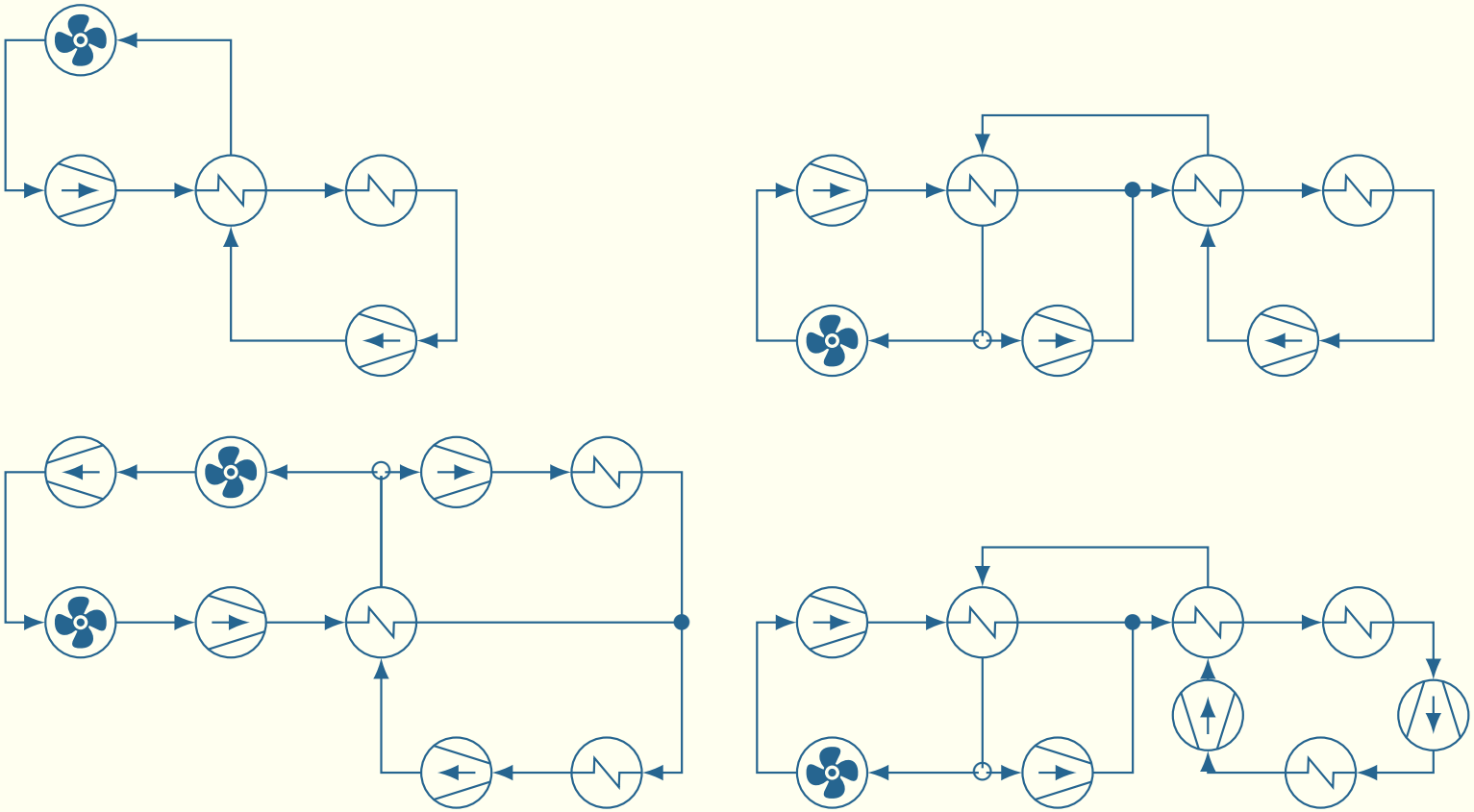


compressor *turbine* *recuperator* *cooler*

- Purpose: find the optimal layout of the power cycle
- Optimality: in terms of efficiency and/or cost of the cycle
- What are the units that should be included in the cycle?
- How to connect them in order to form an optimal process flowsheet?

Motivation (2/2)

Conceptual design of a SCO2 power cycle



Some power cycle flowsheets. How to find the best layout?

Objectives

Methodology

Derive a new method for designing SCO_2 power cycles, such that

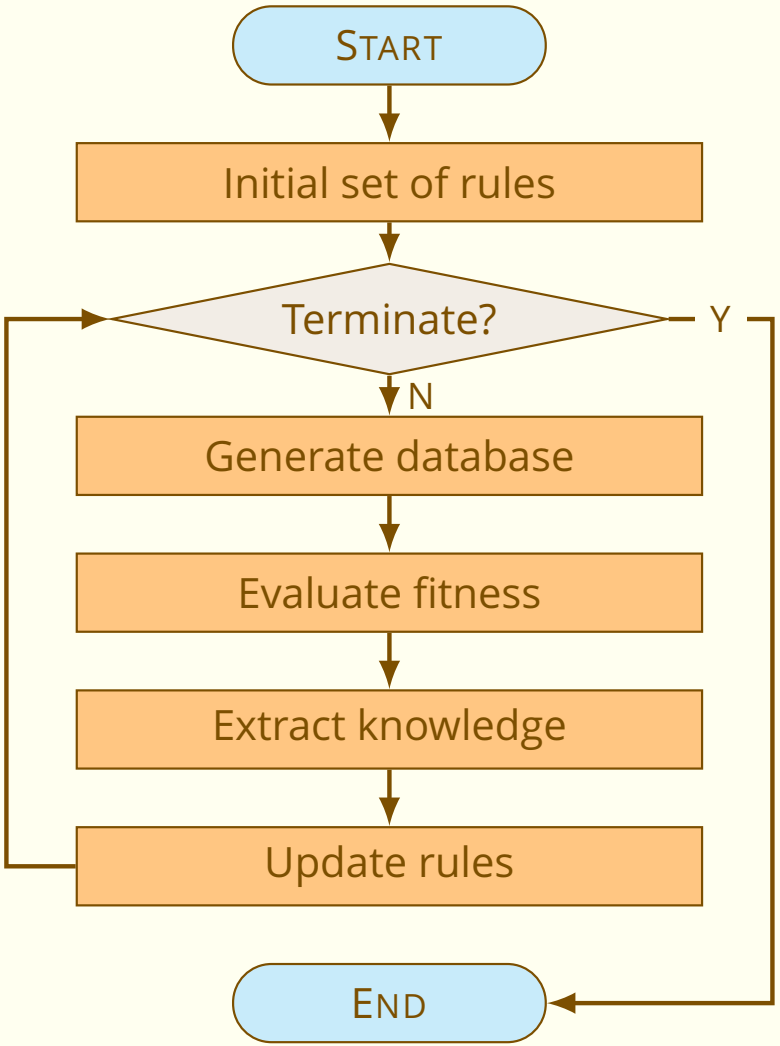
1. no expert knowledge is required
2. it is fully data-driven (statistical)
3. it is a *generative* model

Research questions

1. can a data-driven approach generate a large number of "good", distinct, power cycle designs?
2. can a data-driven approach challenge an expert design without using any domain knowledge?

