NEXOGENESIS
STREAMLINING WATER RELATED POLICIES

# D4.4 Core module of the self-learning nexus engine

**Lead: Lluís Echeverria (Eurecat)**

Date : 30/06/2024

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

# Project Deliverable

| Project Number | Project Acronym | Project Title |
|---|---|---|
| 101003881 | NEXOGENESIS | Facilitating the next generation of effective and intelligent water-related policies, utilizing artificial intelligence and reinforcement learning to assess the water-energy-food-ecosystem (WEFE) nexus |

| Instrument: | Thematic Priority |
|---|---|
| H2020 RIA | LC-CLA-14-2020 |

| Title |
|---|
| **D4.4 Core module of the self-learning nexus engine** |

| Contractual Delivery Date | Actual Delivery Date |
|---|---|
| M34: June 2024 | M34 |

| Start Date of the project | Duration |
|---|---|
| 01 September 2021 | 48 months |

| Organisation name of lead contractor for this deliverable | Document version |
|---|---|
| EUT | 1.0 |

| Dissemination level | Deliverable Type |
|---|---|
| **Public** | **Demonstrator** |



| Authors (organisations) |
|---|
| Lluís Echeverria (EUT), Chaymaa Dkouk (EUT), and Nuria Nievas (EUT) |
| **Reviewers (organisations)** |
| Lydia Vamvakeridou-Lyroudia (KWR) |





This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

## Abstract

Deliverable D4.4 *Core module of the self-learning nexus engine* is classified as a "Demonstrator." This document accompanies the deliverable and offers detailed explanations about the algorithmic fundaments and technical aspects of the Self-Learning Nexus Assessment Engine (SLNAE) and the NXG DSS. It is intended to complement the digital solutions developed from Task T4.4 *Reinforcement Learning engine.*

The self-learning nexus engine is the core mechanism that supports multi-objective decision-making in the SLNAE tool. The self-learning term refers to the underlying Artificial Intelligence (AI) and Machine Learning (ML) technology, enabling the creation of agents that autonomously learn (i.e. self-learning) optimal policy combinations (i.e. policy packages) to achieve the nexus-related objectives. We propose Multi Objective (Deep) Reinforcement Learning (MODRL) as the foundational family of ML algorithms to implement the NXG Decision Support System (DSS).

The current version of the Self-Learning Nexus Assessment Engine is embedded into the public release of the SLNAE at the following urls: https://slnae-dev.nexogenesis.eu or https://nepat-dev.nexogenesis.eu.

The SLNAE acronym, which is the reference to this tool in the Grant Agreement, has been changed to NEPAT (Nexus Policy Assessment Tool), during the project. This was decided because it was easier to pronounce than SLNAE and also it reflected better for the general audience the content of the tool. In this document, the two acronyms SLNAE and NEPAT are both being used indiscriminately and they refer to the same tool. This has been done on purpose, because SLNAE is mentioned in the Grant Agreement and also because several related actions have started and happened under with the tool named SLNAE, while now, more recent activities are referring to NEPAT and this document is an intermediate report for this tool. By the end of the project, NEPAT will be the name of this tool.

Related Deliverables:
D3.4 Complexity science models implemented for all the Case Studies including explanatory manuals
D3.6 Sensitivity/Uncertainty Analysis Report
D4.1 Self-learning nexus engine specifications and technical design
D4.3. Simulation Policy Framework



**Keywords**
SLNAE; NEPAT, Nexus Decision Support System, Multi-Objective optimization, Multi-Objective Deep Reinforcement Learning




This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

# Abbreviation/Acronyms

| | |
|---|---|
| CCS | Convex Coverage Set |
| CS | Case Study |
| DMORL | Deep Multi-Objective reinforcement learning |
| DMP | Data Management Plan |
| DRL | Deep Reinforcement Learning |
| DSS | Decision Support System |
| FNRL | Fingerprint networked reinforcement learning |
| GUI/UI | Graphical User Interface/User Interface |
| GA | Grant Agreement |
| ICT | Information and Communication Technologies |
| IDR | Internal Data Repository |
| JSON | JavaScript Object Notation |
| MDP | Markov Decision Process. |
| ML | Machine Learning |
| MOMDP | Multi-Objective Markov Decision Process |
| MOO | Multi-objective optimisation |
| MORL | Multi-objective reinforcement learning |
| NEPAT | Nexus Policy Assessment Tool |
| NXG | Nexogenesis project |
| PCS | Pareto Coverage Set |
| POMOMDP | Partially Observable Multi-Objective Markov Decision Process |
| PQL | Pareto Q learning |
| R | Reward function |
| RL | Reinforcement Learning |
| SDM | System Dynamic Model |
| SLNAE | Self-Learning Nexus Assessment Engine |
| SH | Stakeholder |
| RP | Reference Pathway |
| WEF | Water-Energy-Food (Nexus) |
| WEFE | Water-Energy-Food-Ecosystem (Nexus) |
| WP | Work Package |


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

# Contents




This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

# Figures





This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

# 1. Introduction

The WEFE nexus framework highlights the intricate linkages between water, energy, food and ecosystems dominated by complexity and modulated by climatic and socio-economic drivers. For instance, water is essential for agriculture (food production) and energy generation (hydropower and cooling in thermal power plants). Conversely, energy is needed for water extraction, treatment, and distribution, as well as for food production and processing. This interconnectedness creates a web of dependencies where actions in one sector can have significant ripple effects across others.

In this context, decision-making face a multi-objective (MO) problem due to the complex and often conflicting objectives inherent in managing these resources. Addressing the nexus requires balancing the competing demands of each sector while considering their interrelated impacts on sustainability, economic growth, and social well-being.

Instead of aggregating the various nexus objectives into a single scalar signal/objective for planning or learning purposes, we consider them individually. This approach allows the end user to define their own aggregation function, technically known as the utility function, based on their preferences. By doing so, we avoid providing an imperfect solution that would restrict the tool's recommendation scope. We also give more “freedom” to the users to express their preferences.

The self-learning nexus engine is the core mechanism that supports MO decision-making in the SLNAE tool. The self-learning term refers to the underlying Artificial Intelligence (AI) and Machine Learning (ML) technology, enabling the creation of agents that autonomously learn (i.e. self-learning) optimal policy combinations (i.e. policy packages) to achieve the nexus-related objectives. We propose Multi Objective (Deep) Reinforcement Learning (MODRL) as the foundational family of ML algorithms to implement the NXG Decision Support System (DSS).

Each Case Study (CS) represents a unique optimization challenge, formulated as an individual problem to optimise in their own, unique, policy decision space. Additionally, there are three further layers of complexity. First, each CS includes a set of reference scenarios (RCP-SSP combinations), each depicting different potential future conditions. Second, the complexity models implemented by WP3 incorporate randomness in the input data, allowing for both deterministic and stochastic execution modes. Finally, one of the CSs (Inkomati) introduces an additional decision-making dimension: the year when a policy is applied, which we call the ‘dynamic policies’ mode, in contrast to the ‘static mode’ with fixed policies. All these options are available through different implementations of the CSs’ System Dynamic Models (SDMs). Consequently, an agent will be trained for every combination of CS, reference scenario, randomness execution mode, and, for the Inkomati CS, i.e., the policy mode.


This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 101003881

Deliverable D4.4 *Core module of the self-learning nexus engine* is classified in th GA as a "Demonstrator." This document accompanies the deliverable and offers detailed explanations about the algorithmic fundaments and technical aspects of the Self-Learning Nexus Engine and the NXG DSS. It is intended to complement the digital solutions developed from Task T4.4 *Reinforcement Learning engine.*

The current version of the Self-Learning Nexus Engine is embedded into the public release of the SLNAE at the following urls: https://slnae-dev.nexogenesis.eu or https://nepat-dev.nexogenesis.eu. This version will be further updated with data from each CS.

## 1.1. Disclaimer

The public version of the SLNAE/NEPAT is currently under development and has been designated as a Beta version (Figure 1). All its modules, including the Self-Learning Nexus Engine and its components or inputs (i.e. SDMs, policies, goals, UI, etc) are undergoing continuous validation and are subject to change. **The results and screenshots in this deliverable are provided solely to demonstrate that the Self-Learning Nexus Engine has been successfully developed and integrated into the SLNAE**.

Following the validation process, in which CSs and SHs will certify the behaviour of the SLNAE/NEPAT modules, the final version of the tool will be deployed. In the upcoming months, the Self-Learning Nexus Engine will be continuously run and adjusted to meet the final requirements of CSs and SHs. This process will be expedited depending on the CS. For the front runners, the Self-Learning Nexus Engine will be ready by August 2024 (M36), including the Inkomati CS. For the followers, it will be ready by December 2024 (M40).

The final version of the SLNAE/NEPAT is expected to be ready by February 2025 (M42) and will be reported in D4.5 *Final version of the self-assessment nexus engine with the corresponding validation* (M42). This final version will include additional secondary functionalities that are not necessary for the success of the upcoming Workshops (WSs). with the CS, to be organised in collaboration with WP1 and WP5.


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

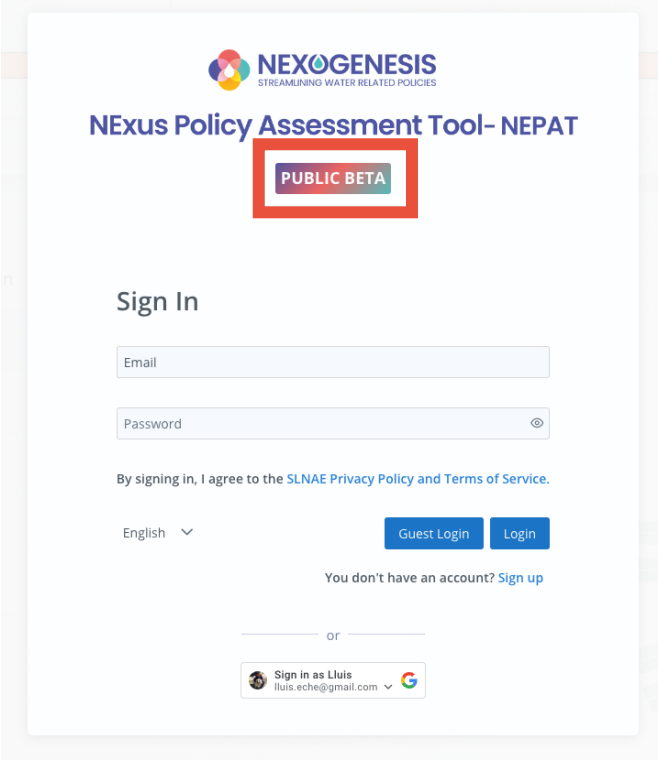


*Figure 1. SLNAE UI. Beta version warning.*

## 1.2. Links to the ICT4WATER Cluster

WP4 is considered the 'digital' WP in the Nexogenesis project. Thus, it is the natural link between the project and the ICT4WATER Cluster[1].

The outcomes of the task T4.4 are specially linked to the ICT4WATER Updated Digital Water Action Plan[2] and its 'Intelligent and smart systems', 'Actor engagement and co-creation' and 'Policies' Action Groups[3].

With regards to the actions and activities outlined in the Digital Water Action Plan, this deliverable and the SLNAE/NEPAT tool (developed under WP4 in NXG) contributes to the following (all action numbers refer to the Digital Water Action Plan):

- The 'Intelligent and smart systems' action 2 activities 1 & 2: A multi-optimization decision-making framework (the Self-Learning Nexus Engine), based on Deep Reinforcement Learning, is implemented for decision support.
- The 'Intelligent and smart systems' action 5 activities 1 & 2: Uncertainty is taken into account in the integrated complexity science models.

[1] https://ict4water.eu/
[2] https://ict4water.eu/wp-content/uploads/2023/06/Update-Digital-Water-Action-Plan-V7.pdf
[3] https://ict4water.eu/action-groups/


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

- The ‘Actor engagement and co-creation’ action 1 activity 1: A public online tool (the SLNAE) is developed.
- The ‘Actor engagement and co-creation’ action 2 activities 1 & 2: Stakeholders from all nexus sectors are taken into account during the Nexogenesis co-creation process for the SLNAE design.
- The ‘Policies’ action 5 activity 1: The SLNAE tool enables policy co-creation.
- The ‘Policies’ action 6 activity 2: The SLNAE tool enables improved management of governance complexity including uncertainty and other factors.

## 1.3. NEPAT: a new name for the SLNAE

During the initial period of the project, it was necessary to explain the definition and meaning of the SLNAE to non-technical audiences (e.g. stakeholders) on multiple occasions. The term "Self-Learning" is a technical concept related to AI and ML algorithms involved in its development, which can be difficult to understand and may generate confusion. Additionally, the acronym SLNAE and the full name are not easy to pronounce. Therefore, it was decided with the NXG consortium that a new name was needed.

WP4 led the initiative to define a new name that would be self-explanatory and avoid technical jargon, while incorporating Nexogenesis-related concepts such as "policy assessment" or "impact". Several options were proposed under the cocreation framework and put to a vote, and eventually, the name "Nexogenesis - Nexus Policy Assessment Tool (NEPAT)" was selected as the best option. The new name is simpler, easier to pronounce, and more reflective of the tool's and project’s purpose. In order to be consistent with the GA and other official documentation, both names are valid to refer to the SLNAE.

Thus, the SLNAE tool is referred to as either SLNAE or NEPAT indiscriminately.

## 1.4. Document structure

The document is structured as follows: Section 2 describes the requirements for the Self-Learning Nexus Engine implementation. Section 3 briefly reviews the state of the art in multi-objective optimization and multi-objective reinforcement learning within the nexus domain, highlighting the advantages of using reinforcement learning in the NXG DSS. Section 4 establishes the foundational knowledge and mathematical formulation in the multi-objective optimization domain and introduces the selected MORL algorithms. Section 5 presents the technical design of the NXG decision-making problem and its formalization. Section 6 showcases the initial results of these implementations. Finally, Section 7 demonstrates the current integration of the Self-Learning Nexus Engine into the NEPAT UI, and Section 8 concludes with the findings and outlines the next steps.


This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 101003881

# 2. Requirements for the development of the self-learning nexus engine

In order to develop the NEPAT's self-learning nexus engine and DSS (step 3 in Figure 2 - Right), the key following components (Figure 2 - Left) have been previously developed (steps 1 and 2), by other technical WPs, through the co-creation framework (Figure 2 - Right) for Nexus Policy packages identification:

- Policies
- Nexus Indicators
- Nexus Policy Goals, years and targets
- System Dynamics Models (SDM)s

All these components have been provided by each of the five NXG CSs and have been integrated into the NXG Simulation Policy Framework (see deliverable D4.3 *Simulation Policy Framework*). The aim of the DSS is to identify which policy combinations (i.e., policy packages) are suitable for achieving the goals' targets. This objective is further explained and technically formalized later in this document. To measure the impact of each policy package, an SDM is required. Policies have been implemented in the SDMs, so their impact and trade-offs can be quantitatively measured using the Nexus indicators, which are also implemented in the SDMs. A Nexus Goal is linked to one of these indicators, making it measurable.

