Systems & Control Transactions

Research Article - Peer Reviewed Conference Proceeding
ESCAPE 35 - European Symposium on Computer Aided Process Engineering
Ghent, Belgium. 6-9 July 2025
Jan F.M. Van Impe, Grégoire Léonard, Satyajeet S. Bhonsale, Monika E. Polańska, Filip Logist (Eds.)

PSE PRESS

Reinforcement learning for distillation process synthesis using transformer blocks

N. Slager[a], M.B. Franke[a,*]

[a] Sustainable Process Technology Group, Department of Chemical Engineering, Faculty of Science and Technology, University of Twente, Drienerlolaan 5, 7522 NB Enschede, The Netherlands
* Corresponding Author: m.b.franke@utwente.nl.

ABSTRACT

A reinforcement learning framework is developed for the synthesis of distillation trains. The rigorous Naphtali-Sandholm algorithm for equilibrium separation modeling was implemented in JAX and coupled with the benchmarking Jumanji RL library. The vanilla actor-critic agent was successfully trained to build distillation trains for a seven-component hydrocarbon mixture. A transformer encoder structure was used to apply self-attention over the agent's observation. The agent was trained on minimal data representation containing quantitative component flows and relative volatility parameters between present components. Training sessions involving $5{\cdot}10^4$ episodes ($3{\cdot}10^5$ column designs) were typically run in under 60 minutes. While training was fast and reliable with appropriate tuning of the hyperparameters, further improvements are needed in the generalizability performance for similar separation problems.

Keywords: Artificial Intelligence, Machine Learning, Process Synthesis, Distillation, Optimization, Reinforcement learning, Transformer Blocks

INTRODUCTION

The field of process systems engineering (PSE) has witnessed significant advancements through the combination of machine learning (ML) techniques with traditional PSE methods. Among these, reinforcement learning (RL) has emerged as a promising approach [1], due to its capacity to tackle sequential decision-making problems inherent to process synthesis. Notably, Midgley (2020) [2] pioneered RL in PSE for process synthesis with the introduction of the Distillation Gym, an open-source platform designed for optimizing distillation train configurations simulated in ChemSep/COCO for user-defined multicomponent feeds. This foundational work was further developed by van Kalmthout et al. (2022) [3], who integrated Aspen Plus for rigorous process modeling within the RL framework. However, major improvements were not achieved due to the insufficient data transfer speed and (un)reliability of the RL/Aspen Plus interface.

Considering flowsheet synthesis involving multiple unit operations and recycles, Göttl et al. (2021) [4] proposed a two-player game framework in which an RL agent competes against itself to generate profitable flowsheets involving multiple unit operations simulated using short-cut models. Building on this, Göttl et al. (2023) [5] explored the generalizability of RL methods by applying them to azeotropic distillation case studies. Using similar process simulation models, Khan et al. (2022) [6] integrated RL with mathematical programming (MP) to allow RL agents to design unit operation sequences while optimizing these sequences via MP.

Recent efforts have addressed challenges related to simulation accuracy and computational efficiency. For instance, Wang et al. (2023) [7] proposed an RL framework coupled with IDEAS as a process simulator. Gao et al. (2023) [8] addressed the issues of slow and inaccurate process simulators by introducing transfer learning. In their approach, the RL agent is pretrained on short-cut process models to facilitate rapid learning, followed by fine-tuning training on rigorous simulations using DWSIM for improved accuracy. Reynose-Donzelli et al. (2024) [9] applied policy masking techniques to simplify the RL agent's learning process while leveraging Aspen Plus for simulations.

The need for process flowsheet data embedding to create meaningful data representations for the RL agent has been widely acknowledged. Stops et al. (2022) [10] emphasized the creation of meaningful flowsheet data embeddings using graph neural networks (GNNs). Göttl et al. (2023) [5] used MLP-mixer methods. Seidenberg et al. (2023) [11] introduced knowledge graphs to represent flowsheet object-relation dynamics more concretely. Wang et al. (2023) [7] exploited the well-established convolutional neural network (CNN) agent architecture.