A new approach: procedural generation (Shaker et al., 2016)

Our method is based on data analysis and freely explores the search space without being constrained by expert knowledge or superstructure definition



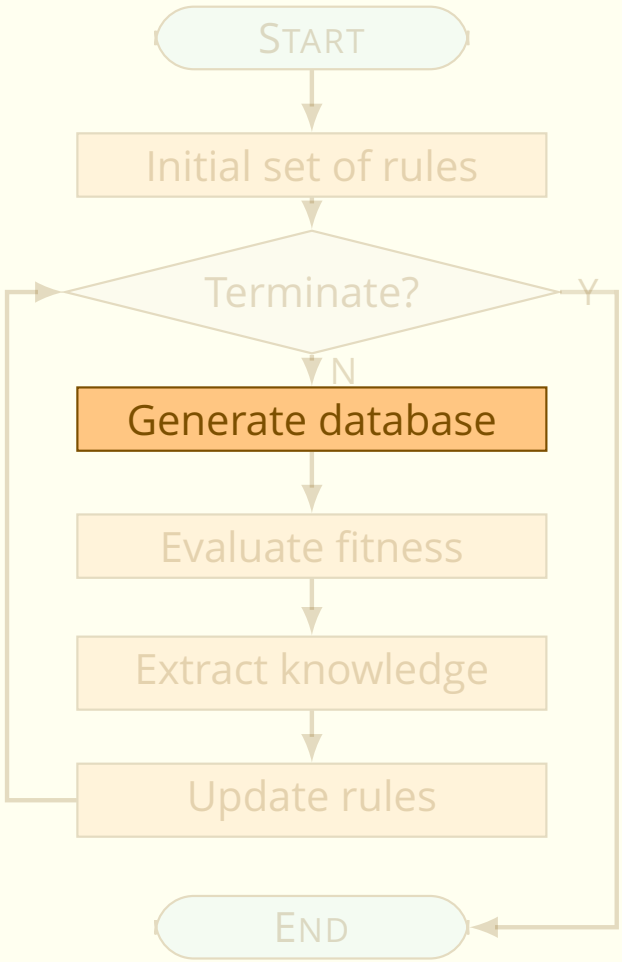
- A power cycle is a labelled, directed, cyclic graph:

  data ↔ graphs

- Fitness function: thermal efficiency and/or economic cost
- Evaluation: compute the efficiency of *any* graph?
- Learning: generate new graphs ("rules"?) and learn features from a set of graphs?

Task I: producing new power cycle layouts

A statistical generative model

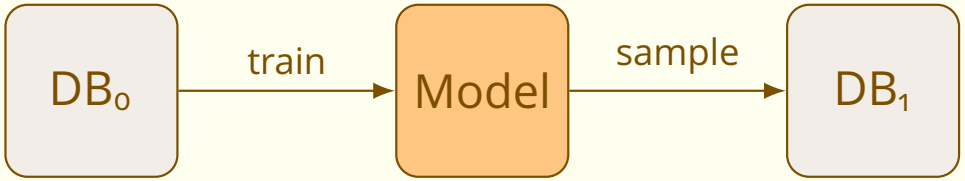


Problem formulation

1. Given an initial database DB_0, train a statistical model that learns the underlying distribution
2. Sample new elements DB_1 from the distribution estimated by the model

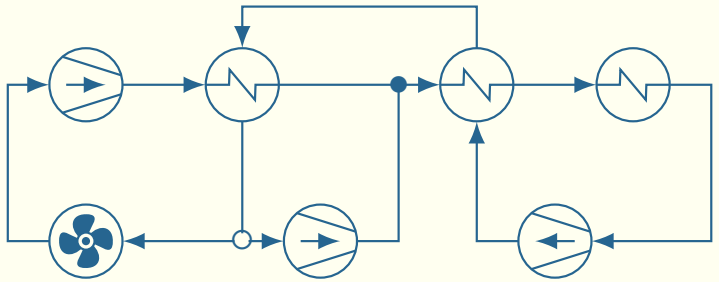
$\Rightarrow$ $DB_1 \neq DB_0$, but the data generating process is close

Here, starting from an initial set of power cycles, we want thus to generate new cycles, distinct yet structurally similar from the original set.



Preprocessing: Canonical word representation

Power cycle $\rightarrow$ unique word written in a finite alphabet `{T, C, H, A, a, b, c, d, 1, 2, 3, 4, 5, -1, -2, -3, -4, -5, >, !}`

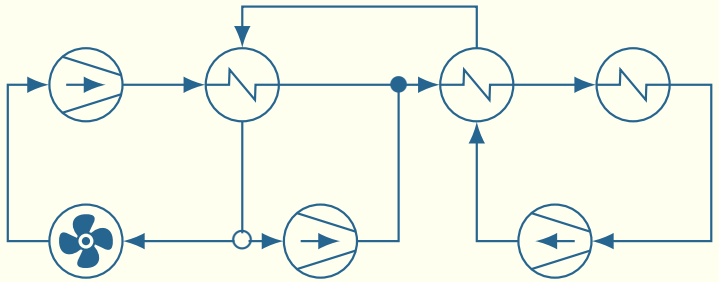


$\longrightarrow$ `'>Tab-1C1ACb1aH!'`

data $\leftrightarrow$ graphs $\leftrightarrow$ words

Preprocessing: Canonical word representation

Power cycle $\rightarrow$ unique word written in a finite alphabet `{T, C, H, A, a, b, c, d, 1, 2, 3, 4, 5, -1, -2, -3, -4, -5, >, !}`



$\longrightarrow$ `'>Tab-1C1ACb1aH!'`

data $\leftrightarrow$ graphs $\leftrightarrow$ words

Model selection

- Task: given a set DB_0 of words (power cycles), learn to generate new words without spelling mistake (new valid power cycle layouts)
- Deep learning model: the Recurrent Neural Network, suited to handle sequential data (Olivecrona et al., 2017; Blaschke et al., 2018)
- We choose a LSTM model, belonging to the class of RNNs, for long-term dependencies (Hochreiter and Schmidhuber, 1997)

RNNs/LSTMs in a nutshell

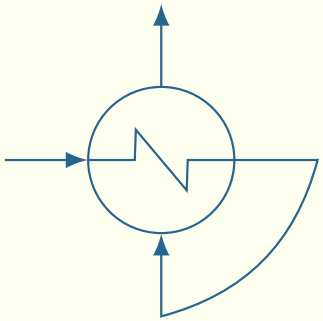
- training mode: for a sequence of letters, learn the probability distribution of the next letter
- generation mode: start from $>$, use previous distribution to sample new letters iteratively until a word is finished (`!` is drawn)

RNNs/LSTMs in a nutshell

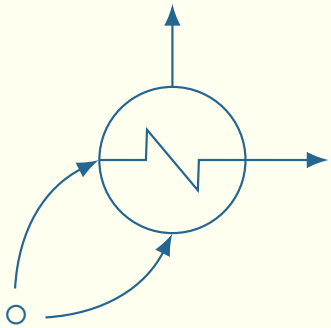
- training mode: for a sequence of letters, learn the probability distribution of the next letter
- generation mode: start from $>$, use previous distribution to sample new letters iteratively until a word is finished (`!` is drawn)
- How to guarantee that the generated sequence of letters is indeed a power cycle? $\rightarrow$ hand-designed "spelling rules"

RNNs/LSTMs in a nutshell

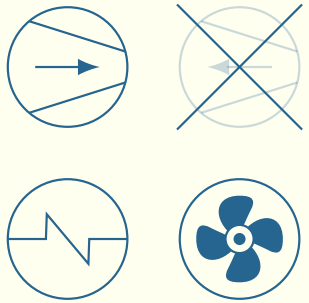
- training mode: for a sequence of letters, learn the probability distribution of the next letter
- generation mode: start from $>$, use previous distribution to sample new letters iteratively until a word is finished (`!` is drawn)
- How to guarantee that the generated sequence of letters is indeed a power cycle? $\rightarrow$ hand-designed "spelling rules"



'..aa..'



'..-1a..-1a...'



