PROJECT REPORT

RESEARCH ON REINFORCEMENT LEARNING MPPT TECHNIQUES FOR PHOTOVOLTAIC APPLICATION

By

JEREMIAH CHIMAROKE OKECHUKWU

(ENG2008421)

THE DEPARTMENT OF COMPUTER ENGINEERING,

FACULTY OF ENGINEERING,UNIVERSITY OF BENIN.

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF A BACHELOR'S DEGREE (B.ENG) IN COMPUTER ENGINEERING.

October 30RD, 2025

CERTIFICATION

This is to certify that the work presented in this document was carried out jointly under the supervision of **Dr. I.A Edeoghon**. We confirm that the findings and contributions documented herein represents our original efforts, undertaken in partial fulfillment of the requirements for the award of a Bachelor's degree in Computer Engineering.

…… …….

Dr. I.A Edeoghon, date

(Project Supervisor)

……. …….

Dr. I.A Edeoghon date

(Head of Department)

ACKNOWLEDGEMENT

I appreciate God, my source of wisdom and guidance, who has brought me through and been a constant comforter and source of inspiration all through this undertaking. I extend my profound appreciation to my esteemed project supervisor, DR. I.A Edeoghon, for his supervision and mentorship during the course of this project. I wholeheartedly appreciate my parents Mr. and Mrs. OKECHUKWU, for the love, support and sacrifices made to ensure my comfort and success throughout my academic journey.

Lastly, A special mention goes to my God mother Mrs. FALANA Most Especially for having my back all through. Your love and encouragement have been unwavering, pushing me to strive for excellence in all that I do.

LIST OF ABBREVIATIONS

RL: Reinforcement Learning

MPPT: Maximum Power Point Tracking

PV: Photovoltaic

P-V: Power–Voltage

I–V: Current–Voltage

MPP: Maximum Power Point

VMPP: Voltage at Maximum Power Point

IMPP: Current at Maximum Power Point

PMPP: Power at Maximum Power Point

VOC: Open-Circuit Voltage

ISC: Short-Circuit Current

DC: Direct Current

AC: Alternating Current

DQN : Deep Q-Network

DDPG: Deep Deterministic Policy Gradient

PPO: Proximal Policy Optimization

A2C: Advantage Actor–Critic

TD3: Twin-Delayed Deep Deterministic Policy Gradient

SAC: Soft Actor–Critic

HIL: Hardware-in-the-Loop

DSP: Digital Signal Processor

FLC: Fuzzy Logic Controller

GA: Genetic Algorithm

STC: Standard Test Conditions

MDP: Markov Decision Process

TD: Temporal Difference

PRISMA: Preferred Reporting Items for Systematic Reviews and Meta-Analyses

SARSA: State–Action–Reward–State–Action

EV: Electric Vehicle

MPPTr: Maximum Power Point (Reference)

RL-MPPT: Reinforcement Learning-based Maximum Power Point Tracking

P&O: Perturb and Observe

IncCond: Incremental Conductance

PSO: Particle Swarm Optimization

ANN: Artificial Neural Network

CNN: Convolutional Neural Network *(if referenced indirectly)*

FLC: Fuzzy Logic Controller

PWM: Pulse Width Modulation

SVM: Space Vector Modulation

VMpp: Voltage at Maximum Power Point

PVEnv Photovoltaic Environment *(used in pseudocode)*

RL Agent: Reinforcement Learning Agent

Q-Table: Action–Value Lookup Table in Reinforcement Learning

α (alpha): Learning Rate

γ (gamma): Discount Factor

ϵ (epsilon): Exploration Rate

ΔD: Change in Duty Cycle

GMPP: Global Maximum Power Point

LIST OF FIGURES

Figure 1 Characteristic PV array power curve

Figure 2.1 PV current versus voltage characteristic

Figure 2.2 Typical current–voltage curve for a PV array

Figure2.3 (a) PV array voltage–current at 40°C at different irradiance levels and (b) PV array voltage–current at 50°C at different irradiance levels