These studies collectively underscore the versatility and potential of RL in PSE, from distillation train optimization to generalized flowsheet synthesis and novel approaches to data representation and simulation efficiency. This growing body of work highlights RL's promise in addressing complex, real-world challenges in process synthesis engineering. Yet, challenges remain both in developing a thorough understanding of the RL agent's performance and integration of fast, reliable, and accurate process simulators in the proposed RL applications.

This work continues the search for optimal RL performance through combining rigorous process modeling with the existing open-source benchmarking Jumanji [12] platform for RL applications, leveraging the use of state-of-the-art RL agent features combined with effective data embedding through transformer blocks. Jumanji is developed in JAX, which is a layer on top of Python and developed for machine learning purposes by Google DeepMind. JAX provides explicit features including parallel computing on GPU backends, Just-In-Time (JIT) compilation, and automatic differentiation. These features are used in this work to simulate equilibrium-based separation processes in JAX, seamlessly integrating with Jumanji.

The choice for transformer data embedding as an alternative to the aforementioned embedding techniques has been made to explore its capacity for the given purpose and optimally exploit parallel computing in JAX. It is explored how the transformer architecture operates on a variety of information representation in terms of quality and quantity.

METHODOLOGY

The work involved the implementation of the rigorous Naphtali and Sandholm (N&S) algorithm into JAX for simulating multi-component equilibrium separation and coupling this algorithm with the RL framework for distillation train sequencing. The N&S algorithm is an equation-grouping-by-stages procedure and exploits the NR iteration for finding the non-linear solution vectors. Finding the Jacobian for the NR iterations either numerically or analytically is traditionally a tedious process involving significant computational effort. In the current work, the JAX library has been leveraged to use automatic differentiation (AD) for finding the Jacobian in a highly efficient and accurate way.

Table 1: Data representation in the initial observation by the RL agent.

Component	**C3**	**I-C4**	**C4**	**C5**
Fraction	0.143	0.143	0.143	0.143
Component	**C-C5**	**C6**	**T-C4**	$\mathbf{R_{tot}}$*****
Fraction	0.143	0.143	0.143	1

*R_{tot} gives the ratio of the current total stream quantity over the initial feed to the train instead of the current stream absolute flow value to minimize steep gradients during training.

RL framework

The RL problem is formally defined by the Markov decision process (MDP). The MDP can be described by the tuple M ={S,A,P,R} where S is the collection of states describing the environment, A contains the possible actions for the agent to take in the environment, P is the transition function describing the system dynamics via the probability of space action trajectories, and R is the reward function. The agent runs an episode which is a series of time steps. In each time step, the agent observes the current state upon which it selects an action to let the environment transition to a new state and receive a reward for the chosen action from the environment. The episode is finalized once the terminal state is reached. The agent learns to maximize the reward.

In this framework, the state observation is an available component stream to be separated. The agent decides on the column design parameters for the observed stream, after which it receive a financially based reward for the generated separation. The episode is terminated once the minimum number of columns has been completed to recover all components in the initial feed as pure products based on binary separation. This termination condition is chosen as it is deemed not reasonable to built additional columns for the presented easy-to-separate mixture.

State space

The state space contains all possibly flowsheet configurations of the multi-component separation train. State representations are stored in a stream table structure wherein the connectivity of separation units and stream information is represented. This potentially includes component flow quantities, column design parameters, column installed and operating costs, and stream revenues if a product purity criteria is met.

Observation space

Given the Markov property, the observation of a current state should provide all information relevant to the RL agent to inform its decision upon action taking. It is considered that all information to build a distillation column is contained in the quantity and quality of the feed stream. The origin of such a feed stream in terms of preceding column passes is therefore irrelevant in this approach and information about existing flowsheet topology is not contained in the observation. Table 1 gives the data presented with only the stream composition in the initial observation (see Figure 1).

Table 2: Design parameter windows and discretization for the multi-variable discrete action space

	Lower	Upper	Steps
Stage number	5	85	81
Reflux ratio	0.1	10	30
Distillate ratio	0.01	0.99	150
Feed location*	0.2	0.85	30

* The feed location is defined as the column fraction above the feed

Action space

While several literature studies focused on hierarchical action space structures, a simplified approach is used in this work. Each time step will generate at least one non-pure outflow stream as long as there are still more than two feed components present. These outflows are systematically acted upon by the agent, either until the product purity constraint is met for all streams or if the maximum number of columns allowed has been built.