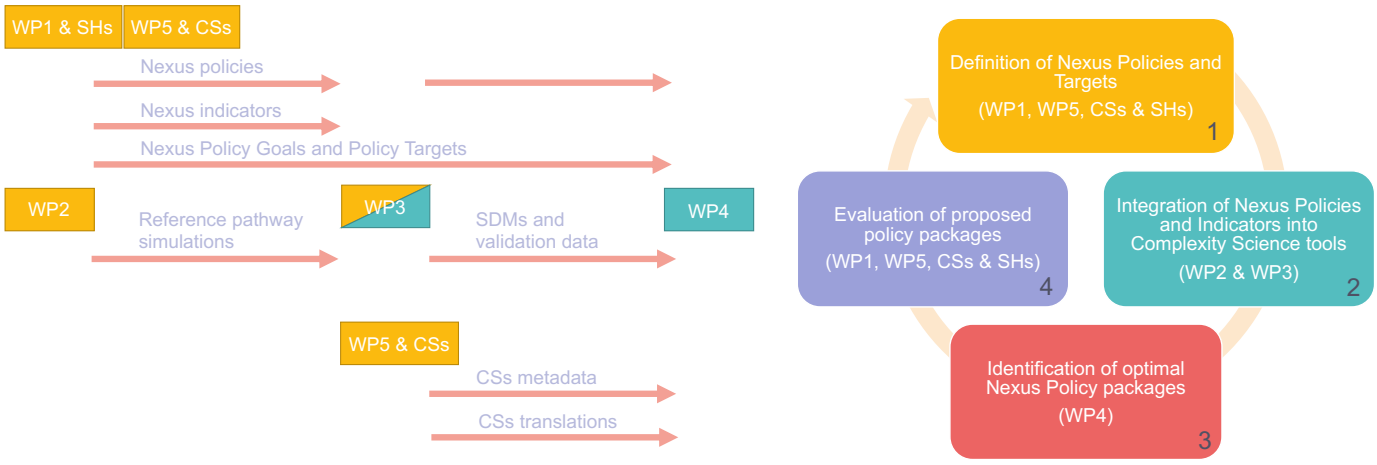


*Figure 2. Left: NXG cross-WP data pipelines in the Internal Data Repository. Right: NXG co-creation framework for Nexus Policy packages identification. Source: D4.1*

All these components are available in the Nexogenesis Internal Data Repository or Data Lake (see D4.2 *Data Lake for data sharing*). WP4 collects all these resources and uses them to develop the NEPAT.


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

## 2.1. Policies

Each CS has proposed an arbitrary number of policies, based on their interest and goals, and the final analysis they want to carry out. Table 1 summarizes the most relevant information, from a decision-making multi-objective point of view, associated with the available policies, as how many of them have been proposed, how many have been finally implemented or its type, among other information.

*Table 1. NEPAT available policies*

| CS | N. of regions | N. of proposed policies | N. of proposed policies at region level | N. of implemented policies | Incompatible policies | Type of policies (Fixed/Dynamic) |
|---|---|---|---|---|---|---|
| **Nestos** | 2+1 | 10 | 5 | 12 | No | F |
| **Lielupe** | 2+1 | 6 | 5 | 24 | Yes | F |
| **Jiu** | 1 | 7 | - | 18 | Yes | F |
| **Adige** | 1 | 12 | - | 12 | No | F |
| **Inkomati** | 1 | 11 | - | 11 | No | F/D |

Given that some CS are transboundary, there are further considerations regarding policies, because the countries involved in each CS may have/select different policies, applying only to their subregion. Hence, there are two CSs (Nestos and Lielupe) that have split their implementation and offer sub-regions, or sub-basins, in their SDMs. In Lielupe, the regions of Latvia and Lithuania have been separately considered as the system sub-regions. In Nestos, although there are 14 sub-basins identified in the SDMs, policies can only be applied to the sub-regions of Bulgaria or Greece. In these cases, we consider there to be three regions: two sub-regions plus the case study as a whole. Therefore, some policies can be applied at a regional level, allowing decision-making and trade-off analysis at a spatial scale as well.

In some cases, CSs have proposed policies that can be modulated (Figure 3). For example, Policy P5 in the Lielupe CS, targets the reduction of Greenhouse Gas (GHG) emissions. This policy affects the fraction of grasslands to renewables parameter in the SDMs, and it is implemented as a range between 0 and 1, or as a percentage. Initially, we discretize this range into three values: 0.33, 0.66, and 1, resulting in three additional policies. Thus, technically speaking, there may be more implemented policies than those originally proposed. Later, this discretization can be modified based on SHs' feedback. Continuing with this example, these additional policies cannot be applied simultaneously because they modulate the same parameter and correspond to the same real policy. We represent this restriction by marking them as incompatible policies.


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

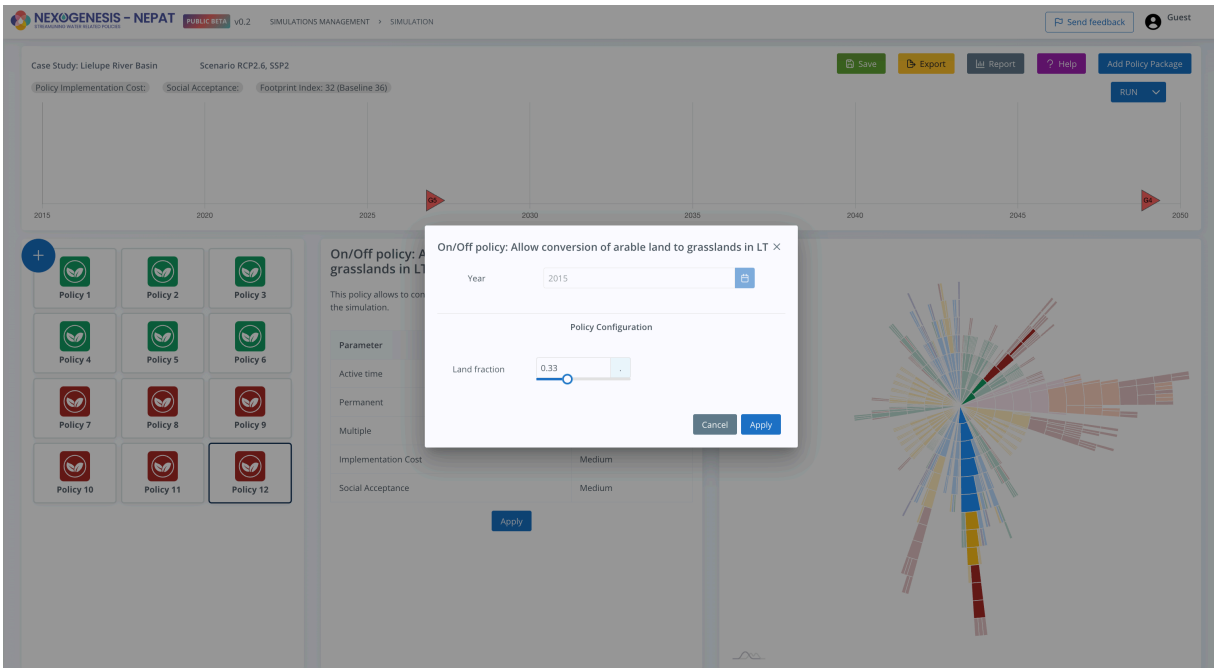


*Figure 3. NEPAT UI. Example of Policy modulation*

In other cases, like the Nestos CS, there were some policies which were described as appliable to the whole basin, while being implemented in the SDM as two variables (one for each region). In these cases, we split the policy into one for each region.

Finally, there are several ways to classify policies, such as by the direct nexus sector affected, the indirect sectors, or the way the policies function. Here, we classify the proposed policies based on whether they can be applied at any custom time between 2015 and 2050 or not. In the former case, we mark them as 'Dynamic policies', enabling decision-making and trade-off analysis at a temporal scale, and in the latter case as 'Fixed policies'.

Additional information about policies, policy integration, and the policy simulation framework can be found in D4.3 *Simulation Policy Framework*.

## 2.2. Policy packages

The number of possible policy combinations (i.e. policy packages), considering both the number of implemented policies and their compatibility, can be computed approximately. Table 2 provides an overview of these possible policy packages, illustrating the variety of combinations available based on the given policies. These computations include both valid and invalid policy packages, since the presented algorithms in section 4.3 don't have action masking implemented, meaning all combinations, whether valid or invalid, can be explored by the agents during the training phase. Hence, these would be the numbers that accurately describe the magnitude of the problem for each case study.


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

*Table 2. CS possible combinations with fixed policies*

| CS | Policy combinations |
|---|---|
| **Nestos** | 28,672 |
| **Lielupe** | 218,103,808 |
| **Jiu** | 2,621,440 |
| **Adige** | 28,672 |
| **Inkomati** | 13,312 |

From all the presented CSs, Inkomati provided SDMs with dynamic policies (Table 1), meaning that they can be configured to start their application in any set time in the simulation. For this case, we have computed the combinations for three scenarios, ordered by the granularity of the policy application time:

- Policy application every 5 years: meaning each policy can be applied in 2015, 2020, and so on up to 2050, providing 7 possible years to choose from.
- Policy application yearly: each policy can be applied in any of the 35 years of the simulation.
- Policy application monthly: each policy can be applied in any of the 420 months of the simulation.

*Table 3. Inkomati combinations with dynamic policies*

| Scenario | Policy combinations |
|---|---|
| Policy application every 5 years | 5.97E+24 |
| Policy application yearly | 1.52E+118 |
| Policy application monthly | 1.32E+1394 |

Table 3 shows the number of combinations for each of the considered scenarios. Given the size and complexity of these combination counts, tabular methods are impractical due to their inability to scale to these magnitudes. Therefore, we will employ Deep Reinforcement Learning (Deep RL) algorithms, which are better suited for handling large-scale problems. Although the second and third scenarios are highly detailed and case-specific, possibly with no real nexus application, they exemplify a significant level of complexity. Therefore, they can effectively highlight and demonstrate the advantages of the methodology introduced in the NXG project, particularly in this deliverable.

## 2.3. Goals

Each CS has proposed an arbitrary number of goals as well, based on its interest and objectives, and the final analysis they want to carry out. Table 4 summarizes the most relevant information, from a decision-making multi-objective point of view, associated with the available goals, as how many of them have been proposed or how many have been finally implemented, among other information.


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

*Table 4. NEPAT available goals*

| CS | N. of proposed goals | N. of proposed goals at region level | N. implemented goals | N. of goals affected by stochastic variables |
|---|---|---|---|---|
| **Nestos** | 5 | 5 | 30 | - |
| **Lielupe** | 4 | 2 | 5 | 1 |
| **Jiu** | 7 | - | 9 | 0 |
| **Adige** | 3 | - | 3 | 1 |
| **Inkomati** | 14 | - | 14 | 9 |

Similarly to the policies: i) goals can also be set at the sub-region level, and ii) there may be more implemented goals than originally proposed. For example, Goal G1 in the Jiu CS aims for an 87% reduction in GHG emissions by 2030 and a 97% reduction by 2050. Two additional goals have been implemented, each corresponding to one of these pairs of years and targets.

Finally, it is important to understand whether the proposed goals are influenced by the underlying data stochasticity implemented in the SDMs for proper consideration.

Additional information about goals, goals integration, and the policy simulation framework can be found in D4.3 *Simulation Policy Framework*.

### 2.3.1. Goal targets and years

Goals are defined as a three-tuple structure. First, the SDM indicator linked to the goal, which must be used to measure its performance and achievement. Second, the target value (e.g. an absolute amount, a percentage) is used as a threshold to evaluate the achievement of the corresponding goal. And third, a specific year between 2015 and 2050 when the goal achievement must be evaluated.

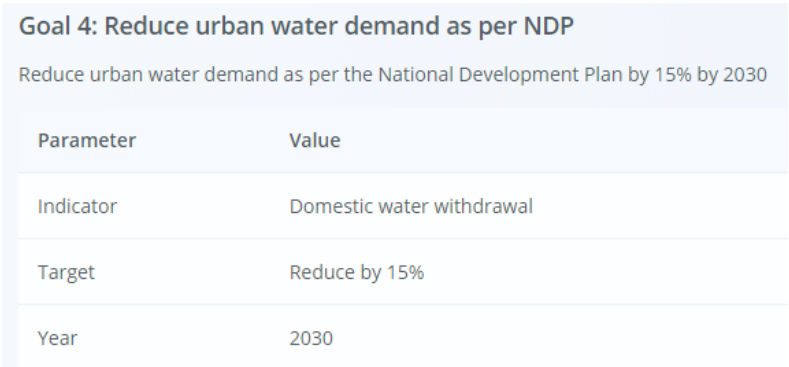


*Figure 4, NEPAT UI goal attributes view*

For example, in the Inkomati CS, Goal 4 is defined by the indicator "Domestic water withdrawal," with a target value of 15% to be achieved by a specified year. Figure 4 illustrates how this goal is represented in the NEPAT UI.


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

# 3. A nexus multi-objective approach

Multi-objective optimization (MOO), also known as multi-criteria or multi-attribute optimization, deals with problems involving more than one objective function to be optimized simultaneously. These problems are ubiquitous in engineering, economics, logistics, and other fields where trade-offs between conflicting objectives need to be made, such as the nexus.

MOO is an "old" research area, dating back to early 1980s [1] or before. Classical methods in MOO include the weighted sum method [2], epsilon-constraint method [3], and goal programming [4] Although they are simple, they also present some limitations. Metaheuristic approaches are used to solve MMO problems as well, they include Particle Swarm Optimization (PSO)[5], Ant Colony Optimization (ACO)[6], and Simulated Annealing (SA)[7]. Specifically, evolutionary algorithms (EAs) have become popular in MOO due to their ability to handle complex, non-linear, and multi-modal objective spaces. Notable EAs include genetic algorithms (GA)[8], the Non-dominated Sorting Genetic Algorithm II (NSGA-II)[9], and the Strength Pareto Evolutionary Algorithm 2 (SPEA2)[10]. Finally, extended algorithms that combine the strengths of different optimization techniques have also produced interesting results, such as the MOEA/D (Multi-Objective Evolutionary Algorithm based on Decomposition)[11]. A detailed review of the State of the Art in these areas can be found in [12].