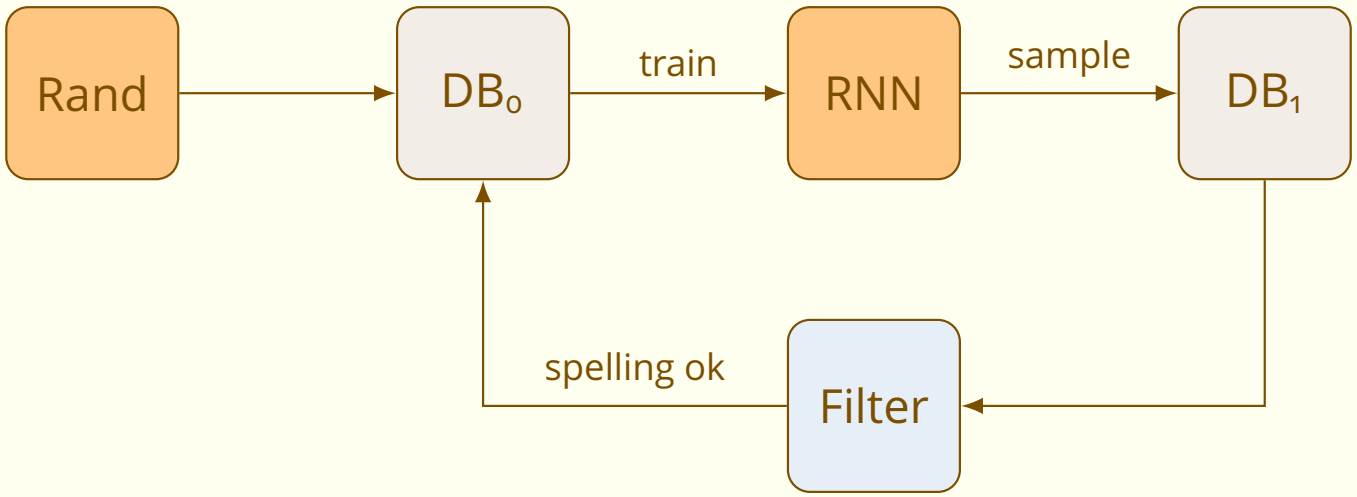
no *T* in *'...'*

Observe: the spelling rules are never specified to the model! RNN learns by mimicking the database only.

Lack of data

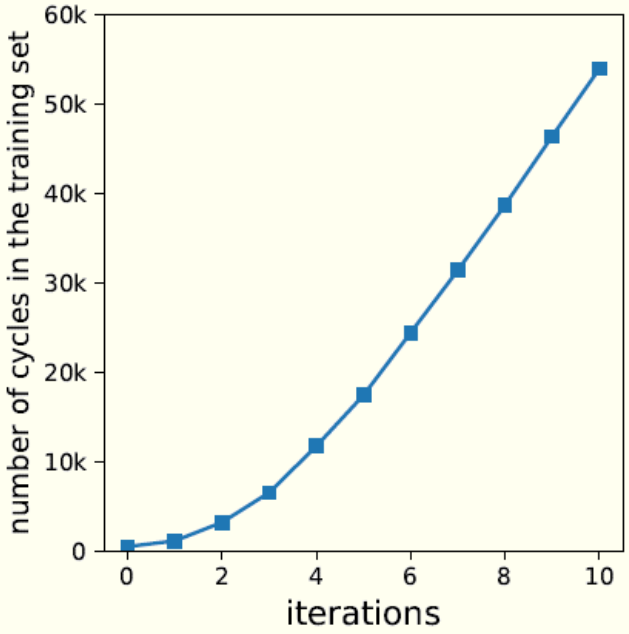
Problem: how to create the initial database DB_0, with sufficiently many instances?

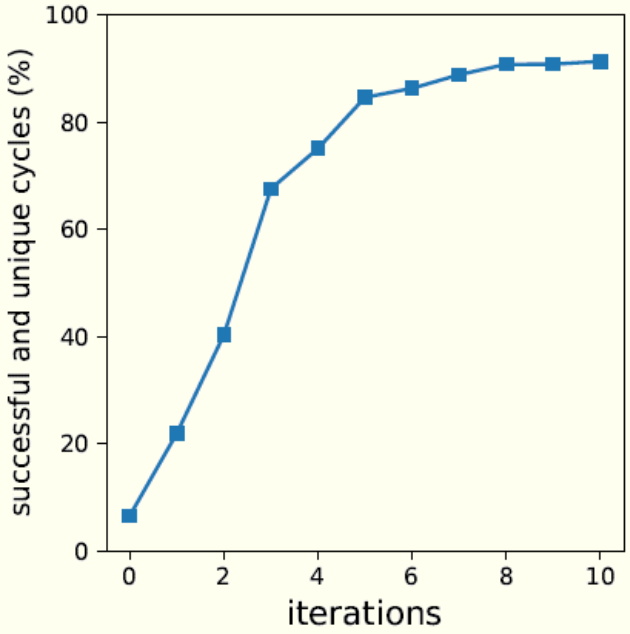
1. Start with a few instances created by a *random* generator $\rightarrow$ dataset DB_0
2. Train a RNN model based on this initial set
3. Generate new cycles with the RNN $\rightarrow$ dataset DB_1
4. Filter the valid cycles, with no spelling mistakes, and append them to the initial set
5. Iterate from Step 2, until the RNN shows sufficiently good performances



Results

450 initial random words - 2 LSTM layers (32 units in each) + 1 dense output layer (softmax activation) -
cycles with at most 1 heat exchanger and 1 pair splitter/mixer

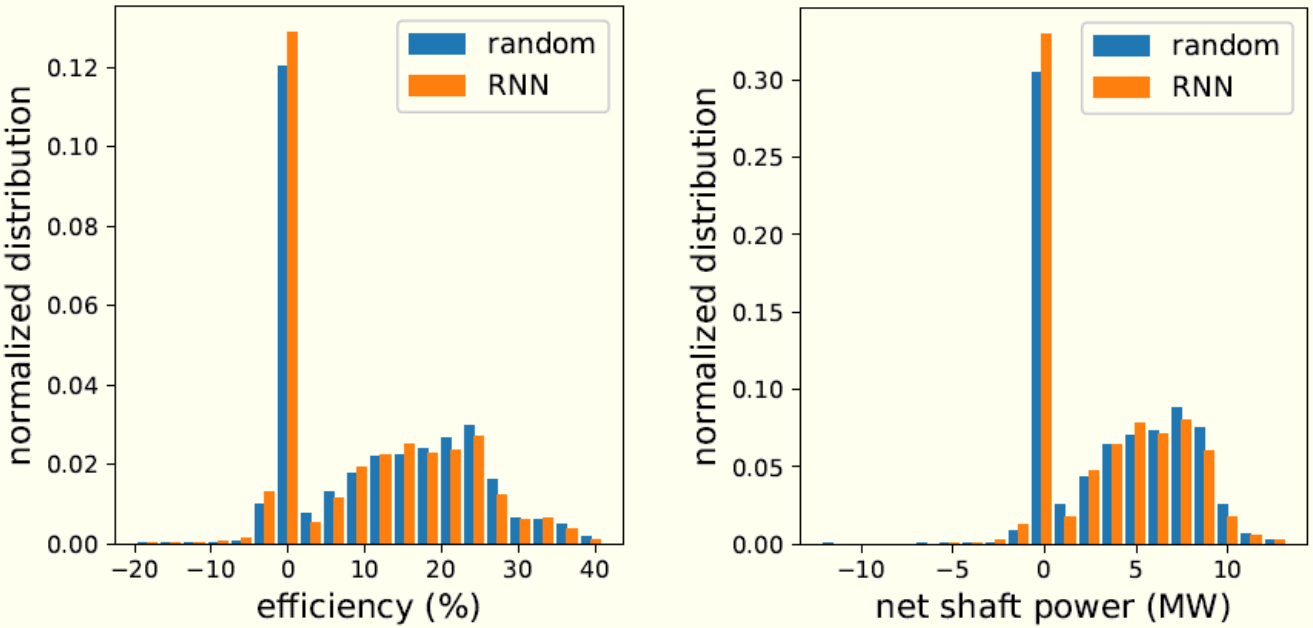




- After 10 iterations, DB_1 has 53,959 distinct unique power cycles
- Final model: 93.1% valid words (actual power cycles) - of which 87% are unique - of which 86% are new, not in DB_0