For a given observed single stream, the agent must decide on the column design parameters including number of trays, distillate rate, reflux ratio, and feed location. A multi-discrete action space has been exploited where each action involves adapting the four design parameters. The chosen feasible parameter bounds and the number of discretization steps are summarized in Table 2. The given discretization gives roughly 11M possible action values combinations for a single column design. Action masking is used to control the different number of discretization steps for each design parameter. The small discretization step size for the distillate ratio was chosen to potentially allow the agent to achieve optimal purity close to the minimum requirement (95%).

Reward definition

The RL agent aims to maximize a cumulative award. It is essential to effective learning that the reward definition accurately captures the environment dynamics. For a flowsheet configuration, reward can be expressed in financial terms through the potential revenues of product streams minus the installed and operating costs.

$$Reward = Revenue - (C_{installed} + C_{operating}) \quad (4)$$

The installed costs were estimated using the Marshall & Swift index correlations and operating costs are considered for heating and cooling in the reboiler and condenser, respectively. The cost correlations for both contributions are detailed in the supplementary material.

The same revenue value is chosen for all components, regardless of possible deviations in market values to prevent introducing bias during learning. A hypothetic constant product revenue of 20$/kmol has been selected.

RL agent architecture

The agent architecture has been adopted from the Jumanji library [12] and features state-of-the-art functionalities. The actor-critic baseline agent uses bootstrapping for the objective function, preventing it from collapsing. The advantage is used for update estimates, and an entropy term is introduced to favor exploration.

Data embedding

The transformer architecture is implemented as an encoder structure with self-attention to the input sequences. The structure allows for addressing the relative importance of input parameters with respect to the overall input for the system dynamics. The transformer encoder takes in a query (Q), key (K), and value (V). Scaled dot self-attention is applied over Q and K and a SoftMax function is used to create the attention weights. The weights are multiplied with V to obtain a weighed value representation indicating the relative importance of the vector entries in Q to focus on the most relevant information from the sequence to take its next action.

Q, K, and V are all the same for the encoder transformer and contain the observation of the environment. Learnable weight matrices are introduced for Q, K, and V to transform the observation into a higher dimensional representation before entering the self-attention mechanism. Trainable parameters are the Q, K, and V weight matrices, the feedforward NN layers, and the linear transformation layers. The agent learns by calculating the update for each of the parameters via the gradient of the loss function with respect to the parameters. The parameter updates are applied for the critic and actor network.

SCOPE

A case study is selected where a multi-component hydrocarbon mixture needs to be separated into >95% pure product streams. The components are listed in Table 3 and have been chosen for their reasonable relative volatilities, rather than representing a potential real system. Non-ideality is not considered in the algorithm. Feed streams are always considered to be liquid at boiling point. Heating and cooling duty costs are based on standard cooling water, even though this may not be realistic for the light boiling components present in the initial feed.

The rigorous algorithm takes in the number of stages, feed location, distillate rate, reflux ratio, feed rate, operating pressure, and feed composition. The algorithm has been tested for its reliability and speed as function of number of components in multi-component systems. It was chosen to initiate random feeds with and find reasonable column designs for these feeds using the Fenske-Underwood-Gilliland shortcut method. These column design were then evaluated using the rigorous algorithm. For batches containing 1000 random initiated feeds for a given number of components, it was found that full convergence for all columns in a batch could no longer be guaranteed with more than nine components present. It was deemed reasonable to verify the RL framework performance using a seven-component mixture to introduce separation complexity while guaranteeing reliable and stable performance of the separation algorithm.

Table 3: Components considered in the multi-component separation train.

Component	BP@1 atm	CAS	Not.
propane	-42 ^oC	74-94-6	C3
isobutane	-12 ^oC	72-28-5	I-C4
n-butane	-1 ^oC	106-97-8	C4
n-pentane	36 ^oC	109-66-0	C5
cyclopentane	49 ^oC	287-92-3	C-C5
n-hexane	69 ^oC	110-54-3	C6
2,2,3-trimethyl butane	81 ^oC	464-06-2	T-C4

Table 4: Hyperparameter settings used for the RL framework.