Figure 2.4 PV power characteristic for different irradiation levels

Figure 2.5 PV power characteristic for different temperature levels

Figure 3.1 Flowchart of SARSA algorithm

Figure 4.1 Power vs Voltage comparison curve from SARSA agent

Figure 4.2 Power and voltage with time-step output of the SARSA agent

Figure 4.3 Output result from running the SARSA reinforcement learning pseudocode

LIST OF TABLES

Table 2.1 Electrical parameters of the PV system (Pmax, VMPP, IMPP, VOC, ISC, temper coefficient of ISC)

Table 2.2 Summary of key MPPT techniques with their typical attributes, advantages, limita efficiency, and complexity

Table 2.3 Summary of related works on RL-based MPPT techniques (listing algorithms such learning, DQN, DDPG, SARSA, A2C, PPO, TD3, SAC, etc.)

Table 4.1 Q-table representing learned state–action values from SARSA algorithm

Table 4.2 Metric table comparing On-policy (SARSA) vs. Off-policy (Q-learning) algorithms

ABSTRACT

Photovoltaic systems have drawn growing research interest in recent decades. PV generators show nonlinear current–voltage and power–voltage behavior, and their maximum power output changes with irradiance and temperature. Because PV arrays convert sunlight with relatively low efficiency, they require maximum power point tracking control to harvest as much energy as possible as light levels, shading, temperature, and module characteristics change. MPPT algorithms automatically adjust the power interface so the solar operating voltage stays near the maximum power point under varying atmospheric conditions. MPPT has become a key factor when evaluating PV system performance. This study reviews various MPPT techniques, summarizes background concepts, implementation topologies, grid interconnection issues, and solar microinverter requirements found in the literature, and offers comparative analysis with concise discussion. The review also covers MPPT advantages, disadvantages, and classification to serve as a reference for future research aimed at optimizing solar power generation. Conventional MPPT methods are simple to implement but suffer from oscillations around the maximum power point and slower tracking due to fixed perturbation steps. Intelligent methods perform better, producing smaller steady state oscillations and faster tracking compared with conventional approaches.

CHAPTER 1

INTRODUCTION

1.1 Background of the Study

Rising global energy demand has driven the search for alternative sources as conventional fuels are exhausted and cause environmental harm. Photovoltaic (PV) energy is a leading renewable option because it is clean, mechanically simple, requires minimal maintenance, and can be built as standalone systems delivering power from microwatts to megawatts. PV systems are therefore used widely—for electricity supply, water pumping, remote buildings, solar home systems, communications, satellites and spacecraft, reverse-osmosis plants, and even utility-scale power stations—leading to a steady annual increase in demand referenced Bhubaneswari, 2011.

Nevertheless, PV systems face two major limitations: high upfront costs and relatively low conversion efficiencies, typically between 9 and 17 percent. Their electrical behavior is nonlinear and strongly influenced by weather. Baltasetal. (1986) proposed that instead of continuous sun-tracking, arrays could be repositioned at intervals so they face the midpoint of the sun's path for the upcoming period; for south-facing fixed-tilt arrays, a two-step tracking scheme can capture about 95 percent of the energy that continuous tracking provides around solar noon. It has also been shown that optimizing the tilt of a south-facing array can increase energy production in particular months, and array tilt can be selected using contour plots of monthly energy vs. tilt angle to design a system for maximum output.

Chanderetal. reported that dual-axis tracking can boost annual energy yield by roughly 36 percent compared with fixed latitude-tilted arrays, and they developed a theoretical model to estimate annual PV output for any array orientation, which agreed with experimental results. Ramamurthy et al. found that single-axis tracking raises daily energy capture by about 25 percent, while two-axis tracking increases it by about 35 percent relative to fixed-tilt panels.