Recent studies have increasingly applied multi-objective solutions to the nexus approach, recognizing the need to address the interconnected challenges of resource management. [13] presents a decision support framework that integrates water and power system models to address multiple objectives in hydropower development. It evaluates different scenarios focusing on minimizing power deficits, supporting irrigation for food self-sufficiency, reducing flood risks, maintaining environmental flows, and maximizing power exports. It uses a multi-objective optimization algorithm based on the epsilon-constraint method. In [14], they utilize a multi-objective approach to optimize the Water-Energy-Food (WEF) nexus by integrating short- and long-term reservoir operations with irrigation ponds, significantly improving hydropower output, water supply, and food production. The NSGA-II algorithm is employed for short-term optimization, effectively balancing hydropower generation and reservoir storage during typhoon seasons.

Also in the WEF nexus, [15] developed a multi-objective modelling approach to optimize operations in irrigated agriculture. Although the objective function focuses solely on maximizing the total net economic benefits and minimizing the environmental impacts of food production, dual uncertainty of water availability is addressed. In this line, [16] presents a multi-objective genetic algorithm model to balance sustainable agricultural development goals, integrating social variables with the WEF nexus index. It aims to reduce water and energy use,


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

lower environmental impacts, and improve farmers' social status. The study highlights that sustainable cultivation patterns depend on socio-economic conditions and policy, offering a decision-making tool for policymakers to achieve sustainable agriculture. Recently, addressing the inefficiencies of traditional MOEAs in handling more than three objectives and numerous decision variables, [17] explores the application of Large-Scale Many-Objective Optimization Evolutionary Algorithms (LSMaOEAs) to the WEF nexus. Finally, [18] develops a comprehensive nexus model to manage resources efficiently without compromising any sector. The multi-objective model evaluates resource consumption and allocation across different scenarios, where pareto analysis reveals trade-offs between sectors, highlighting how priorities impact interactions.

As it is highlighted in some of the previous research, challenges in MOO include scalability (handling high-dimensional objective spaces and large decision variable), uncertainty and robustness, dynamic and interactive optimization and computational efficiency (e.g. to handle real-time applications and large-scale problems).

The objective of Task T4.4 *Reinforcement Learning engine* is to develop a flexible, real-time, and user-friendly decision support system (DSS) accessible to the general public. Classical optimization algorithms, such as metaheuristic approaches and genetic algorithms, require significant time and computational resources to provide an answer each time it is requested (i.e., each time a user presses the 'Give me advice' button in the NEPAT UI). This presents two main issues. First, a reduced user experience. NEPAT's users would have to wait a considerable amount of time to receive policy-package recommendations, resulting in poor interaction and a negative experience with the tool. Second, an increased operational cost. To moderate the first issue and reduce computational time, we could include additional computational resources. However, this approach would lead to increased operational expenses.

In order to mitigate these issues and additional key algorithmic limitations, we propose a hybrid approach that combines multi-objective (MO) optimization with Machine Learning (ML) methodologies, specifically from the Reinforcement Learning (RL) domain. Reinforcement learning is a type of ML where an agent learns to make decisions by performing actions in an environment to maximize cumulative rewards [19]. Through trial and error, the agent uses feedback from its actions to improve its performance over time (Figure 5).

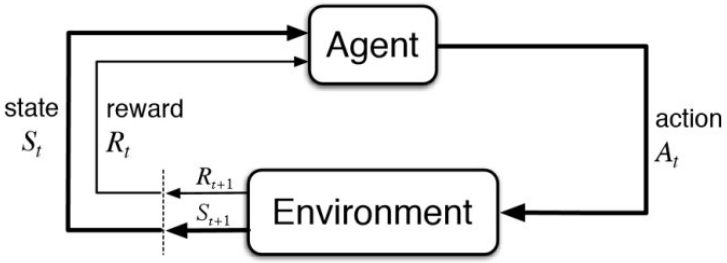


*Figure 5. Reinforcement Learning agent-environment interaction flow*


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

As a result, Multi-Objective Reinforcement Learning (MORL) offers further significant advantages that may be considered as well.

Unlike classical methods, which often require explicit, predefined trade-offs between objectives, MORL can learn and adapt to these trade-offs dynamically through interactions with the environment. This adaptability makes MORL more flexible and robust, particularly in complex and dynamic environments. For instance, MORL can explore the solution space more efficiently, often leading to the discovery of more optimal and diverse set of solutions (Pareto fronts) than traditional methods [12], [20].

Additionally, MORL can handle complex, high-dimensional spaces and stochastic environments more effectively, leveraging the power of Deep Learning (DL) and other ML techniques to generalize well across a wide range of scenarios and adapt to changing conditions without the need for retraining from scratch, which is a limitation in many classical algorithms [21].

Finally, after the training session, MORL agents can provide solutions in near real-time, making them suitable for real-time DSSs.

This makes (Deep)-MORL a powerful tool for solving real-world multi-objective problems where classical algorithms may fall short or may present other disadvantages (e.g. the computational time and resources requirements).

Recently, RL and DRL approaches have been utilized in the nexus domain to face the decision-making problem. [22] proposes a novel computational framework incorporating "algorithmic resilience thinking" to develop adaptive and robust systems, framed as a RL problem using Markov Decision Processes (MDP). In the same study, a case study is presented that focuses on weather volatility and its impact on agriculture, demonstrating the framework's ability to track and mitigate risks, thereby enhancing the resilience and operational effectiveness of integrated resource systems against disruptions and global supply chain stresses. In [23], the decision-making problem within the Food-Energy-Water nexus is modeled as a multi-agent system in a mixed competitive and cooperative environment using a Markov game perspective. The authors propose a fingerprint networked reinforcement learning (FNRL) framework to facilitate collective learning in the multi agent system, integrating a neural network to extract fingerprint information from historical data. Numerical simulations for an urban WEF nexus demonstrate that the FNRL framework effectively guides agents towards optimal decisions in a dynamic environment.

Multi-objective (deep) reinforcement learning, however, has not been extensively explored as a solution for the multi-objective decision-making problem in the nexus, and few studies can be found. An initial approach to the nexus thinking can be found in [24], where objectives such as energy generation, ecological protection and water supply are considered. The paper introduces a Deep Reinforcement Learning (DRL) approach for optimizing multi-objective operations of multiple hydropower reservoirs. They demonstrate significant improvements over


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

existing methods, including an increase in electricity generation and a reduction in annual flow deviation.



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

# 4. Multi-objective RL: problem setting, taxonomy and algorithms

This section first introduces the multi-objective problem setting, extending the single-objective Markov Decision Process (MDP) formalization. Second it presents the taxonomy utilized throughout the rest of the document, and final, it describes the MORL and DMORL algorithms implemented to solve the proposed nexus decision-making problem.

A deep introduction to MDPs and RL can be found in [Sutton]. The following sub-sections are based on [25] and [20].

## 4.1. Multi-objective problem setting

Extending the MDP formalization presented in D4.1 section *6.2.5.1 MDP formalization*, we introduce the Multi-Objective Markov Decision Process (MOMDP) as the proposed mechanism to support decision-making in the nexus.

A MOMDP is represented by the tuple $\langle S, A, T, \gamma, \mu, R \rangle$, where:

- $S$ is the state space
- $A$ is the action space
- $T : S \times A \times S \rightarrow [0,1]$ is a probabilistic transition function
- $\gamma \in [0, 1)$ is a discount factor
- $\mu : S \rightarrow [0,1]$ is a probability distribution over initial states
- $R : S \times A \times S \rightarrow \mathbb{R}^d$ is a vector-valued reward function, specifying the immediate reward for each of the considered $d \geq 2$ objectives

The key distinction between a single-objective MDP and MOMDP lies in the vector-valued reward function. In a MOMDP, this function provides a numerical feedback signal for each

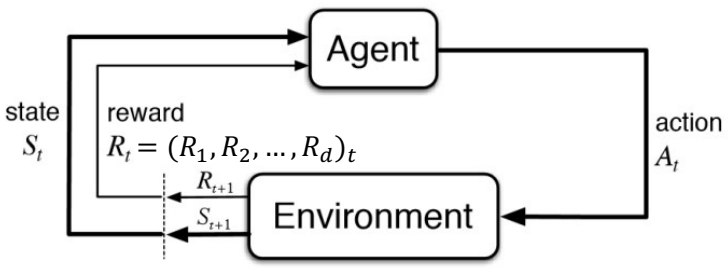


*Figure 6. Multi-Objective Reinforcement Learning agent-environment interaction flow*


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

objective under consideration, resulting in a reward vector whose length matches the number of objectives (Figure 6).
As a result, the value function of a state $s$ (i.e. the expected return G(s) when starting in $s$ and following $\pi$ thereafter) is also vector-valued, $V^{\pi}(s) \in \mathbb{R}^d$.

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | \pi, s_t = s\right]$$

In single-objective settings, the value functions provide a complete ordering over the policy space. This means that for any two policies, $\pi$ and $\pi'$, $V^{\pi}(s)$ will either be greater than, equal to, or less than $V^{\pi'}(s)$. Consequently, finding the optimal policy $\pi^*$ is equivalent to maximizing the expected cumulative discounted reward. For a MOMDP this is not necessarily the case.

If we have access to a utility function $u: \mathbb{R}^d \rightarrow \mathbb{R}$ that maps the multi-objective value of a policy to a scalar value $V_u^{\pi} = u(V^{\pi})$, it would provide a total ordering over policies, effectively reducing the MOMDP to a single-objective decision-making problem. However, this approach is not always possible, feasible, or desirable, as discussed in [20]. Consequently, when dealing with multi-objective value functions, we might encounter situations where $V_i^{\pi} > V_i^{\pi'}$ for objective $i$, while $V_j^{\pi} < V_j^{\pi'}$ for objective $j$.

As a consequence, in MOMDPs, value functions only provide a partial ordering over the policy space. Therefore, determining the optimal policy is not possible without additional information on how to consider or prioritize the objectives to order the policies.

## 4.2. Multi-objective taxonomy

This section introduces various concepts from the multi-objective optimization domain that will be utilized later in the document. Building on the previously presented term, the utility function, two extensions are defined. These extensions form the basis for the different types of solution sets that can be derived.

### 4.2.1. A Monotonically increasing utility function

A monotonically increasing utility function, $u$, follows the rule that if a policy enhances one or more of its objectives without diminishing any others, the scalarized value will also increase.

$$(\forall i : V_i^{\pi} \geq V_i^{\pi'}) \wedge (\exists i : V_i^{\pi} > V_i^{\pi'}) \Rightarrow u(V^{\pi}) \geq u(V^{\pi'})$$

A monotonically increasing utility function can represent both linear (with nonzero positive weights) and non-linear user preferences. Monotonicity in the utility function is a minimal


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

assumption for MORL, as it aligns with the fundamental definition of an objective: we always seek to maximize value in any of the objectives.

### 4.2.2. A linear utility function

A linear utility function calculates the inner product of a weight vector $w$ and a value vector $V^{\pi}$.

$$u(V^{\pi}) = w^{T}V^{\pi}$$

Each element of the weight vector $w$ specifies how much one unit of value for the corresponding objective contributes to the scalarized value. The elements of the weight vector are all positive real numbers and are constrained to sum to 1.

### 4.2.3. Solution sets

In single-objective RL problems, there exists a unique optimal value $V^{*}$, and there can be multiple optimal policies $\pi^{*}$ that all achieve this value. The goal in single-objective RL is typically to learn an optimal policy. However, in MOO cases, without any additional information about the user's utility, there can be multiple potentially optimal value vectors $V$. Therefore, it is necessary to consider different sets of potentially optimal value vectors and policies.

The selection of the solution set is crucial for the efficiency of algorithms used in solving MO problems, as it requires computing all the policies within these sets.

#### 4.2.3.1. The undominated set

The undominated set $U(\Pi)$ consists of the subset of all possible policies $\Pi$ and their associated value vectors for which there exists a potential utility function $u$ that achieves the maximum scalarized value.

$$U(\Pi) = \{\pi \in \Pi \mid \exists u, \forall \pi' \in \Pi : u(V^{\pi}) \geq u(V^{\pi'})\}$$

However, the undominated set might include redundant policies. These are policies that are optimal for a particular utility function, but there are other policies that are also optimal for the same utility function. In such cases, it is unnecessary to keep all these policies to maintain optimal utility.

#### 4.2.3.2. The coverage set


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

A set $CS(\Pi)$ is considered a coverage set if it meets two conditions: it must be a subset of $U(\Pi)$, and, for every $u$ in, it includes a policy with the highest scalarized value.

$$CS(\Pi) \subseteq U(\Pi) \ \wedge \ (\forall u, \exists \pi \in \ CS(\Pi), \forall \pi' \in \Pi : \ u(V^{\pi}) \geq u(V^{\pi'}))$$

#### 4.2.3.3. The Pareto Front

In those cases where the utility function $u$ is any monotonically increasing function, then the Pareto Front (PF) is the undominated set $U(\Pi)$.

$$PF(\Pi) = \{\pi \in \Pi \mid \nexists \pi' \in \Pi : \ V^{\pi'} \succ_p V^{\pi}\}$$

where $\succ_p$ is the Pareto dominance relation

$$V^{\pi} \succ_p V^{\pi'} \Leftrightarrow (\forall i : \ V_i^{\pi} \geq V_i^{\pi'}) \wedge (\exists i : \ V_i^{\pi} > V_i^{\pi'})$$

In other words, the Pareto front is constituted by all the policies such that there is no other policy with value that is equal of better in all the objectives. A collection of policies whose value functions align with the PF is known as a Pareto Coverage Set (PCS).

#### 4.2.3.4. The convex hull

The convex hull (CH) consists of the subset of Π for which there exists a weight vector $w$ (for a linear utility function) such that the linearly scalarized value is maximized. It is the undominated set for linear utility functions.

$$CH(\Pi) = \{\pi \in \Pi \mid \exists w, \forall \pi' \in \Pi: w^T V^{\pi} \geq w^T V^{\pi'}\}$$

#### 4.2.3.5. The convex coverage set

A set CCS(Π) is a convex coverage set if it is a subset of CH(Π) and if, for every weight vector, it includes a policy whose linearly scalarized value is maximized.

$$CCS(\Pi) \subseteq \mathrm{CH}(\Pi) \wedge (\forall \mathrm{w}, \exists \pi \in \mathrm{CCS}(\Pi), \forall \pi' \in \Pi: \mathrm{w}^{\mathrm{T}} \mathrm{V}^{\pi} \geq \mathrm{w}^{\mathrm{T}} V^{\pi'})$$

## 4.3. Multi-objective Reinforcement learning algorithms


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

Multi-objective reinforcement learning encompasses a diverse array of methodologies, from bandit problems to deep reinforcement learning architectures. For a thorough review see [12]. A prevalent and widely adopted approach in MORL is extending established single-objective model-free value-based methods like Q-learning [26] to handle multiple objectives simultaneously.