Agent		**Transformer network**	
Learning rate	0.4e-3	Blocks	1
Bootstrap factor	0.95	Heads	1/2/4
Entropy term	0.1	Key size	32
Discount factor	1.0	MLP-layer size	512
Training			
Batch size	32		
Episode steps	6		

RESULTS

The goal for the agent was to construct a flowsheet in which all seven components are recovered as purified product streams using six separation units producing 2000 kmol/hr total pure product. When trained on a fixed feed composition, the agent was well capable of doing so, given that the framework hyperparameters were appropriately chosen. While the hyperparameter settings can be tuned over a wide range, findings for fixed sets of the hyperparameter as given in Table 4 are presented. The choice for these fixed parameter values is informed by the settings for different games with similar RL networks as provided within the Jumanji library.

Figure 1 shows the episode return as function of episode number or time. In the base case, training is done on basis of quantitative component flow observation (ϕ)

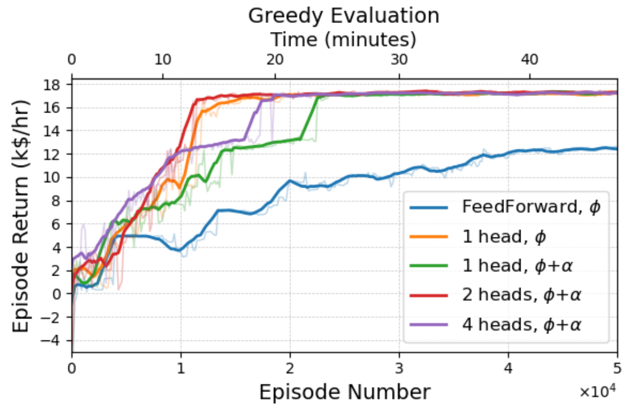


Figure 1: Episode return using argmax over the policy parameters as function of episode number (lower axis) or time (upper axis). ϕ for training on quantitative component flows observation. $\phi + \alpha$ for training on additional relative volatility data.

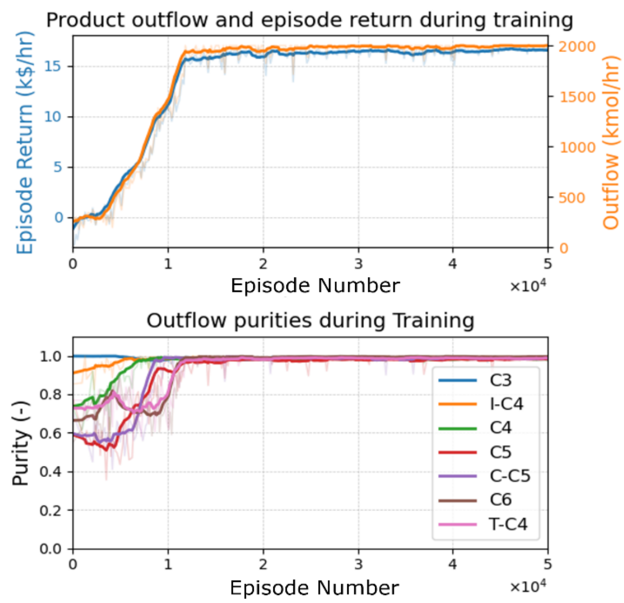


Figure 2: Training progression profiles in terms of cumulative episode return (Fig 2A), total product recovery during training (Fig 2A), and outflow purity development during training (Fig 2B) for 2 heads $\phi + \alpha$ (Figure 1).

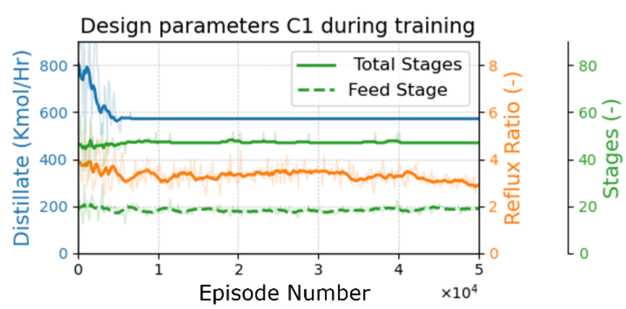


Figure 3: Separation column design parameter of the first column in the sequence profiles during training.

using single head attention. In the absence of a transformer encoder (feedforward network), the agent was not able to learn how to produce all outflows as products with six columns. The impact of adding the relative volatilities to the observation space is illustrated ($\phi+\alpha$) in combination with the number of attention heads. It is observed that the fastest learning was achieved for the observation space of form $\phi+\alpha$ in combination with two heads. The effectiveness of multi-head attention is however not easily quantified as also the training parameter set size changes upon adding additional attention heads in the current setup. Having four attention heads shows decreased learning performance indicated by the slowdown in convergence towards full product recovery.