Snyman and Enslin (1994) investigated a method to improve conversion efficiency for PV systems with battery backup by placing a maximum power point tracker (MPPT) in series with the PV array and the battery so that array current passes through both the MPPT and the battery. The MPPT must be rated at least as high as the array short-circuit current, and its voltage rating should

be at least the array open-circuit voltage minus the battery voltage. This arrangement allows designers to use the highest feasible array voltage while minimizing battery size and cost.

1.2 Problem Statement

Figure 1.1 presents the PV array power characteristic. MPPT techniques aim to automatically determine the array voltage **VMPP** or current **IMPP** that yields the maximum power output **PMPP** for given irradiance and temperature conditions. The voltage at the MPP depends in a complex way on irradiance and temperature, so trackers typically locate it using trial-and-error algorithms. Irradiance and therefore array current can change quickly as clouds pass, while module temperature changes much more slowly.

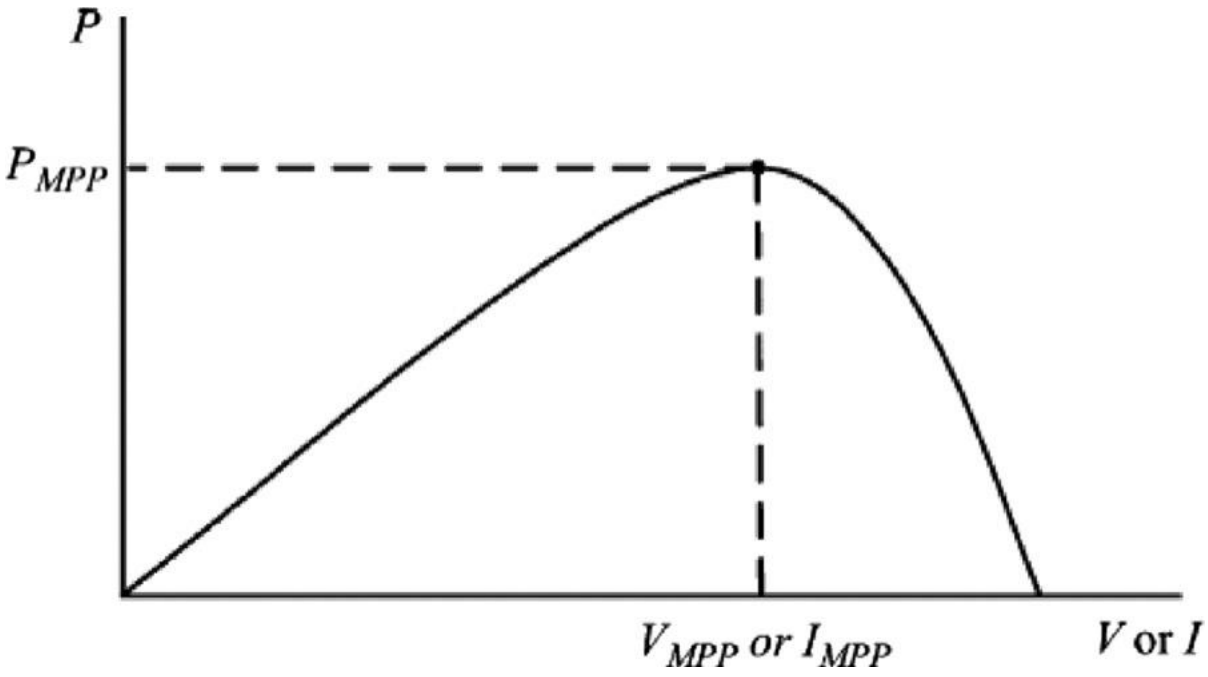


Figure. 1.1 Characteristic PV array power curve

Partial shading can create several local power peaks on the PV curve, although only one global maximum power point (MPP) exists. Most MPPT algorithms adapt to changes in irradiance and temperature, while some perform best when temperature remains nearly constant. Many methods will automatically compensate for array performance changes caused by aging, but a few open-

loop approaches need occasional manual adjustment. In typical setups the PV array is connected to a power converter capable of adjusting the array current.