Some examples of this practice include MO Q-Learning, MPMO Q-Learning [27], Pareto Q-Learning [25] and MPQ-learning [28]. These methods are restricted to tabular representations of Q-values, limiting their applicability to more complex problems. For these types of problems, some DRL algorithms have been adapted to multiple objectives. Most of these methods extend the single objective DQN architecture and some examples are Pareto DQN [29] and Envelope Q-learning [30].

There are also some alternatives to the seen value-based approaches, adopting policy search algorithms. Some examples are the Expected Utility Policy Gradient (EUPG) implementation by Roijers et al. [31], the multi-objective categorical Actor-Critic (MOCAC) by Reymond et al. [32] or the multi-objective extension of PPO by Xu et al [33].

For the Self-Learning Nexus Engine, we have considered the tabular method Pareto Q-learning and the DRL algorithm Envelope Q-learning implementations from MORL-Baselines [34] Python library. Considering the number of goals we have (Table 4), we have focused on the algorithms that do not impose explicit limitations on the number of objectives. Finally, we have adapted the available implementations to match the NXG case and its characteristics.

The usage of Pareto Q-learning will allow us to explicitly capture Pareto-optimal solutions, allowing a thorough modelling of the trade-offs between different objectives. Also, since its tabular nature, it will allow us to have a more variate range of recommendations. On the other hand, Envelope Q-learning focuses on efficiently representing the envelope of the Pareto front, simplifying the computational complexity associated with maintaining multiple solutions. This method scales effectively to larger state-action spaces, being the best option for solving the bigger problems presented in section 2.2.

### 4.3.1. Pareto Q-learning

Pareto Q-learning, or PQL [25], is an on-line Temporal Difference-based [19] multi-objective learning algorithm proposed as an extension of Q-learning to multi-objective problems. The algorithm is made for only episodic problems, and works model-free, learning both stationary and non-stationary deterministic policies.

Similar to Q-learning, PQL is a method where an agent interacts with its environment to learn the optimal expected long-term rewards of its actions (Algorithm 1). For each state-action pair $(s, a)$ sets of vectors, $Q(s, a)$ are learned. Each vector $q \in Q(s, a)$ represents the expected vector reward when following a particular Pareto-optimal policy after taking action $a$ in state


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

$s$. Additionally, the set $V(s_0)$ contains all the Pareto-optimal vector estimates for all actions available in state $s_0$. These sets are the multi-objective equivalents of the scalar $Q$ and $V$ sets used in traditional Q-Learning.

**Algorithm** Pareto $Q$-learning algorithm

Initialize $\hat{Q}_{set}(s,a)$'s as empty sets
**for** each episode $t$ **do**
  Initialize state $s$
  **repeat**
    Choose action $a$ from $s$ using a policy derived from the $\hat{Q}_{set}$'s
    Take action $a$ and observe state $s' \in S$ and reward vector $\mathbf{r} \in \mathbb{R}^m$

    $ND_t(s,a) \leftarrow ND(\cup_{a'} \hat{Q}_{set}(s',a'))$ ▷ Update ND policies of $s'$ in $s$
    $\overline{\mathfrak{R}}(s,a) \leftarrow \overline{\mathfrak{R}}(s,a) + \frac{\mathbf{r} - \overline{\mathfrak{R}}(s,a)}{n(s,a)}$ ▷ Update average immediate rewards
    $s \leftarrow s'$ ▷ Proceed to next state
  **until** $s$ is terminal
**end for**

*Algorithm 1. Pareto Q-learning algorithm. Source [25]*

During training, PQL employs an $\epsilon$-greedy exploration strategy to balance between exploration and exploitation phases. For the exploitation, the algorithm presents three mechanisms that allow action selection based on the content of the sets $\hat{Q}_{set}(s,a)$:

- Hypervolume: the volume of the objective space covered by the $\hat{Q}_{set}$ of each action. It is the only quality indicator to be strictly monotonic with the Pareto dominance relation.
- Cardinality: number of Pareto dominating vectors in the $\hat{Q}_{set}$ of each action. It is a heuristic that can guide the search process towards actions that dominate the others locally within a state.
- Pareto set evaluation: metric similar to the cardinality. Instead of computing the number of non-dominated elements in the $\hat{Q}_{set}$ of each action, it randomly samples from all the actions that have a non-dominated vector across every other action $a'$.

Once the algorithm is trained, in traditional Q-learning, the learned policy can be easily tracked by using the argmax operator over all actions. However, in the case of PQL, many optimal policies are available in each state, so there is the possibility of tracking any of the vectors in $\hat{Q}_{set}(s,a)$ for all actions given a state $s$ . The process works as follows: given a target vector $q$ to follow, for each action $a \in A$, we retrieve both the averaged immediate reward $\overline{\mathfrak{R}}(s,a)$ and $ND_t(s,a)$, which is discounted. With both values we compute the $\hat{Q}_{set}(s,a)$. If any of the vectors in $\hat{Q}_{set}(s,a)$ is equal to $q$, we select the corresponding action and step into the next state. The target vector $q$ is then set to the corresponding vector of $ND_t(s,a)$ and the process continues until a terminal state is reached.

PQL is an algorithm that effectively balances multiple objectives by maintaining $\hat{Q}_{set}$'s for each state-action pair, approximating the Pareto front for optimal policy discovery. However, many challenges arise when using this algorithm, including an increased computational complexity


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

due to managing multiple $\hat{Q}_{set}$'s and difficulties in scaling to large state and action spaces and number of objectives.

### 4.3.2. Envelope Q-learning

Contrary to PQL, Envelope Q-learning [30] doesn't compute a set of optimal policies that encompass the entire space of possible preferences in the domain. Instead, it learns a single policy network that is optimized over the entire space of preferences in a domain. In this way, just one network can produce the optimal policy for any user-specified preference.

This algorithm assumes that the utility function will be linear, i.e., that the user-preferences will be modelled as weighted averages, where the user sets the weight vector. Given this assumption, the set of optimal policies learnt by the algorithm are the ones from the convex coverage set (CCS) of the problem.

To obtain the CCS solutions, the algorithm uses double Q-learning with target Q-networks following [35], [36]. The key differences from standard double Q-learning are in the target update mechanism and the loss function. In Envelope Q-learning, the target update involves maintaining an envelope around the Q-values to constrain possible updates, ensuring they remain within a plausible range. This helps mitigate overestimation and stabilizes learning. Additionally, the loss function in Envelope Q-learning incorporates a penalty for predictions that fall outside the envelope, further guiding the Q-values to stay within the predefined bounds. This combination enhances the robustness of the learning process and improves convergence to optimal policies.

Once a policy model is obtained in the learning phase (Algorithm 2), the agent can adapt to any provided preference by simply feeding a weights vector $\omega$ (a linear utility function) into the trained network. This weights vector $\omega$ adjusts the importance of the different objectives according to the user's preferences. Without a need for retraining, the agent can quickly adapt to changing conditions or preferences in near real-time, ensuring an optimal policy across a variety of scenarios.

In many evaluations, Envelope Q-learning proves to be an algorithm that can scale to high dimensional state and action spaces, and work with many objectives. Also, it is sample-efficient and seems capable of adapting to different preferences effectively, showing effective generalization and policy adaptation. The added complexity of maintaining and updating the envelope can increase computational overhead and may require careful tuning of hyperparameters to balance the size and adaptation of the envelope.


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

**Algorithm** Envelope MOQ-Learning

**Input:** a preference sampling distribution $\mathcal{D}_\omega$, path $p_\lambda$ for the balance weight $\lambda$ increasing from 0 to 1.

Initialize replay buffer $\mathcal{D}_\tau$, network $\boldsymbol{Q}_\theta$, and $\lambda = 0$.

**for** *episode* = $1, \ldots, M$ **do**

  Sample a linear preference $\boldsymbol{\omega} \sim \mathcal{D}_\omega$.

  **for** $t = 0, \ldots, N$ **do**

    Observe state $s_t$.

    Sample an action $\epsilon$-greedily:

$$a_t = \begin{cases} \text{random action in } \mathcal{A}, & \text{w.p. } \epsilon; \\ \max_{a \in \mathcal{A}} \boldsymbol{\omega}^\intercal \boldsymbol{Q}(s_t, a, \boldsymbol{\omega}; \theta), & \text{w.p } 1 - \epsilon. \end{cases}$$

    Receive a vectorized reward $\boldsymbol{r}_t$ and observe $s_{t+1}$.

    Store transition $(s_t, a_t, \boldsymbol{r}_t, s_{t+1})$ in $\mathcal{D}_\tau$.

    **if** *update* **then**

      Sample $N_\tau$ transitions $(s_j, a_j, \boldsymbol{r}_j, s_{j+1}) \sim \mathcal{D}_\tau$.

      Sample $N_\omega$ preferences $W = \{\boldsymbol{\omega}_i \sim \mathcal{D}_\omega\}$.

      Compute $y_{ij} = (\mathcal{T}\boldsymbol{Q})_{ij} =$

$$\begin{cases} \boldsymbol{r}_j, & \text{for terminal } s_{j+1}; \\ \boldsymbol{r}_j + \gamma \arg_Q \max\limits_{\substack{a \in \mathcal{A}, \\ \boldsymbol{\omega}' \in W}} \boldsymbol{\omega}_i^\intercal \boldsymbol{Q}(s_{j+1}, a, \boldsymbol{\omega}'; \theta), & \text{o.w.} \end{cases}$$

      for all $1 \leq i \leq N_\omega$ and $1 \leq j \leq N_\tau$.

      Update $Q_\theta$ by descending its stochastic gradient according to equations 6 and 7:

$$\nabla_\theta L(\theta) = (1-\lambda) \cdot \nabla_\theta L^{\mathrm{A}}(\theta) + \lambda \cdot \nabla_\theta L^{\mathrm{B}}(\theta).$$

      Increase $\lambda$ along the path $p_\lambda$.

*Algorithm 2. Envelope Q-learning algorithm. Source [30]*



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

# 5. The Nexogenesis decision-making problem formalization

[12] and [20] propose six scenarios where a multi-objective approach is required. The NXG decision-making case falls into the decision support scenario, where the user's preferences are unknown or difficult to specify.

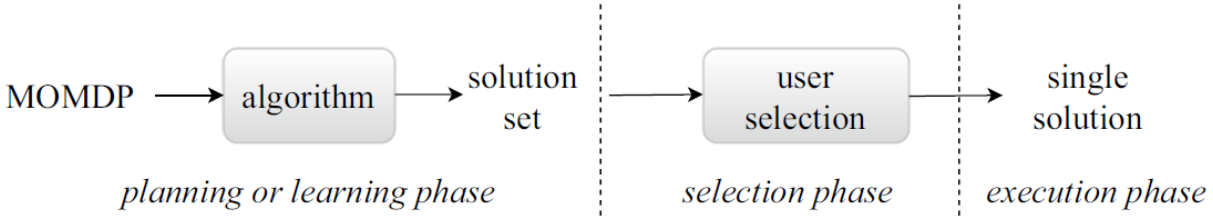


*Figure 7. Decision support scenario diagram, extracted from [X]*

As seen in Figure 7, in this scenario, since a priori we don't know the user's preferences, we compute a solution set with the Pareto front solutions to be able to respond with an optimal solution regardless of the preferences. Once the user wants a recommendation, he provides its preferences and with those set, the selection phase is run, obtaining the best solution according to the set preferences.

The MOMDP in the NXG decision-making problem is presented as an episodic case where, given an initial situation, the agent must provide the undominated set solutions (further details provided in section 5.1). Two problem variants are proposed, one considering deterministic environments (i.e. deterministic SDMs), and the other considering stochastic environments. In the first case, the average reference scenario is used. In the second case, available data stochasticity in SDMs is not aggregated, thus a more challenging situation is presented. Further details on how these environments are constructed can be found in D4.3.

Each project CS represents a unique optimization problem. Additionally, there are three further layers of complexity. First, each CS includes a set of reference scenarios (RCP-SSP combinations), each depicting different potential future conditions. Second, the complexity models implemented by WP3 incorporate randomness in the input data, allowing for both deterministic and stochastic execution modes. Finally, one of the CSs (Inkomati) introduces an additional decision-making dimension: the year when a policy is applied, which we call the 'dynamic policies' mode, in contrast to the 'static mode' with fixed policies. All these options are available through different implementations of the CS' System Dynamic Models (SDMs). Consequently, an agent will be trained for every combination of CS, reference scenario, randomness execution mode, and, for the Inkomati CS, the policy mode.

The NXG decision-making problem formalization is described below. We start discussing the details of the proposed utility functions space, and later discuss about the deterministic and stochastic cases.


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

## 5.1. Utility functions set by the stakeholder

As introduced in previous sections, the key of MORL is allowing the users to set their preferences and return the optimal solutions according to their settings. To do that, we studied an adequate utility function $u$ that fits the SHs requirements while simplifies enough the solution set we have to deal with.

$$u: \mathbb{R}^d \longrightarrow \mathbb{R}$$

In the proposed nexus decision-making scenario, we consider that all the plausible functions in the utility function set are monotonically increasing, meaning that if a policy increases for one or more of its objectives without decreasing any of the objectives, the scalarised value also increases. Furthermore, available utility functions are also linear, thus the weighted sum for each value of the objective is computed.

Therefore, in our solution, the user will be able to set a weight for each objective, which will be used to compute a scalar value and filter the solutions of the pareto front to get only the ones that are the best according to the user's preferences.

Let's consider an example. Suppose we have three objectives:

- O1 - Water Availability: Ensuring a sustainable water supply, measured in million cubic meters per year.
- O2 - Energy Efficiency: Optimizing energy consumption, measured in gigawatt-hours (GWh) saved per year.
- O3 - Food Production: Maximizing food output, measured in million tons per year.

The utility function is a weighted sum of the three objectives, with weights assigned based on their relative importance, selected by the stakeholders. For instance, suppose the stakeholders assign the following weights:

- $w_1 = 0.5$ Water Availability
- $w_2 = 0.3$ Energy Efficiency
- $w_3 = 0.2$ Food Production

The utility function $u(p)$ for a policy $p$ can be expressed as:

$$u(p) = 0.5 \times O_1(p) + 0.3 \times O_2(p) + 0.2 \times O_3(p)$$

Now, consider three different policies $p_1$, $p_2$ and $p_3$ with the following impacts and objective values:

- Policy $p_1$:
  - Water Availability $O_1(p_1)$ = 120 million cubic meters/year


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

- Energy Efficiency $O_2(p_1)$ = 200 GWh/year
- Food Production $O_3(p_1)$ = 50 million tons/year

- Policy $p_2$:
  - Water Availability $O_1(p_2)$ = 100 million cubic meters/year
  - Energy Efficiency $O_2(p_2)$ = 250 GWh/year
  - Food Production $O_3(p_2)$ = 60 million tons/year
- Policy $p_3$:
  - Water Availability $O_1(p_3)$ = 110 million cubic meters/year
  - Energy Efficiency $O_2(p_3)$ = 180 GWh/year
  - Food Production $O_3(p_3)$ = 55 million tons/year

Using the utility function, we can calculate the utility for each policy.