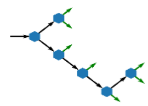
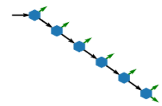
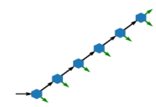
Figure 2A shows the total molar outflow of streams being product streams and the episode return. The episode return corresponds to the cumulative reward received by the agent for completing one flowchart of 6 columns. The product outflow strongly correlates with the episode return as the agent receives high reward for producing product streams. It can be expected that the reward increases if the agent would more closely approach the purity constraint by tuning the design parameters for a converged flowsheet. However, such reward optimization is hardly observed for the current training (flat purity profiles once seven components are recovered as product streams), indicating the potential need for reward shaping to achieve better static optimization. Figure 2B showcases, however, that the agent was able to learn the purity constraint of >95 mol% on product flows, based on the reward definition.

Figure 3 provides an exemplary plot based on the first column in the flowchart for the learned design parameter tuning by the agent. The variation remains larger for the reflux ratio, and feed location which is due to the larger discretization steps for both parameters (see Table 1).

Flowchart benchmarking

The performance of the RL agent was validated through the generation of benchmark flowcharts in Aspen Plus. The comparison of episode return is done for the flowchart generated by the agent and two flowcharts where either all distillate or bottom streams are >95% pure products. The comparison is summarized in Table 3. The column number of stages, feed locations, and distillate rates are determined using the DWSTU model in Aspen Plus. The reflux is determined using a purity design spec after converting the DWSTU models into RADFRAC models. The comparison shows that the episode return obtained by the agent is competitive with the values found for the benchmark flowcharts. It must be noted that the DWSTU involves a shortcut method for which the dynamics might be different compared to the reward function on which the agent was trained.

Table 5: Flowchart episode return comparison between RL-generated and Aspen Plus optimized designs and learned flowsheet column design parameters. RR is reflux ratio, D is distillate rate (kmol/hr), N_{tot} and N_f is total and feed stage number, respectively.

RL	**Distillates**	**Bottoms**
		
17.3 k$/hr	17.2 k$/hr	15.8 k$/hr

Column design parameters RL flowchart

	RR	D	N_{tot}	N_f
C1	2.8	572	47	19
C2	1.6	296	82	51
C3	1.7	296	35	20
C4	3.2	287	42	13
C5	2.3	570	55	24
C6	2.3	283	79	35

* Using argmax over the policy params

Inference

RL is believed to be capable of outperforming other optimization approaches for its inference properties. Where traditional approaches optimize problem-specific, the RL approach involves the optimization of its policy to make the right decisions under the given circumstances. As such, new problems following a similar problem definition as the problem trained for may be solvable by the RL agent using its trained policy. This inference property has been tested in the current work to provide some insight into how well the agent is capable of generalizing between similar problems. Since the agent was trained on a fixed initial feed composition, inference can be tested by introducing random distributions for the feed composition and see how well the agent is capable of obtaining an optimal reward for these distributions.

Table 6: Inference capacity of the trained networks in Figure 1.

Policy	**Mean reward** (k$/h)	**Mean product outflow** (kmol/h)
$\boldsymbol{random}$	-1.9	470
$\boldsymbol{1\ head, \phi}$	3.3	504
$\boldsymbol{1\ head, \phi+\alpha}$	3.8	562
$\boldsymbol{2\ head, \phi+\alpha}$	2.6	463
$\boldsymbol{4\ head, \phi+\alpha}$	3.8	532

The episode return for 100 randomly sampled feed distributions with a minimum fraction of 0.05 for each component was evaluated for the trained policies in Figure 1. The results are in Table 6. The trained policy outperforms the random policy in terms of episode return but there is hardly a gain in product recovery and also no strong correlation between number of heads vs performance is found.