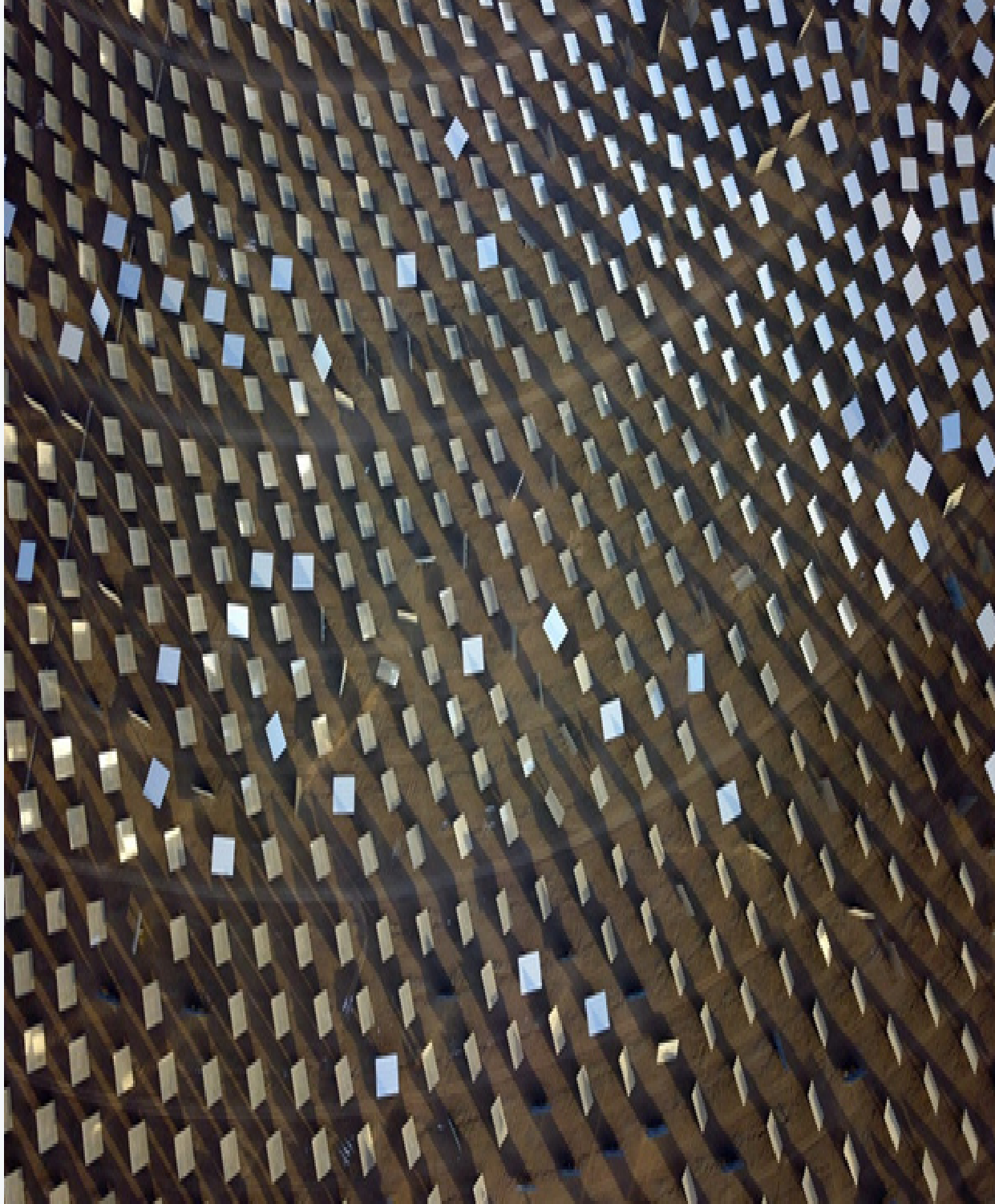
Results

450 initial random words - 2 LSTM layers (32 units in each) + 1 dense output layer (softmax activation) - cycles with at most 1 heat exchanger and 1 pair splitter/mixer

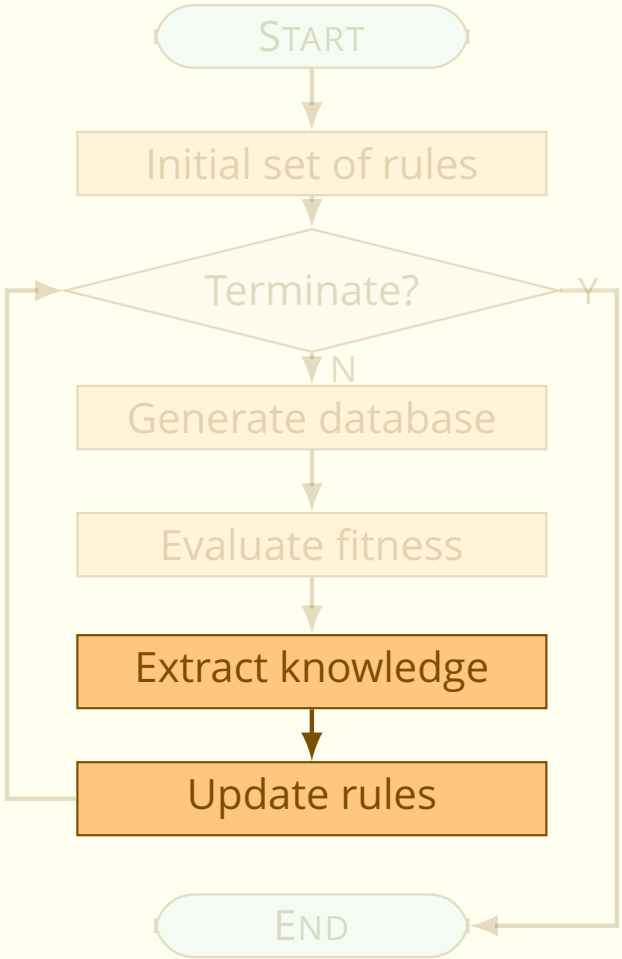


- Estimation of the optimal design parameters for $\sim$ 10k cycles (fitness: efficiency η (%) $\times$ shaft power $\mathcal{P}$ (MW))
- The distributions of η and $\mathcal{P}$ is similar in DB_0 (random generator) and DB_1 (RNN)

Task II: generating *good* power cycles

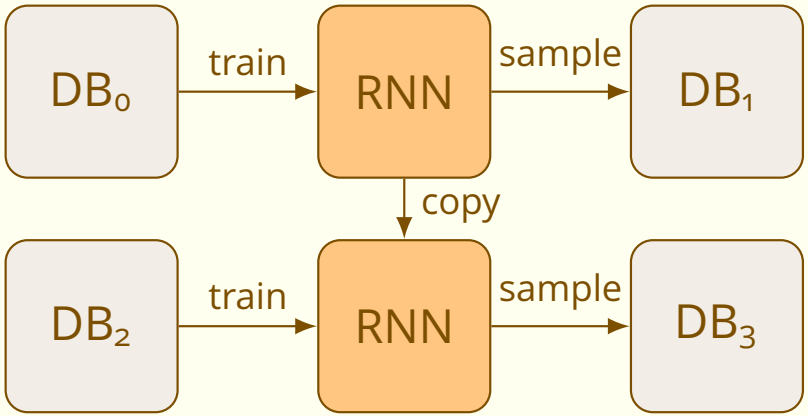


Aim: create new power cycles all s.a. fitness $>$ threshold, with fitness $= \eta \times \mathcal{P}$