- Policy $p_1$:

$$u(p_1) = 0.5 \times 120 + 0.3 \times 200 + 0.2 \times 50 = 130$$

- Policy $p_2$:

$$u(p_2) = 0.5 \times 100 + 0.3 \times 250 + 0.2 \times 30 = 137$$

- Policy $p_3$:

$$u(p_1) = 0.5 \times 110 + 0.3 \times 180 + 0.2 \times 55 = 120$$

Comparing the utility values of the policies, we find that $u(p_2) = 137$ is the highest. Therefore, policy $p_2$ is the preferred policy according to the given utility function, as it maximizes the overall utility by effectively balancing water availability, energy efficiency, and food production.

## 5.2. The deterministic case

In this section, the decisions made for the design of the MOMDP in the deterministic case are described.

### 5.2.1. The state space

In a RL environment based on a fully observable MDP, states are typically defined to capture all relevant information about the current situation the agent is in. Based on the Markov property [Sutton], this information should be sufficient to make decisions about the next action.

We model the state space $S$ as an N-binary vector that indicates to the agent which actions have been selected; in simpler terms, it denotes the current policy package.

Given that the agent learns from the average scenario of the SDM, the results of the application of a policy package are deterministic, hence the transition function $T(s, a, s')$ in our problem is deterministic $(T(s, a, s') = 1)$. Therefore, it is unnecessary to include nexus variables in the


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

observation. The current policy package alone encapsulates all essential information about the agent's current context, and is sufficient to distinguish all possible states, since the initial state $s_0$ is always the same (i.e. a deterministic SDM execution with no policies applied).

Depending on the CS to be solved, its dimensions N change. In a CS with fixed policies, the state $s$ will be a binary vector with one component for each implemented policy (see Figure 8-a). As for case studies with dynamic policies, the state s is defined as a binary matrix with as many rows as applied policies, and one column per each year of application (Figure 8-b).

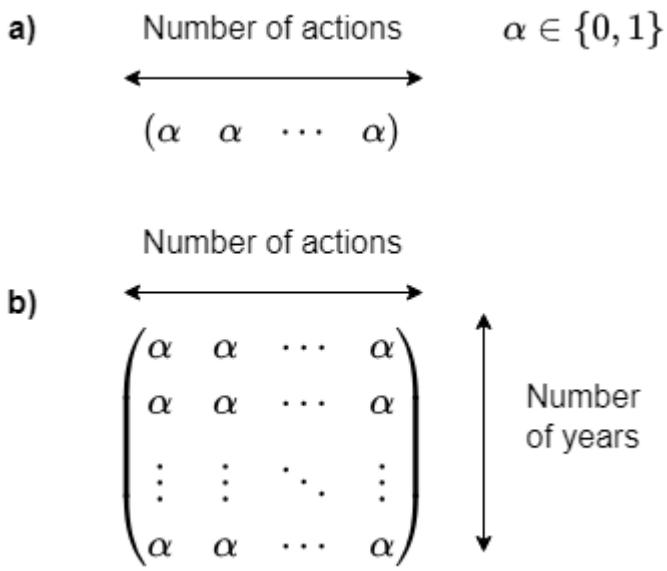


*Figure 8. Nexogenesis state space*

## 5.2.2. The action space

Regarding the action space $A$, the agent's actions correspond to applying one of the *implemented* policies (Table 1). When we refer to *implemented* policies, we mean the transformed final versions designed for the self-learning nexus engine. Additionally, an extra action $a$ is considered, which represents not applying any policy. This is for the agent to be able to choose to stop applying policies and the keep the obtained policy package, thus finishing an episode.

Similarly to states, the number of actions varies depending on the "type of policies" being considered. In the static case, an action corresponds to applying a policy, while in the dynamic case the actions correspond to applying a policy in a specific year. In this latter case, the number of actions significantly increases, totalling the number of implemented policies multiplied by the number of years over which they are applied.


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

### 5.2.3. The reward function

The reward function *R* is crucial in RL as it guides the agent's behavior by providing feedback on the desirability of its actions. In this context, the objective is for the agent to identify optimal policy packages. However, defining this optimality has proven complex, with several considerations explored:

- Achieving solutions that meet specified goal targets.
- Attaining goal targets while minimizing the number of policies in the package.
- Meeting goal targets while optimizing a specific sector.
- Maximizing goal targets across all sectors.
- Maximizing other nexus indicators.

Since there are numerous possible definitions for an 'optimal policy package,' we asked SHs what type of recommendations they would prefer to receive. The most popular response was the smallest policy package that achieves the goal targets. This seems a reasonable choice, as smaller policy packages are easier to execute and implement. However, incorporating additional constraints, such as economic policy costs or social impact, could provide further clarity on this issue.

With this definition, we have designed the reward to be a vector with one component for each CS goal, and a last component indicating the number of policies applied in the policy package. Specifically, the rewards corresponding to the CS goals are defined as the distance to achieve the goal. For instance, if a goal requires a 20% increase in a nexus indicator compared to the baseline, the initial distance to the target would be 20. This distance is computed from the goal starting month to the end of the simulation in a monthly basis. Following this example, if the initial year for evaluating the goal is January 2015, knowing that all CSs simulations run from January 2015 to December 2049, the distance should be computed for the 420 months of the simulation (35 years * 12 months). Hence, the total difference would be -20 * 420 = -8,400. To signify that we are 8,400 units away from achieving the target, this reward is expressed as a negative value.

These distances can range from 0, indicating goal achievement, to any negative values, influenced by factors such as the goal's starting date and the target percentage. All the goals should be equally considered by the agent, so there can't be different value ranges across the components of the reward. To resolve this issue and avoid having the agent prioritize the completion one goal over another, we scale these total distances between -100 and 0 using a MinMax scaler[4], where -100 corresponds to the baseline distance and 0 to achieving the goal. The last component of the reward needs no scaling, as it is just a negative number indicating

[4] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html#sklearn.preprocessing.MinMaxScaler


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

the number of policies applied in the current policy package. This value is negative, so as to favour smaller policy packages over bigger ones.

To help the agent converge faster, and easily identify the individual effects of his actions, the reward function has been defined as a dense function, providing feedback at every step of the agent's actions. This reward function returns a reward vector with the increment or decrement in distance/level of achievement with respect to the previous step in an episode. The reward component indicating the number of policies is also incremental, having a -1 in each step. For instance, given that the baseline distance is -100 by the number of goals, if in the first step we apply a policy that improves one goal by 10%, we would get a reward vector with a 10 in the achievement of that affected vector, and the rest of goal distance values would be zeroes since no change has been made.

With this incremental reward definition, the total reward will always have a goal achievement level between a vector of zeros, indicating no achievements beyond the baseline, and a vector of one hundred by the number of goals, indicating full achievement of all goals in the policy package. Depending on the behavior of the SDM, there is also the possibility of having a negative value in the episode total reward, signaling maladaptation where the achieved state is worse than the baseline.

For a comprehensive global nexus analysis, the proposed goals for some of the CS may be too sector or policy-specific. In such cases, we study incorporating nexus footprint indicators as additional objectives. This approach, including these indicators in the reward function, provides a more holistic perspective, allowing us to consider all trade-offs within the nexus.

### 5.2.4. Nexus Optimal Solutions

Given the explained problem definition, the self-learning nexus engine will learn through a repetitive interaction with the environment, the set of Convex Hull solutions. In this case, this corresponds to the set of solutions/strategies that yield a value vector where each component of the final reward is at least as good as all other possible solutions, with at least one component of the vector being strictly superior to the others.

To clarify this concept and understand what type of solutions the algorithms used will consider, let's illustrate it with an example. Imagine a toy case study where we have three policies and two CS goals. Given the state space represented by Figure 9, we can see the different states and the incremental rewards obtained in each step, given by the transition from one state to another. Although not represented in the diagram for clarity, note that in every state there is the possibility of choosing the action of not applying any policy, which would return a reward of [0 0 0] and end the episode.


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

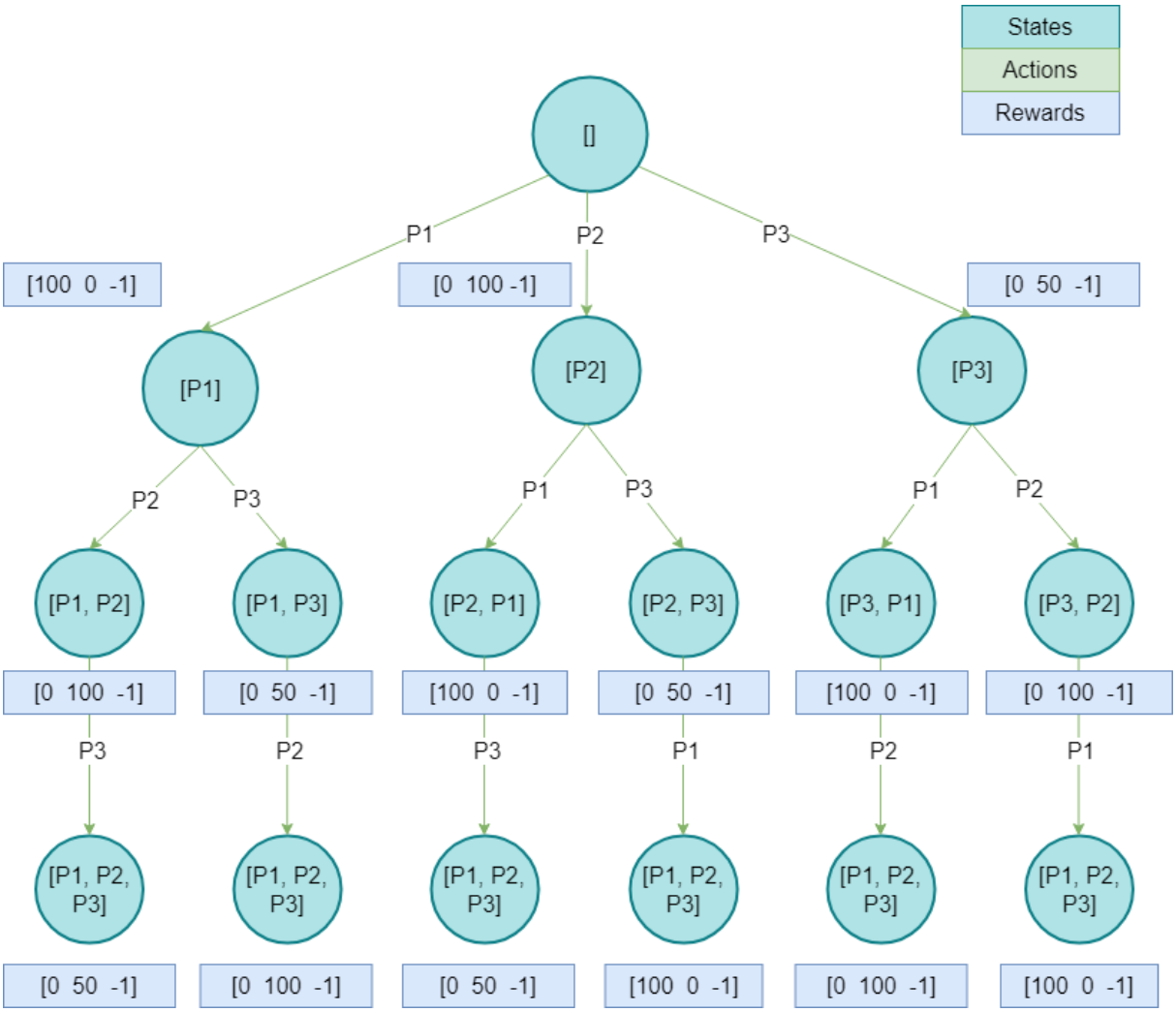


*Figure 9. Multi-objetive nexus problem example*

As already mentioned, the optimal set for each state will include vectors resulting from the implementation of optimal strategies. These vectors will be such that there is no other strategy yielding a vector that is strictly superior in all components. For example, in the state defined by "[P1, P2]", the optimal set includes two vectors: one from applying no policy [0, 0, 0] and another from applying policy P3, resulting in [100, 150, -3].

From the initial state "[]," most of the possible policy combinations would be considered optimal strategies as listed in Table X. However, certain strategies are discarded. For example, [P3] with returns [0 50 -1] is dominated by [P2] with returns [0 100 -1]. Similarly, [P1, P3] with returns [100 50 -2] is dominated by [P1, P2] with returns [100 100 -2].

*Table 5. Multi-objective nexus problem example Convex Hull*

| **Strategy** | **Value vector** | | |
|---|---|---|---|
| | G1 | G2 | PP size |
| **[]** | 0 | 0 | 0 |
| **[P1]** | 100 | 0 | -1 |


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

| | | | |
|---|---|---|---|
| **[P2]** | 0 | 100 | -1 |
| **[P1, P2]** | 100 | 100 | -2 |
| **[P2, P3]** | 0 | 100 | -2 |
| **[P1, P2, P3]** | 100 | 150 | -3 |

## 5.3. Stochastic case

During the previous problem definition phase, only deterministic SDMs were considered, and no implementation has been made so far to include stochastic elements. In this section, we discuss the modifications needed to accommodate uncertain results in our problem.

Theoretically, if no changes were made, it would result in dealing with a Partially Observable Multi-Objective Markov Decision Process (POMOMDP). Since in the stochastic scenario there are some nexus variables that take random values, having only the state defined in section 5.2.1, would not provide enough information and the agent wouldn't be able to distinguish different situations, or even starting states.

Given the potential complexity added by incorporating stochastic elements, another approach could involve integrating all pertinent random variables into the state $s$. This would enable the agent to precisely assess its current state. A simplified approach could involve incorporating only key nexus variables, such as the nexus footprint indicators, to represent the nexus state. In this case, the added complexity in terms of state definition would be minimized.

However, due to the inherent randomness in SDMs, where variables follow a uniform distribution, additional analysis is necessary to determine how optimizing the policy package would function in such volatile scenarios.

We are continuing to work on this point and it will be reported in the next version of the SLNAE/NEPAT tool..


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

# 6. Initial results

Given the problems presented in section 2.2, we applied the proposed mechanisms to the Lielupe, Jiu and Inkomati CSs with fixed policies using the RCP26-SSP2 scenario and deterministic SDMs. Additionally, we also ran some initial processes for the Inkomati CS with dynamic policies every 5 years. We will continue adjusting the framework, training agents and validating their performance over the coming months.

For the small, fixed policies problems, a three-step procedure was employed to obtain the Convex hull solutions for each and ensure effective learning.