CONCLUSIONS

With the implementation of rigorous modeling into JAX, seamlessly integrated with the Jumanji framework, this work demonstrates the potential of combining RL with rigorous simulation methods for process synthesis. The RL agent successfully handles the complexity of the optimization problem, managing an extensive action space containing 11 million possible action values for a single state. Flowsheet convergence toward full product recovery was typically achieved after evaluating fewer than 20,000 episodes (or six-column flowcharts). The simplified approach of using a non-hierarchical action space and single-stream observations highlights that RL does not necessarily require massive data input for optimal learning. However, the need for meaningful data representation was indicated, as flowchart convergence was not achieved without a transformer encoder. While transformer structures are renowned for their parallel multi-head attention, adding more attention heads did not necessarily improve learning performance.

Further improvements are needed to enhance the inference performance of the trained policy. While there was a slight improvement in reward gain between the trained and random policies, the overall performance remains unsatisfactory. One possible way to improve generalizability could be to train the agent on random feed distributions for given multi-component mixtures, rather than relying on fixed compositions as is currently done.

The major limitation from a process synthesis perspective lies in the feedforward structure of the generated flowchart. Process synthesis relevance could be enhanced by incorporating a variety of unit operations along with the presence of recycle streams. Additionally, challenges are likely to arise in achieving convergence of the rigorous model when non-idealities are introduced.

DIGITAL SUPPLEMENTARY MATERIAL

Supplementary information on the code accessibility, hardware usage, and reward function definition can be found at https://psecommunity.org/LAPSE:2025.0026.

REFERENCE

1. Mowbray, M. et al. Industrial data science-a review of machine learning applications for chemical and process industries. React. Chem. Eng. 7:141-1509 (2022) https://doi.org/10.1039/D1RE00541C
2. Midgley LI. Deep reinforcement learning for process synthesis. arXiv preprint (2020) arXiv:2009.13265
3. van Kalmthout SC, Midgley LI, and Franke MB. Synthesis of separation processes with reinforcement learning. arXiv preprint 4:32-42 (2022) arXiv:2211.04327, 2022
4. Göttl Q, et al. Automated Flowsheet Synthesis Using Hierarchical Reinforcement Learning: Proof of Concept. Chem. Ing. Tech. 4:32-42 (2020) https://doi.org/10.1234/56789.10
5. Göttl Q, et al. Deep reinforcement learning uncovers processes for separating azeotropic mixtures without prior knowledge. arXiv preprint (2023) arXiv:2310.06415, 2023
6. Khan AA, Lapkin, AA. Designing the process designer: Hierarchical reinforcement learning for optimisation-based process design. Chem Eng Process Process. Intensif. 180:108885 (2022) https://doi.org/10.1016/j.cep.2022.108885
7. Wang D, et al. A coupled reinforcement learning and IDAES process modeling framework for automated conceptual design of energy and chemical systems.. Energy Adv. 2:1735-1751 (2023) https://doi.org/10.1039/D3YA00310H
8. Gao Q, et al. Transfer learning for process design with reinforcement learning. Comput. Aided Chem. Eng. 52:2005-2010 (2023) https://doi.org/10.1016/B978-0-443-15274-0.50319-X
9. Reynoso-Donzelli S, Ricardez-Sandoval LA. A reinforcement learning approach with masked agents for chemical process flowsheet design. AIChE Journal 71:e18584 (2024) https://doi.org/10.1002/aic.18584
10. Stops L, et al. Flowsheet generation through hierarchical reinforcement learning and graph neural networks. AIChE Journal 69:e17938 (2023) https://doi.org/10.1002/aic.17938
11. Seidenberg JR, Khan AA, Lapkin, AA. Boosting autonomous process design and intensification with formalized domain knowledge. Comput. Chem. Eng. 169:108097 (2023) https://doi.org/10.1016/j.compchemeng.2022.108097
12. Bonnet C, et al. Jumanji: a diverse suite of scalable reinforcement learning environments in JAX. arXiv preprint (2023) arXiv:2306.09884