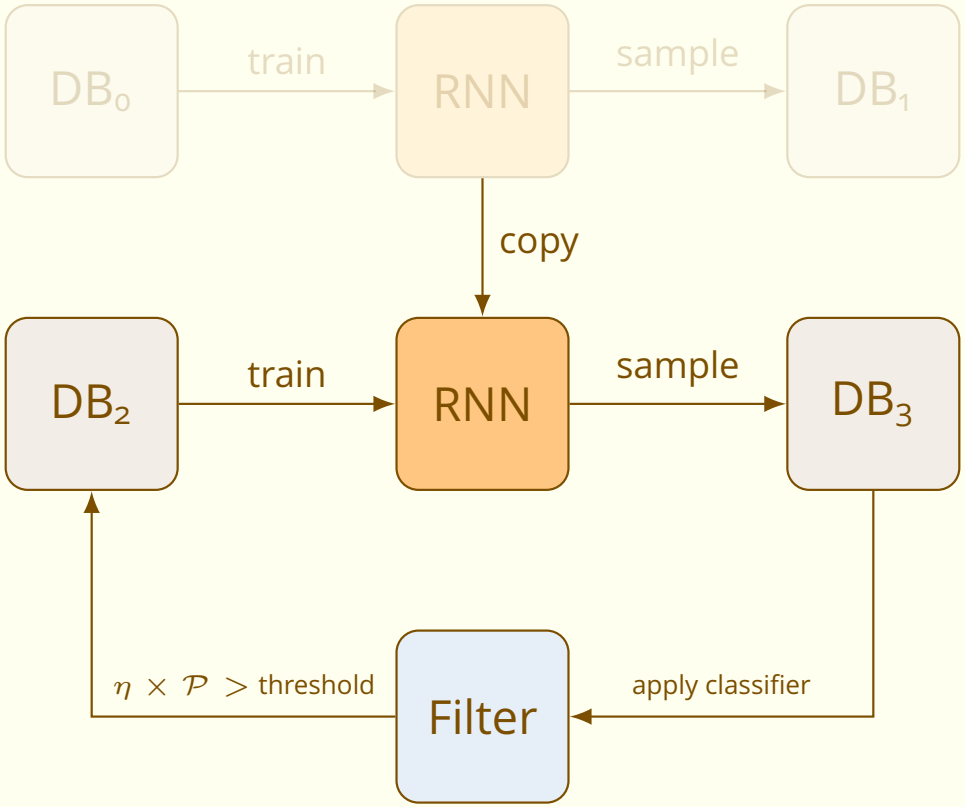
Methodology: finetuning

1. Start with a RNN that has learned the power cycle language (previous section)
2. Form a database DB_2 of cycles all s.t. $\eta \times \mathcal{P} >$ theshold
3. Re-train the model from Step 1 on DB_2



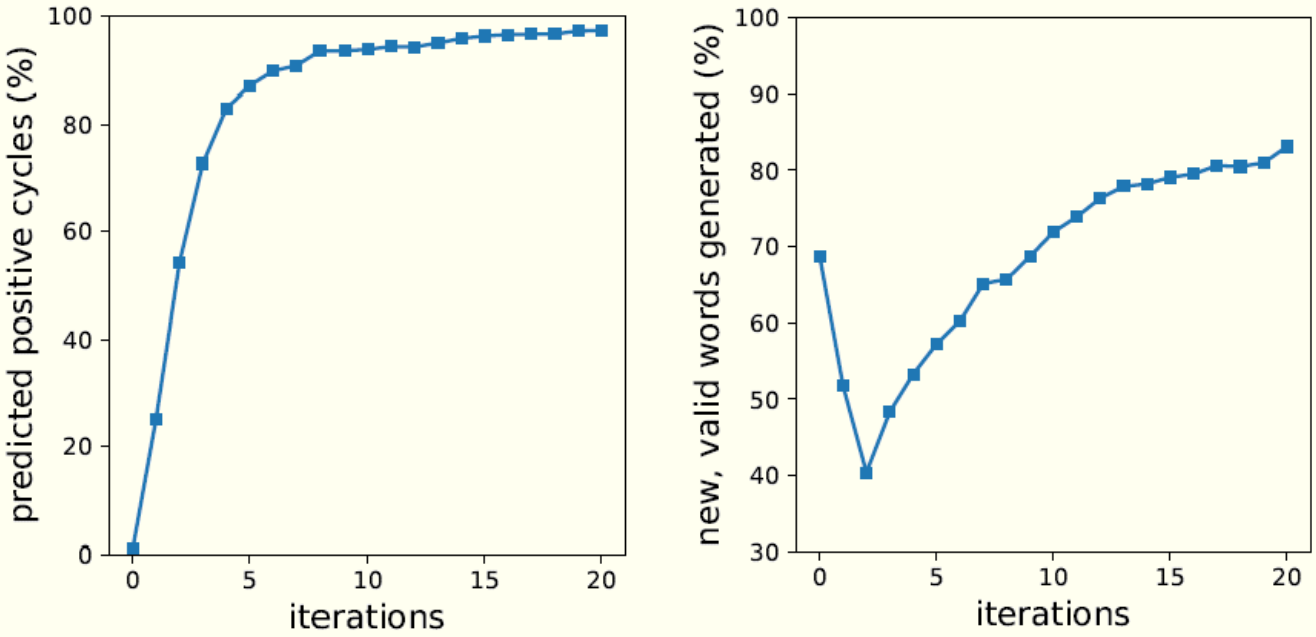
Iterative procedure

- Adopt a procedure similar to lack of data during language learning
- Use the classifier to predict $\eta \times \mathcal{P} >$ threshold and filter



Results

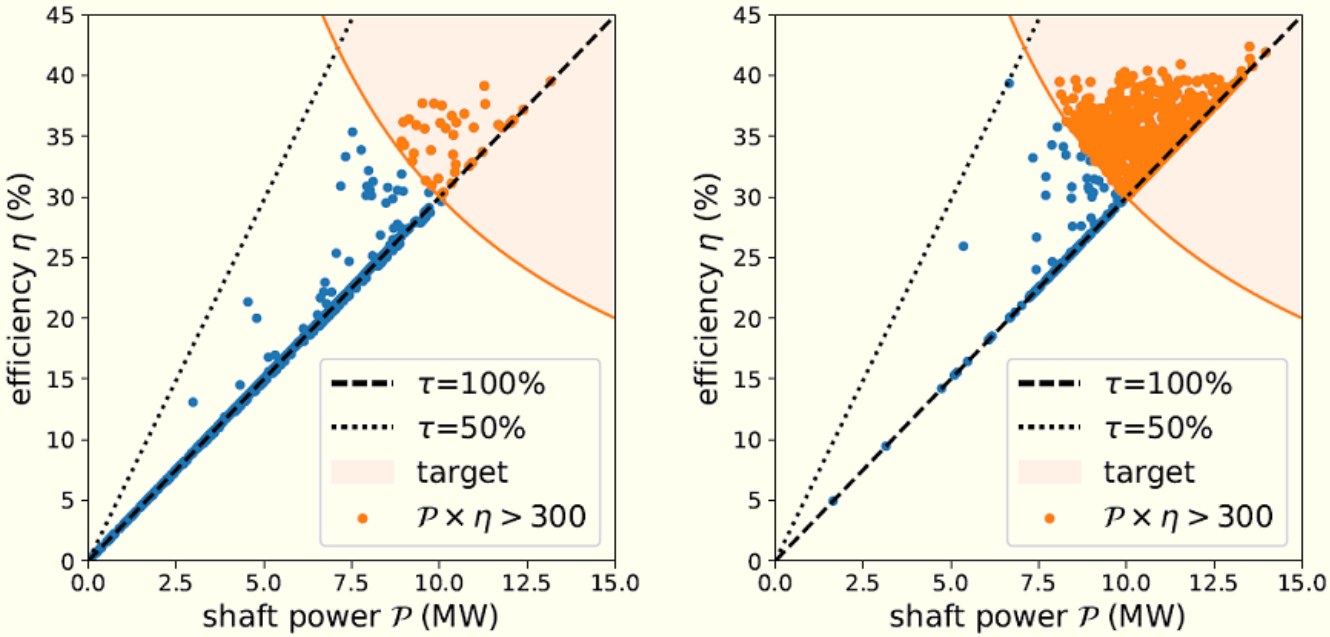
Criteria: $\eta \times \mathcal{P} \geq 300$ *(cycles better than a cycle with 30% efficiency, 10 MW shaft power)*