Firstly, thanks to the feasible size of the state space S, a highly parallelized exhaustive search approach is run to obtain the episode rewards for each of the valid policy combinations in the CS. This is used for validation. With these rewards, we employ a function to filter only the dominant vector rewards. Since all the valid combinations were considered in the filtering, the result is the Convex hull set from the initial state, which corresponds to the baseline or reference scenario, i.e., the case where no policies are applied. This front comprehends all the possible optimal solutions, since all the states in the problem are reachable from the initial state.

After the Convex hull is known, a Pareto Q-learning agent is trained, which directly learns the Pareto front (i.e. the Convex hull set in or case), and the found solutions are compared with the results obtained from the exhaustive search.

Finally, an agent with the algorithm Envelope Q-learning undergoes training, to see if we can learn the optimal solutions using a more scalable and sample efficient algorithm. In this case, this algorithm doesn't explicitly learn all the solutions from the Convex Hull as explained in section 4.3.2. Instead, it trains a unified policy network that is optimized across all possible preferences within a domain. Therefore, this last approach is evaluated by a set of tests, involving different setup states and defined priorities to ensure that the solutions provided actually correspond to the Convex Hull.

For the larger-scale problems like Lielupe with fixed policies, or the dynamic policies problem solved for Inkomati, where the number of combinations is exceedingly large, it becomes unfeasible to run an exhaustive search. Due to this reason, for these cases only Pareto Q-learning and Envelope were run. The subsequent subsections will detail the results obtained for each of the solved cases.

The training sessions for the Envelope Q-Learning algorithm, which uses Deep Learning as an approximation function, were run on GPU hardware. The remaining algorithms and approaches were run on CPUs.


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

## 6.1. Jiu

In this problem, as seen in Table 2, there is a total of 2,621,440 combinations. By running an exhaustive search, a total of 8,000 valid combinations were found and from all these solutions. From these valid combinations, 5,887 solutions correspond to the Convex hull set for this problem.

Knowing this, we've trained a PQL agent to run one million steps. Although this number of steps is reduced compared to the total number of combinations, the results show that the agent makes an intelligent exploration, obtaining 5761 solutions (98%), which is a number of solutions close to the real Convex Hull, as seen in Figure 10.

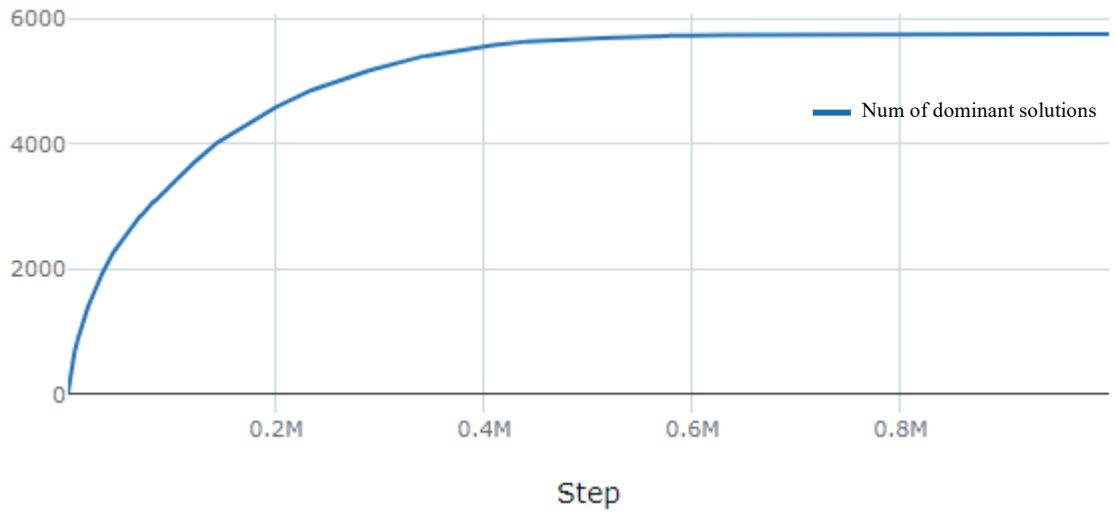


*Figure 10. Number of dominant solutions during PQL training in Jiu CS*

In regard to the Envelope agent, it was also trained during one million steps, and it showed promising metrics during the training, demonstrating a positive trend with consistently decreasing loss Figure 11.

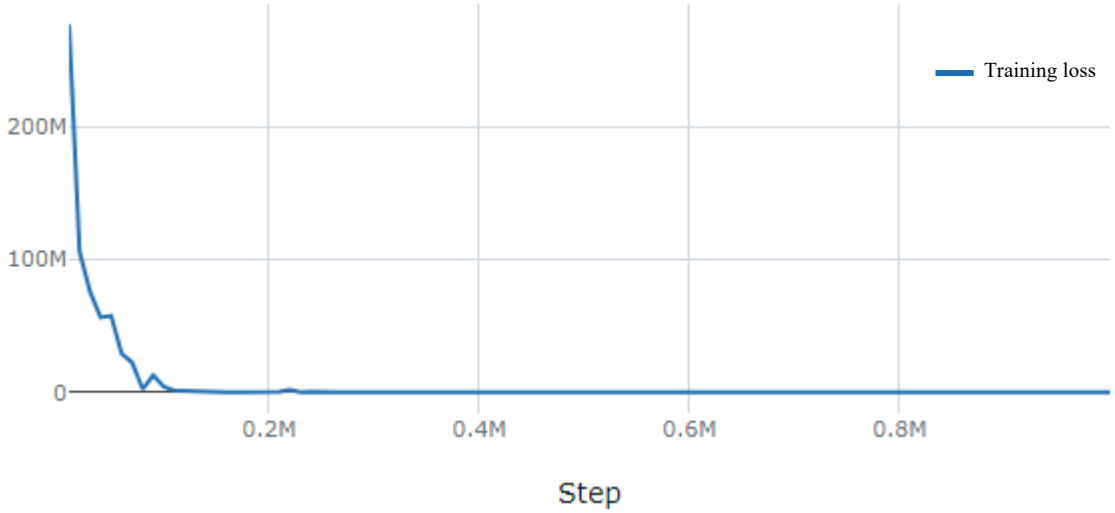


*Figure 11. Envelope network loss during training in Jiu CS*


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

To evaluate its performance, a set of tests were carefully selected, detailed in Annex I. These results present the number of solutions provided by the agent that actually are from the Convex hull set, considering there are 5 tests and each one is repeated three times, one per each of the chosen weights.

*Table 6. Envelope test results in Jiu CS*

| Test | N° of solutions in the Convex hull |
|---|---|
| **T1** | 2/3 |
| **T2** | 1/3 |
| **T3** | 2/3 |
| **T4** | 3/3 |
| **T5** | 3/3 |

As it can be seen in Table 6, considering the number of steps sampled by the agent, most tests yield optimal solutions except for test 2. While the results are quite promising, longer training times may be required for the agent to converge and yield optimal results in all tests.

## 6.2. Inkomati

As mentioned previously, in the case of the Inkomati CS, we've focused on solving the problem for fixed policies, but also trained some agents on the smallest dynamic mode scenario. In the following sections, we review the results obtained so far.

### 6.2.1. Fixed policies

For this scenario, the results of the exhaustive search returned a total of 2,048 valid combinations from the 13,312 possible ones (Table 2). After filtering the non-dominant solutions, a total of 418 solutions constitute the Convex hull set.

A PQL agent was trained for 100,000 steps in this case, giving a good margin for the agent to find all the solutions. As seen in Figure 12, in this case also a close number of the Convex hull solutions were found.


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

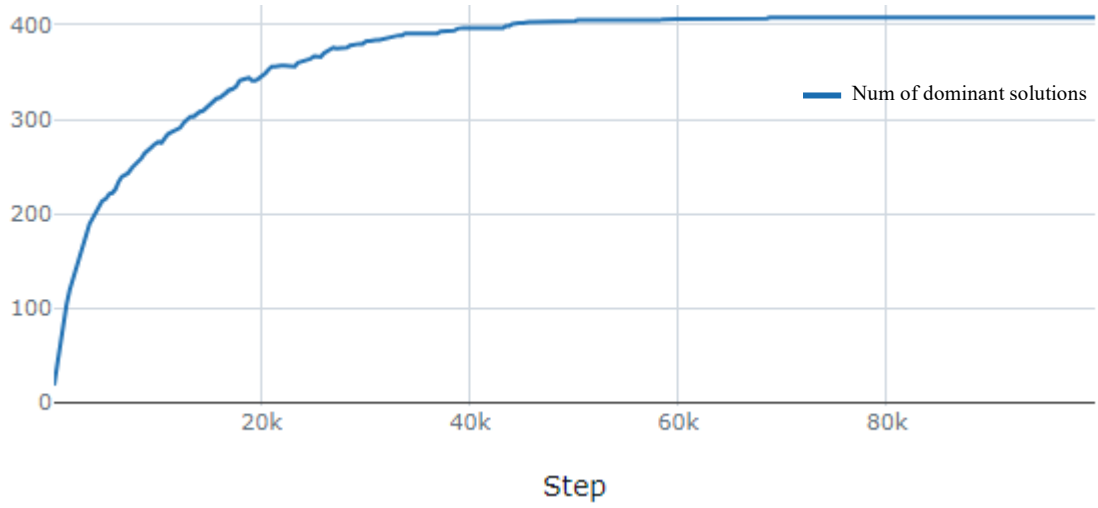


*Figure 12. Number of dominant solutions during PQL training in Inkomati CS*

Upon examination, it was found that the algorithm had learnt most of the solutions and was missing 14 vectors in the obtained set of solutions, meaning a 96% of the solutions were found. Looking at the tendency of the line representing the number of dominant solutions found by the agent (Figure 12), it seems that the size was still growing and it still needed more steps for it to converge.

In the case of the Envelope agent, it was trained for 50,000 steps, demonstrating a promising trend with a steadily decreasing loss (Figure 13).

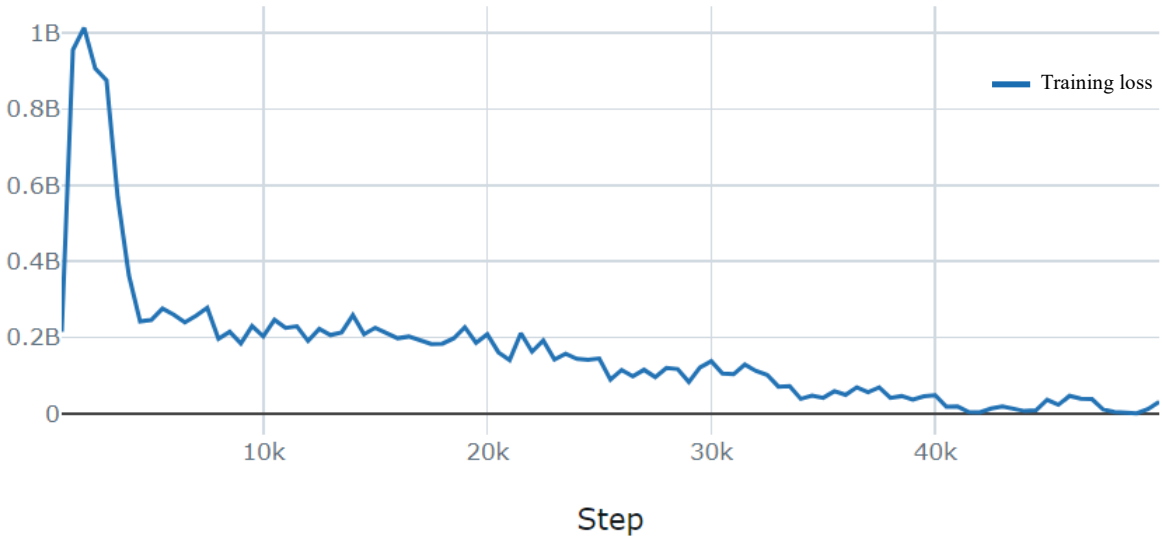


*Figure 13. Envelope network loss during training in Inkomati CS with fixed policies*

The results of this training can be seen in Table 7, where we can see the agent's performance across the different tests detailed in the Annex I.

*Table 7. Envelope test results in Inkomati CS with fixed policies*

| Test | N° of solutions in the Convex Hull |
|---|---|
| **T1** | 3/3 |


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

| | |
|---|---|
| **T2** | 0/3 |
| **T3** | 0/3 |
| **T4** | 0/3 |
| **T5** | 0/3 |

In this case, except for test 1, the results seem to be all suboptimal, indicating that the agent still needed much more training and may require a revision on the training parameters and exploration strategies. We are continuing to work on this point.

### 6.2.2. Dynamic policies

From all the scenarios introduced in Table 3, some trials were made in the scenario with dynamic policies with 7 application years (every 5 years). Given its size, only Envelope could be applied to solve this problem since both exhaustive search and Pareto Q-learning can't scale for such numbers of possible states.

After running many trials with different hyperparameters, the best results were found in a training of 500,000 steps yielding the best results in the evaluations along with presenting a decaying loss (Figure 14).

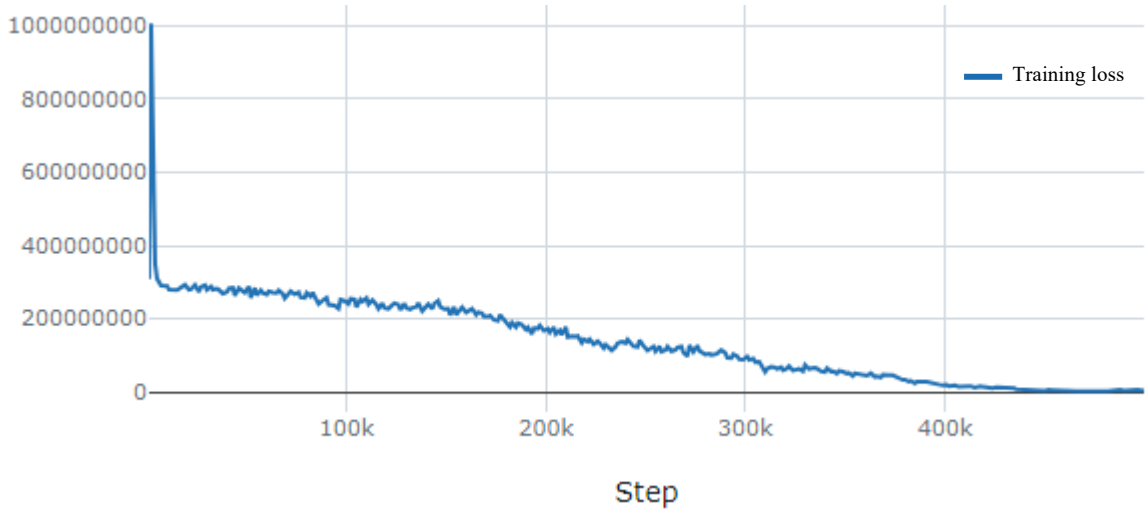


*Figure 14.Envelope network loss during training in Inkomati CS with dynamic policies every 5 years*

Given the lack of results from both the exhaustive search and the Pareto Q-learning algorithm due to the problem's size, we need to consider alternative approaches for tackling and evaluating the problem. Moving forward, we may employ multi objective genetic algorithms as a comparative benchmark to assess our agent's performance. This strategy will allow us to explore different solution spaces and potentially achieve more efficient and scalable outcomes.