- After a few iterations, the finetuned RNN model generates mostly unique, valid cycles such that $\eta \times \mathcal{P} \geq 300$ (according to the classifier)
- In total, 307,165 distinct cycles are predicted to have $\eta \times \mathcal{P} \geq 300$

Results

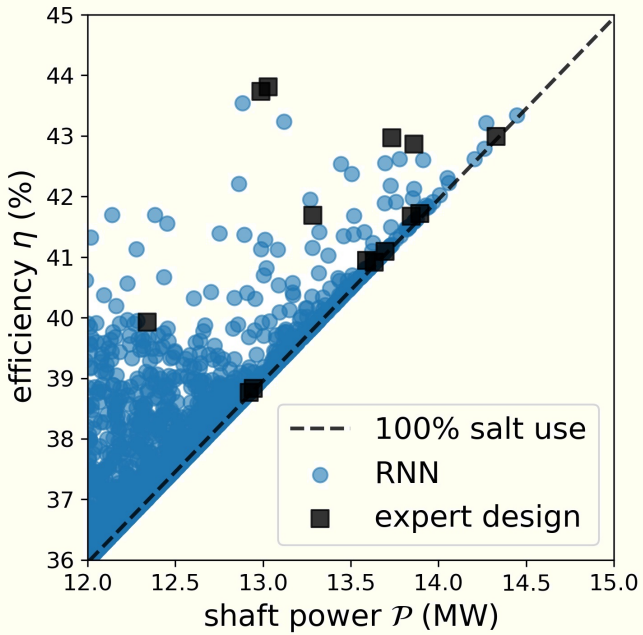
Criteria: $\eta \times \mathcal{P} \geq 300$ *(cycles better than a cycle with 30% efficiency, 10 MW shaft power)*



- Performing the actual optimization of $\eta \times \mathcal{P}$ for 17,212 cycles in the final database: 86% of them do have $\eta \times \mathcal{P} > 300$, against $< 4\%$ with no finetuning

Results

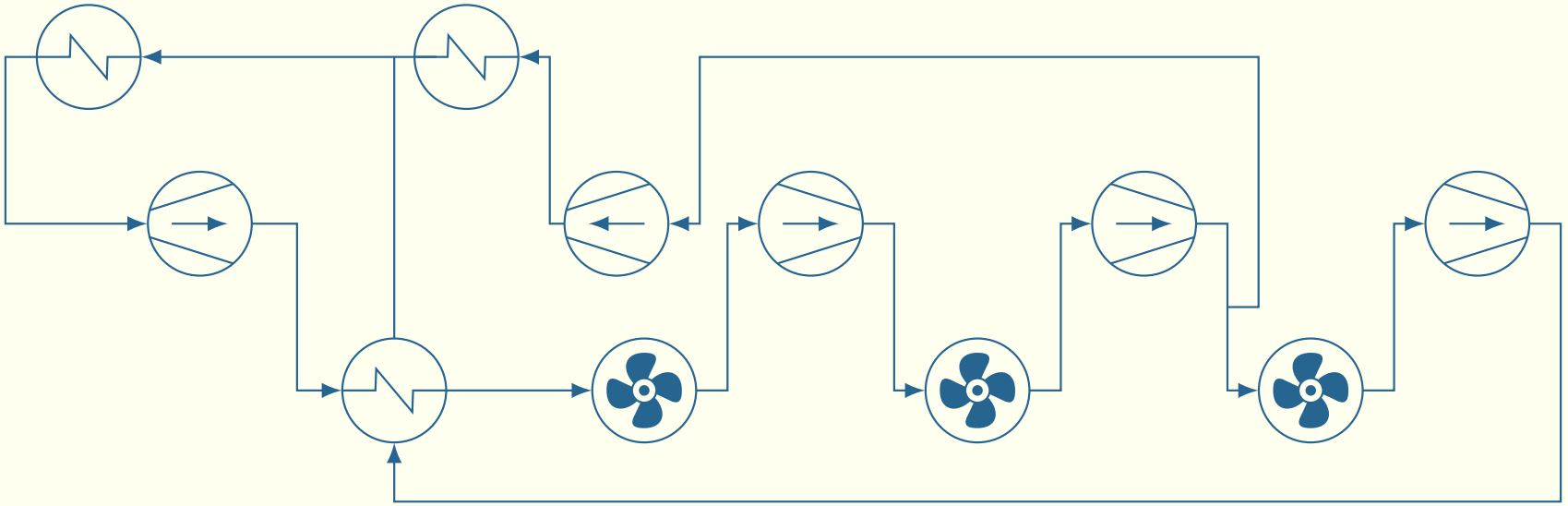
Criteria: $\eta \times \mathcal{P} \geq 300$ *(cycles better than a cycle with 30% efficiency, 10 MW shaft power)*



- Comparison to 16 expert design (black squares): RNN produces many cycles with performances equivalent or better than state-of-the-art knowledge

Results

Criteria: $\eta \times \mathcal{P} \geq 300$ *(cycles better than a cycle with 30% efficiency, 10 MW shaft power)*

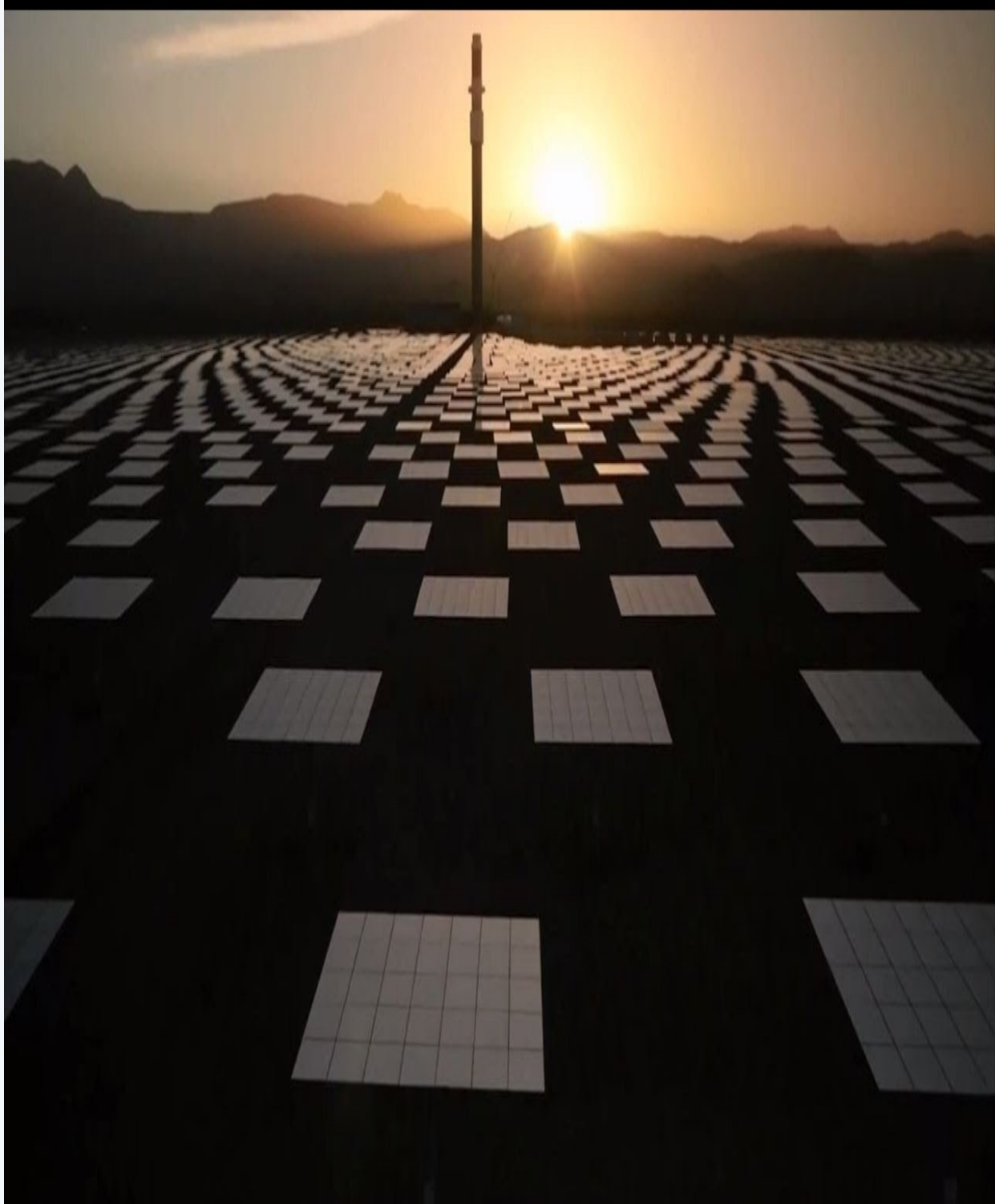


`TaACAC-1CH1ACa1H`

Layout of the best cycle found by RNN.

- The best layout generated by RNN has $\eta = 43.3\%$, $\mathcal{P} = 14.4$ MW
- It is a combination of partial cooling, intercooling and preheating

Conclusion



Conclusion

Main achievements: revisited power cycle design

- RNN imitates an initial database by producing new power cycles with similar structure and thermal properties
- RNN improves DB_0 by specializing into high η and high $\mathcal{P}$ cycles
- $DB_0 \leftrightarrow$ *a priori* knowledge for RNN - here: purely random DB_0, hence no expert knowledge
- RNN still retrieves some known patterns: partial cooling, intercooling, preheating
- Final result: a large pool of power cycles all with very good thermal properties, challenging the expert design
- Main limitation: computation time in the fitness evaluation step

Future work

- Analyze pool of best power cycles
- Improve the quality of the fitness evaluation step
- Techno-economic design of SCO_2 cycles

Thank you for your attention

More questions? tahar-t.nabil@edf.fr

References

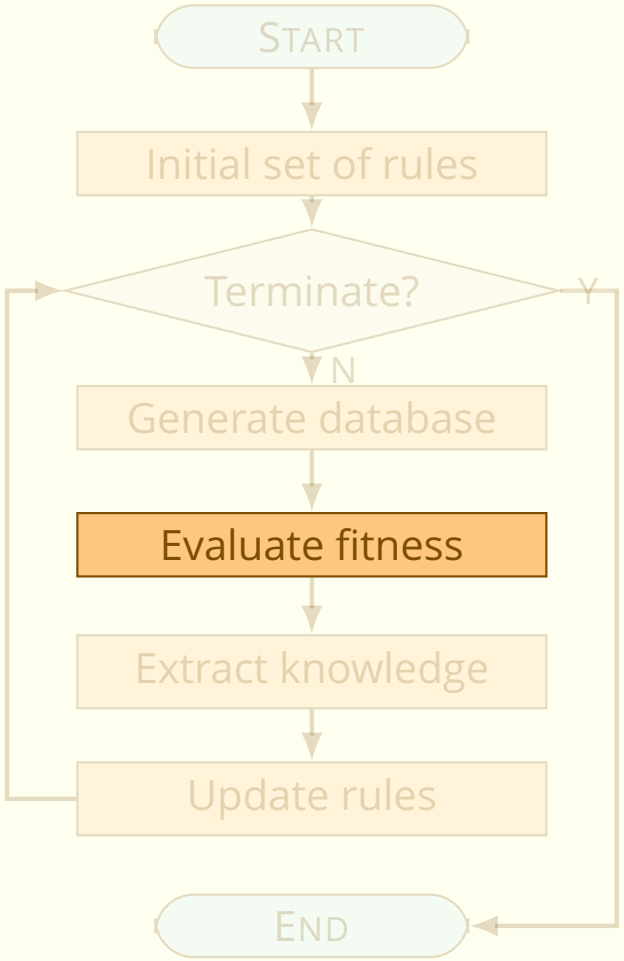
Blaschke, T., Olivecrona, M., Engkvist, O., Bajorath, J., and Chen, H. (2018). Application of generative autoencoder in de novo molecular design. *Molecular informatics*, 37(1-2):1700123.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Olivecrona, M., Blaschke, T., Engkvist, O., and Chen, H. (2017). Molecular de-novo design through deep reinforcement learning. *Journal of cheminformatics*, 9(1):48.

Shaker, N., Togelius, J., and Nelson, M. J. (2016). *Procedural content generation in games*. Springer.

Fitness = thermal efficiency



Necessary tools

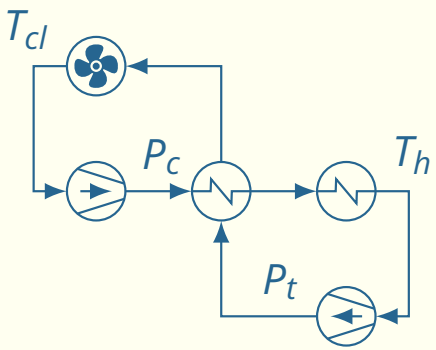
- Static simulation: given any power cycle layout and a set of design constraints, compute the net cycle efficiency
- Optimization: given any power cycle layout and a minimal set of design constraints, find the optimal (e.g. highest efficiency) design parameters
- Machine Learning (classification): given any power cycle layout, predict whether the efficiency will be greater than a certain threshold

In the sequel, we assume that those tools all are available.

Observe: any data-intensive method will consider a lot of data, for instance $10^4 - 10^5$. However, the costly optimization problem may be solved in around 1 minute (Python), i.e. in total $\sim 7 - 69$ days! The classifer is thus helpful in that it provides an immediate first estimate of the fitness function.