## 6.3. Lielupe


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

In the Lielupe CS, there's a total of 218,103,808 possible combinations as seen in Table 2. Like with the case of Inkomati with dynamic policies, given the scale of this problem it was not possible running an exhaustive search and obtaining the Convex Hull.

It was possible however executing both the Pareto Q-learning and Envelope Q-learning algorithms. The PQL agent was trained for 1.5 million steps and showed signs of convergence, as evidenced by the stability in the number of dominant solutions found by the algorithm on the last steps in Figure 15.

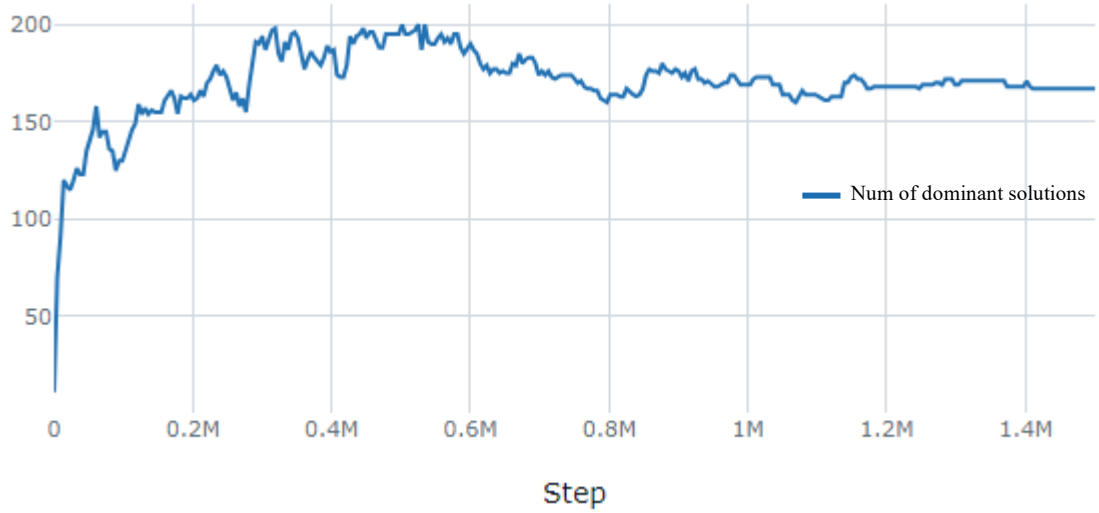


*Figure 15. Number of dominant solutions during PQL training in Lielupe CS*

As for the Envelope agent, this was also trained for about 1.5 million steps and showed good metrics, with a loss that decreases steadily (Figure 16).

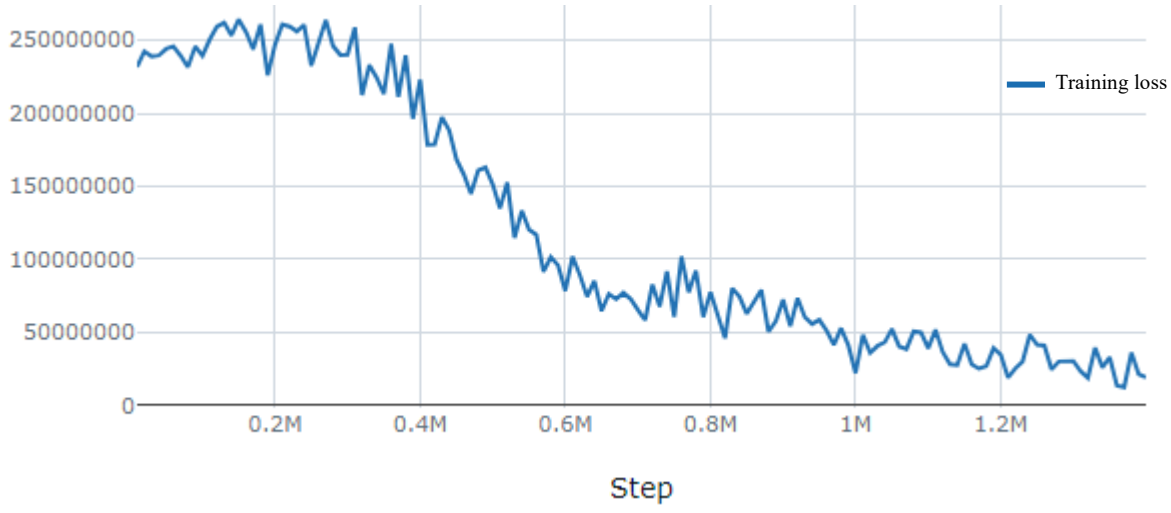


*Figure 16. Envelope network loss during training in Lielupe CS*

Although we don't have the Convex Hull because of the problem size, we can evaluate the envelope solutions considering the front given by the execution of Pareto Q-learning. These will provide us with a comparison of the two algorithms and allow us to evaluate the Envelope Q-learning solutions like in the other cases.


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

*Table 8. Envelope test results in Lielupe CS with fixed policies*

| Test | Nº of solutions in the Convex Hull |
|---|---|
| **T1** | 0/3 |
| **T2** | 0/3 |
| **T3** | 0/3 |
| **T4** | 0/3 |
| **T5** | 0/3 |

Although the agent's learning appears to converge and suboptimal solutions were obtained, none of these solutions match the convex hull set derived from Pareto Q-learning. Further research on this issue will be conducted, and different hyperparameters will be reviewed. Additionally, agents will be trained for further iterations, as the current number of iterations seems insufficient given the total number of possible states.


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

# 7. The NEPAT DSS

The trained agents discussed in the previous section have been integrated into NEPAT to validate the recommendation pipeline. A specific view in the NEPAT UI, the Decision Support System view (Figure 17), has been deployed, allowing users to interact with the recommendation service.

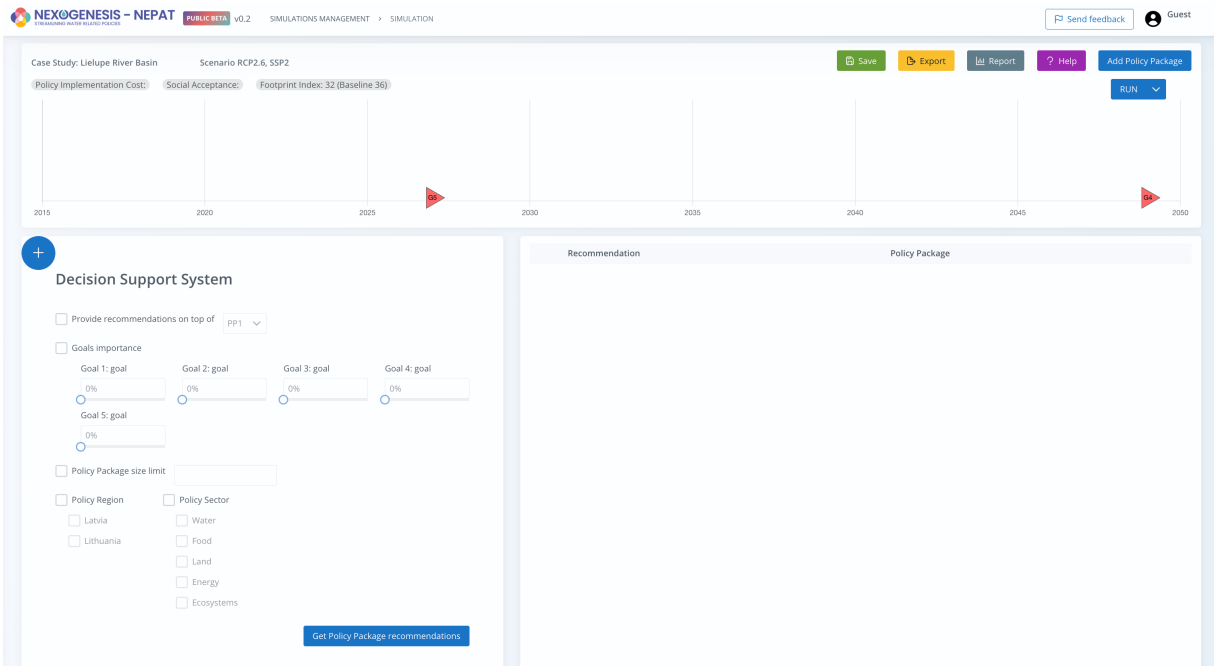


*Figure 17. NEPAT UI. Decision Support System view*

The view is divided into two sections. On the left side (Figure 18), the user can determine their preferences to build a customized utility function. Various functionalities are provided in this section. First, the user may choose to obtain recommendations based on a pre-defined policy package. This option sets the initial state (starting policy package) from which the corresponding agent will provide recommendations. Second, and most importantly, the user can define the goals importance by setting different weights using slider mechanisms, one per goal. Next, the user can specify the limit size of the recommended policy package. Finally, as an additional filter, the recommended policies can be limited by sector or region (in those sub-regional contexts) of application.


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

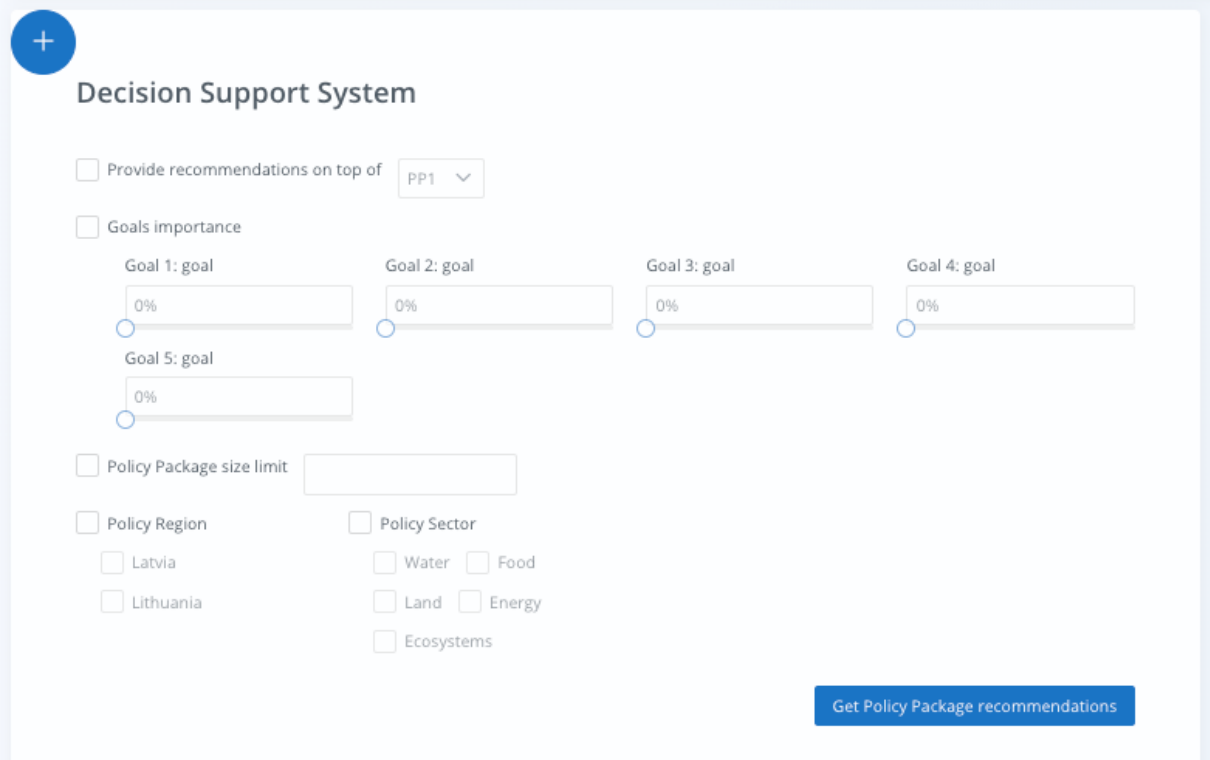


*Figure 18. NEPAT UI. Decision Support System view. Zoom on preferences section.*

Once the custom utility function is configured, the user must press the "Get Policy Package Recommendation" button. This action will trigger the DSS to obtain the recommendations, which will be listed on the left side of the screen (Figure 19).

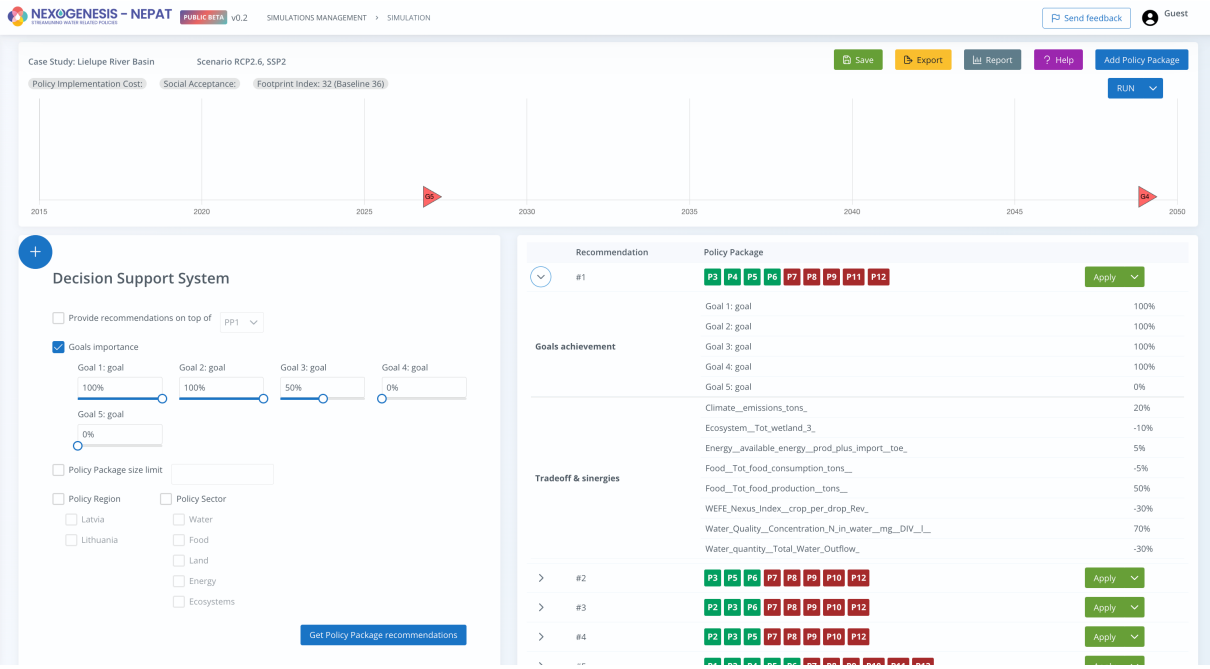


*Figure 19. NEPAT UI. Decision Support System view providing policy package recommendations*

The recommended policy packages are listed in a table (Figure 20), which can be expanded to obtain further information about the impact of these policies if applied. First, the achievement of goals is assessed, and second, the status of the nexus footprint indicators is provided. This


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

approach enables a comprehensive overview of the policy package impact across the entire nexus.