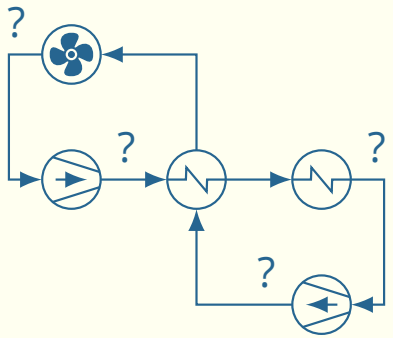
Prerequisite

Static simulation



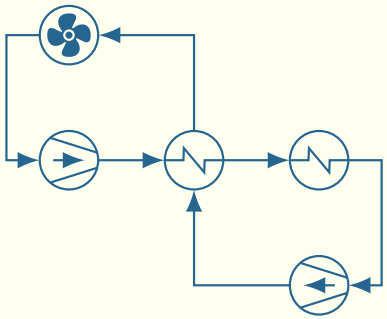
$$\begin{pmatrix} T_{cl} \\ P_c \\ T_h \\ P_t \end{pmatrix} \mapsto \eta$$

Optimization



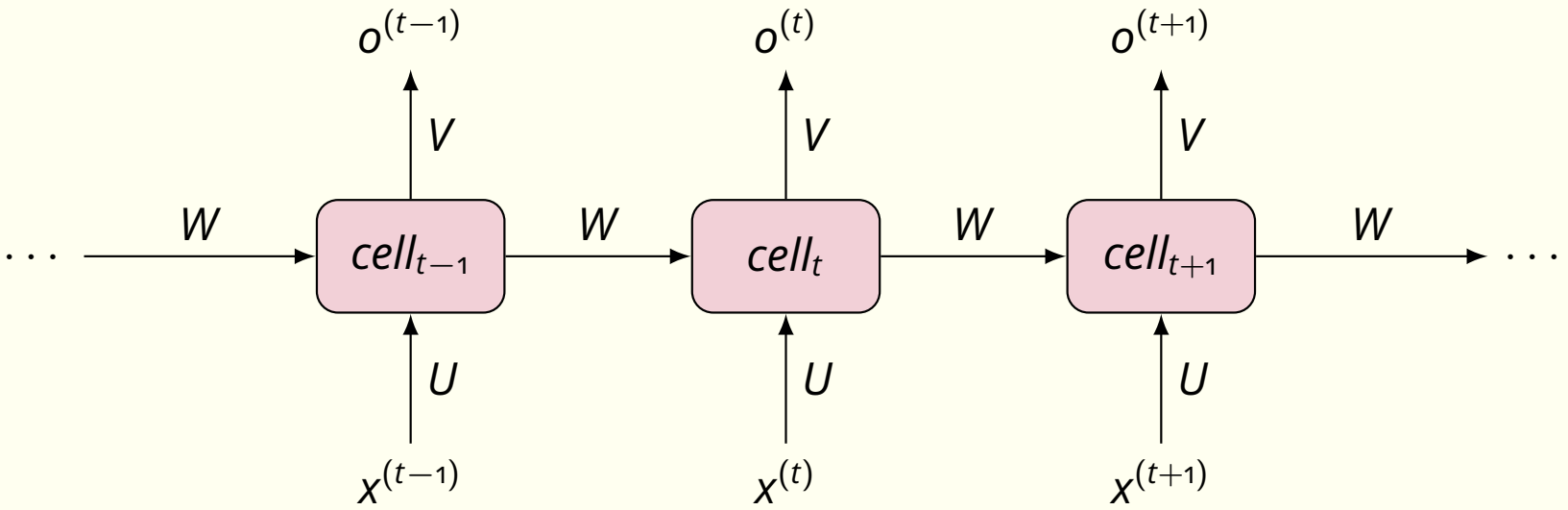
$$TaACaH \mapsto \begin{pmatrix} T_{cl}^{\star} \\ P_c^{\star} \\ T_h^{\star} \\ P_t^{\star} \end{pmatrix} = \arg\max \eta$$

Machine learning (classification):



$$TaACaH \mapsto \delta_{\eta^{\star} > 0.3}$$

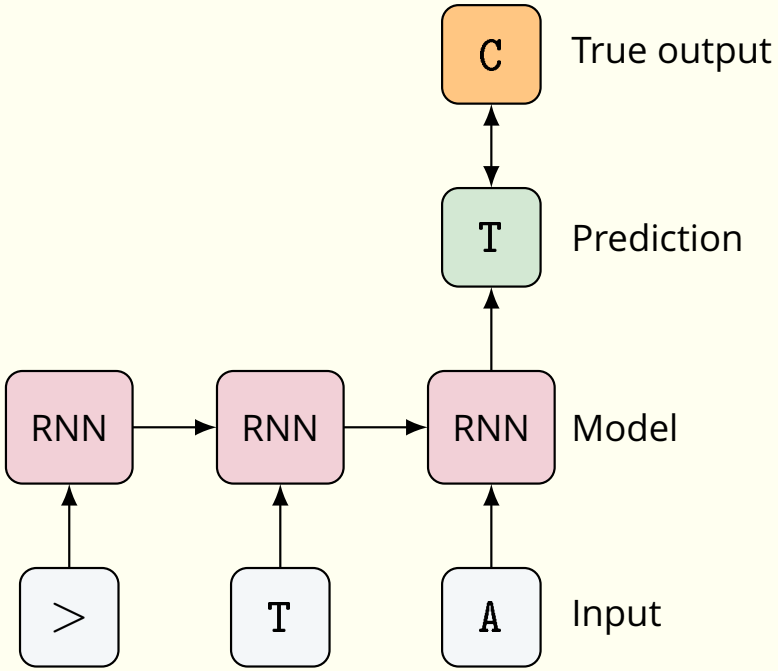
Recurrent neural networks (RNNs)



Back to slides.

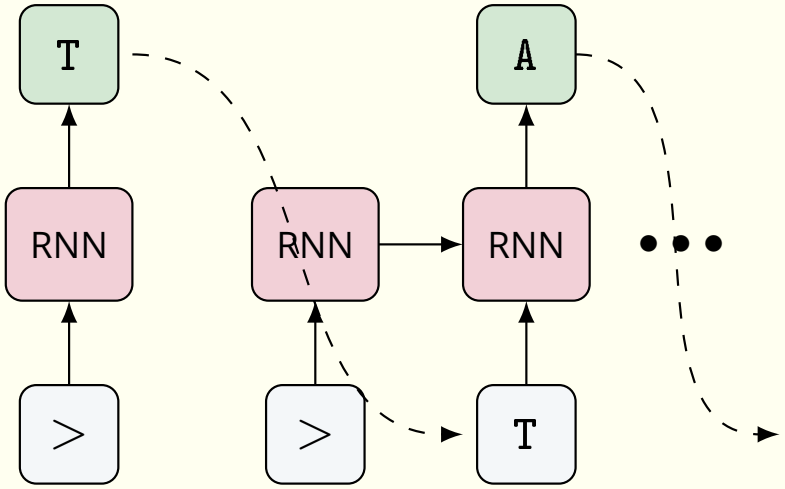
Generate new power cycles with a RNN

For training



1. The model predicts the 4th letter ('T') given the first 3 ('>TA')
2. Wrong prediction ('T' $\neq$ 'C') $\Rightarrow$ the model auto-tunes itself
3. Repeat for every cycle in the database

For generation



1. With start symbol '>' as input, draw next letter ('T') from the distribution defined by the trained model
2. Append the letter to the sequence
3. Repeat until end symbol '!' is drawn

Design constraints

Table: Supercritical CO2 CSP plant design constraints.

Symbol	Description	Value
T_{max}	maximum operating temperature	560 °C
T_{min}	minimum operating temperature	35 °C
P_{max}	maximum operating pressure	250 bar
dp_c	pressure drop (cooler)	0.5 bar
dp_{hx}	pressure drop (heater, exchanger)	1 bar
ΔT	temperature pinch (heat exchanger)	5 K
η_c	isentropic efficiency (compressor)	0.89
η_t	isentropic efficiency (turbine)	0.93
$\dot{m}$	molten salt mass flow	80 kg/s
T_1	molten salt minimum temperature	290 °C
T_2	molten salt maximum temperature	565 °C

Vector representation of the power cycles