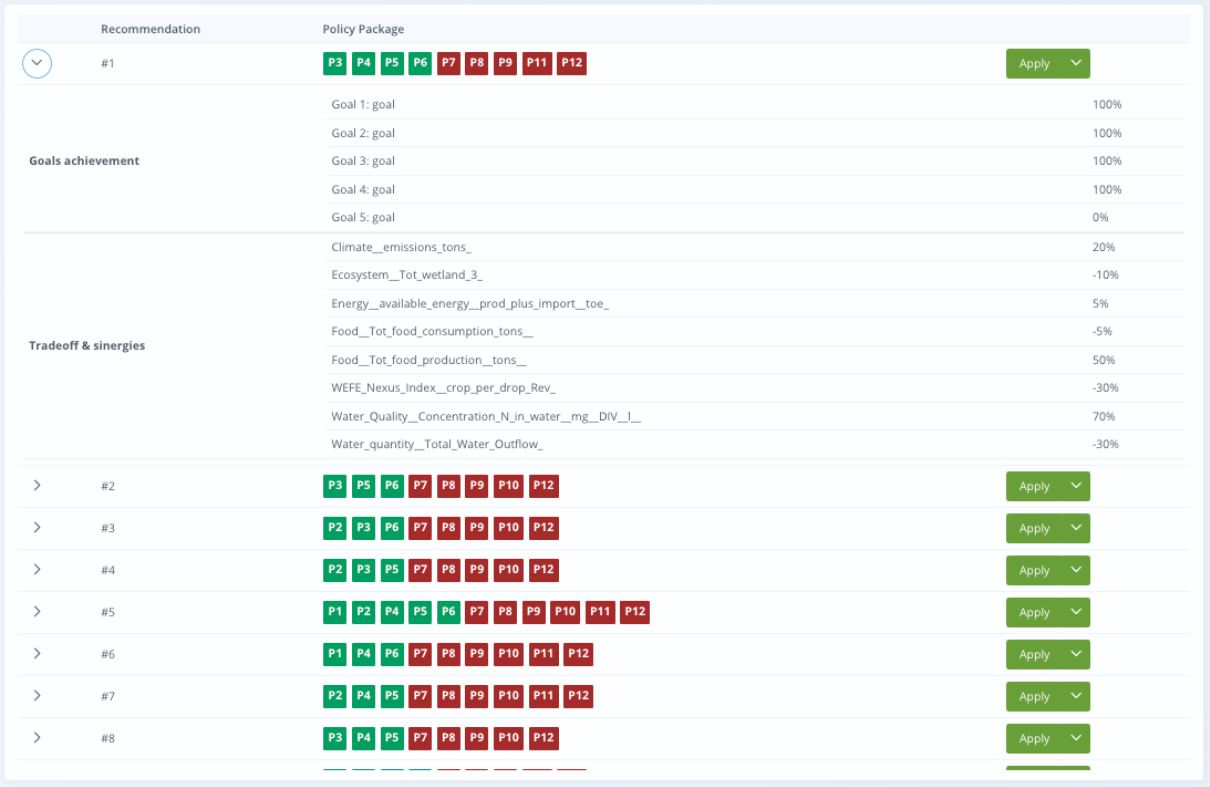


*Figure 20. NEPAT UI. Decision Support System view providing policy package recommendations. Zoom on recommendations table.*

If the user decides to accept any of the recommendations, the proposed policy package can be directly imported into the Policy Package Builder section by clicking the apply button. This allows the user to continue analyzing the impacts of that policy package through the NEPAT functionalities.


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

# 8. Conclusions

The AI algorithmic foundations of the Self-Learning Nexus Engine have been formalized, implemented, tested, and initially validated. Additionally, the Decision Support System service has been successfully implemented and deployed, achieving one of the main objectives of task T4.4 and WP4.

The self-learning nexus engine is the core mechanism that supports multi-objective decision-making in the SLNAE/NEPAT tool. The self-learning term refers to the underlying Artificial Intelligence (AI) and Machine Learning (ML) technology, enabling the creation of agents that autonomously learn (i.e. self-learning) optimal policy combinations (i.e. policy packages) to achieve the nexus-related objectives. We discuss and propose Multi Objective (Deep) Reinforcement Learning (MODRL) as the foundational family of ML algorithms to implement the NXG DSS.

Until M40 (end of task T4.4) this framework will be run for all CSs since each of the represents a unique optimization problem. Additionally, there are three further layers of complexity. First, each CS includes a set of reference scenarios (RCP-SSP combinations), each depicting different potential future conditions. Second, the complexity models implemented by WP3 incorporate randomness in the input data, allowing for both deterministic and stochastic execution modes. Finally, one of the CSs (Inkomati) introduces an additional decision-making dimension: the year when a policy is applied, which we call the 'dynamic policies' mode, in contrast to the 'static mode' with fixed policies. All these options are available through different implementations of the CSs' System Dynamic Models (SDMs). Consequently, an agent will be trained for every combination of CS, reference scenario, randomness execution mode, and, for the Inkomati CS, the policy mode, complementing those already presented.

The current version of the Self-Learning Nexus Engine is embedded in the public release of the SLNAE/NEPAT at the following urls: https://slnae-dev.nexogenesis.eu or https://nepat-dev.nexogenesis.eu.

Since the Self-Learning Nexus Engine inputs (i.e. the SDMs, the policies, the goals or the Nexus footprint) are under constant validation, the current published version of the service is designated as a Beta version and is subject to change. The DSS will be ready by August 2024 (M36) for the frontrunners CS (including the Inkomati CS), and for the followers CSs, it will be ready by December 2024 (M40). Final policy package recommendations will be reported in the corresponding WP5 deliverables (D.5.2 to D5.6) (M42).

The final version of the SLNAE is expected to be ready by February 2025 (M42) and will be reported in D4.5 *Final version of the self-assessment nexus engine with the corresponding validation* (M42).


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

# 9. References


[1] L. C. Thomas, "Constrained Markov decision processes as multi-objective problems." University of Manchester. Department of Decision Theory., 1982.

[2] L. Zadeh, "Optimality and non-scalar-valued performance criteria," *IEEE Trans. Automat. Contr.*, vol. 8, no. 1, pp. 59–60, 1963.

[3] Y. Haimes, "On a bicriterion formulation of the problems of integrated system identification and system optimization," *IEEE Trans. Syst. Man. Cybern.*, no. 3, pp. 296–297, 1971.

[4] A. Charnes, W. W. Cooper, and R. O. Ferguson, "Optimal estimation of executive compensation by linear programming," *Manage. Sci.*, vol. 1, no. 2, pp. 138–151, 1955.

[5] C. A. C. Coello, G. T. Pulido, and M. S. Lechuga, "Handling multiple objectives with particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 256–279, 2004.

[6] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Comput. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, 2006.

[7] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *Science (80-. ).*, vol. 220, no. 4598, pp. 671–680, 1983.

[8] C. M. Fonseca and P. J. Fleming, "Genetic algorithms for multiobjective optimization: formulationdiscussion and generalization.," in *Icga*, 1993, vol. 93, no. July, pp. 416–423.

[9] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.

[10] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm," *TIK Rep.*, vol. 103, 2001.

[11] Q. Zhang and H. Li, "A multiobjective evolutionary algorithm based on decomposition," *IEEE Trans. Evol. Comput.*, 2006.

[12] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley, "A survey of multi-objective sequential decision-making," *J. Artif. Intell. Res.*, vol. 48, pp. 67–113, 2013.

[13] S. Dhaubanjar, C. Davidsen, and P. Bauer-Gottwein, "Multi-objective optimization for analysis of changing trade-offs in the Nepalese water–energy–food nexus with hydropower development," *Water*, vol. 9, no. 3, p. 162, 2017.

[14] T.-S. Uen, F.-J. Chang, Y. Zhou, and W.-P. Tsai, "Exploring synergistic benefits of Water-Food-Energy Nexus through multi-objective reservoir optimization schemes," *Sci. Total Environ.*, vol. 633, pp. 341–351, 2018.

[15] M. Li, Q. Fu, V. P. Singh, D. Liu, and T. Li, "Stochastic multi-objective modeling for optimization of water-food-energy nexus of irrigated agriculture," *Adv. Water Resour.*, vol. 127, pp. 209–224, 2019.

[16] F. Karamian, A. A. Mirakzadeh, and A. Azari, "Application of multi-objective genetic algorithm for optimal combination of resources to achieve sustainable agriculture based on the water-energy-food nexus framework," *Sci. Total Environ.*, vol. 860, p. 160419, 2023.

[17] I. Okola, E. O. Omulo, D. O. Ochieng, and G. Ouma, "A comparison of evolutionary algorithms on a Large Scale Many-Objective Problem in Food–Energy–Water Nexus," *Results Control Optim.*, vol. 10, p. 100195, 2023.

[18] F. Mansour, M. Al-Hindi, M. Abou Najm, and A. Yassine, "Multi-objective optimization for comprehensive water, energy, food nexus modeling," *Sustain. Prod. Consum.*, vol. 38, pp. 295–311, 2023.

[19] R. S. Sutton and A. G. Barto, "Reinforcement learning: an introduction 2018 complete




This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

draft," 2017. doi: 10.1109/TNN.1998.712192.
[20] C. F. Hayes *et al.*, "A practical guide to multi-objective reinforcement learning and planning," *Auton. Agent. Multi. Agent. Syst.*, vol. 36, no. 1, p. 26, 2022.
[21] K. Van Moffaert, M. M. Drugan, and A. Nowé, "Hypervolume-based multi-objective reinforcement learning," in *Evolutionary Multi-Criterion Optimization: 7th International Conference, EMO 2013, Sheffield, UK, March 19-22, 2013. Proceedings 7*, 2013, pp. 352–366.
[22] O. Emamjomehzadeh, R. Kerachian, M. J. Emami-Skardi, and M. Momeni, "Combining urban metabolism and reinforcement learning concepts for sustainable water resources management: A nexus approach," *J. Environ. Manage.*, vol. 329, p. 117046, 2023.
[23] W. Zhang, A. Valencia, and N.-B. Chang, "Fingerprint Networked Reinforcement Learning via Multiagent Modeling for Improving Decision Making in an Urban Food–Energy–Water Nexus," *IEEE Trans. Syst. Man, Cybern. Syst.*, 2023.
[24] R. Wu, R. Wang, J. Hao, Q. Wu, and P. Wang, "Multiobjective multihydropower reservoir operation optimization with transformer-based deep reinforcement learning," *J. Hydrol.*, p. 130904, 2024.
[25] K. Van Moffaert and A. Nowé, "Multi-objective reinforcement learning using sets of pareto dominating policies," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 3483–3512, 2014.
[26] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3–4, pp. 279–292, May 1992, doi: 10.1007/bf00992698.
[27] K. Van Moffaert, M. M. Drugan, and A. Nowé, "Scalarized multi-objective reinforcement learning: Novel design techniques," in *2013 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*, 2013, pp. 191–199.
[28] M. Ruiz-Montiel, L. Mandow, and J.-L. Pérez-de-la-Cruz, "A temporal difference method for multi-objective reinforcement learning," *Neurocomputing*, vol. 263, pp. 15–25, 2017.
[29] M. Reymond and A. Nowé, "Pareto-DQN: Approximating the Pareto front in complex multi-objective decision problems," 2019.
[30] R. Yang, X. Sun, and K. Narasimhan, "A generalized algorithm for multi-objective reinforcement learning and policy adaptation," *Adv. Neural Inf. Process. Syst.*, vol. 32, 2019.
[31] D. M. Roijers, D. Steckelmacher, and A. Nowé, "Multi-objective reinforcement learning for the expected utility of the return," in *Proceedings of the Adaptive and Learning Agents workshop at FAIM*, 2018, vol. 2018.
[32] M. Reymond, C. F. Hayes, D. Steckelmacher, D. M. Roijers, and A. Nowé, "Actor-critic multi-objective reinforcement learning for non-linear utility functions," *Auton. Agent. Multi. Agent. Syst.*, vol. 37, no. 2, p. 23, 2023.
[33] J. Xu, Y. Tian, P. Ma, D. Rus, S. Sueda, and W. Matusik, "Prediction-guided multi-objective reinforcement learning for continuous robot control," in *International conference on machine learning*, 2020, pp. 10607–10616.
[34] F. Felten *et al.*, "A toolkit for reliable benchmarking and research in multi-objective reinforcement learning," *Adv. Neural Inf. Process. Syst.*, vol. 36, 2024.
[35] H. Hasselt, "Double Q-learning," *Adv. Neural Inf. Process. Syst.*, vol. 23, 2010.
[36] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, 2016, vol. 30, no. 1.


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

# Annexes



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881

# Annex I – Tests for MODRL agents

To test the performance of the Envelope algorithm, five random tests have been proposed for each solved case study. For each of these tests, the same 3 sets of weights were used to see the variety of responses offered by the algorithm. The following tables present the starting episodes and weights used for each CS.

*Table 9. Test episodes for Envelope*

| | Test starting episodes | | |
|---|---|---|---|
| **Test name** | **Jiu** | **Inkomati Fixed policies** | **Lielupe** |
| **T1** | [] | [] | [] |
| **T2** | [P2, P6] | [P2, P7] | |
| **T3** | [P1, P18] | [P4, P8] | |
| **T4** | [P12, P16] | [P1, P10] | |
| **T5** | [P1, P5, P15] | [P1, P5, P11] | |

*Table 10. Test weights for Envelope*

| Test weights | |
|---|---|
| **Jiu** | [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] |
| | [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0] |
| | [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.5024043002285505, 0.0, 0.4975956997714494, 0.0] |
| **Inkomati Fixed policies** | [0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142] |
| | [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] |
| | [0, 0, 0, 0, 0.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.5] |
| **Lielupe** | |
| | |
| | |


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003881