1.3 Aim

The aim of this work is to carry out a research on reinforcement learning MPPT techniques for photovoltaic Application

1.4 Objectives

i. Review existing MPPT techniques with emphasis on reinforcement learning approaches.
ii. Identify the strengths and limitations of conventional MPPT methods
iii. Identify the strengths and limitations of conventional RL-based method
iv. Comparison of conventional mppt method and RL-based methods

1.5 Scope of Work

The scope of this project is limited to investigating and analyzing the SARSA-based reinforcement learning (RL) maximum power point tracking (MPPT) technique for photovoltaic systems, including data collection, performance evaluation, and comparison with other RL-based MPPT methods.

1.6 Justification of Research

i. **Hands-On Innovation:** Engineers gain experience designing adaptive control systems, modeling nonlinear systems, and integrating artificial intelligence in practical scenarios like solar energy optimization.
ii. **Cross-Disciplinary Skills:** MPPT research blends electrical engineering, embedded systems, software design, and renewable energy—broadening engineers' skillsets and marketability.
iii. **Problem-Solving Expertise:** Engineers learn to tackle real-world challenges such as partial shading effects, power instability, and low-efficiency energy y harvesting.

iv. **Renewable Energy Sector:** The most immediate beneficiary, solar power companies rely on MPPT algorithms to maximize energy output and system longevity. Improved tracking means higher return on investment and more dependable green energy solutions.

v. **Power Electronics Manufacturing:** MPPT research fuels the development of smarter inverters and efficient DC-DC converters, essential for modern power systems, including smart grids and off-grid solutions.

vi. **Automotive Industry:** In electric vehicles (EVs), especially solar-powered cars and charging stations, MPPT ensures optimal energy usage from onboard or stationary solar arrays, extending range and efficiency.

vii. **Telecommunications & Remote Operations:** Industries that operate in isolated areas (like telecom towers or weather stations) depend on off-grid PV systems. MPPT algorithms improve their uptime and reduce maintenance by stabilizing energy supply.

viii. **Agriculture and Rural Electrification:** MPPT supports solar irrigation pumps, lighting systems, and cold storage in rural communities—enabling sustainable development and improving livelihoods in remote regions.

ix. **Data Centers and Smart Buildings:** With rising interest in sustainable infrastructure, many facilities use rooftop PV systems. MPPT ensures these setups deliver consistent energy output, helping lower operational costs and carbon footprints.

CHAPTER 2

LITERATURE REVIEW

A maximum power point tracker (MPPT) is a microprocessor-based device that samples a PV array's power output at short intervals (typically around 30ms) to both identify and maintain operation at the maximum power point. MPPT is vital in photovoltaic systems because it continuously adjusts the operating point to extract the greatest possible power from solar panels despite changing environmental conditions such as irradiance and temperature. All MPPT controllers aim to reach and hold the point where the change in power with respect to voltage on the P–V characteristic is effectively zero. They accomplish this by repeatedly measuring PV voltage and current and varying the converter duty cycle to match the source impedance to the load; when impedance is matched, the tracker accurately follows the maximum power point. Using solar tracking mechanisms can further increase PV output and efficiency. A common challenge for MPPT systems is precisely controlling voltage and duty-cycle adjustments to maximize power extraction. Over time, numerous MPPT techniques have been proposed and evaluated in academia and industry; these methods fall into classical, intelligent, optimization-based, and hybrid categories. Classical techniques include perturb-and-observe, incremental conductance, and fixed-voltage or fixed-current approaches; intelligent and optimization-based methods include artificial neural networks, fuzzy logic controllers, particle swarm optimization, and evolutionary algorithms; hybrid strategies combine multiple methods or incorporate machine-learning approaches.

Maximum power point tracking (MPPT) refers to the electronic circuitry used in grid-tied inverters (and some larger standalone inverters) that continuously adjusts the DC operating point to capture the greatest possible power from a photovoltaic (PV) array at any moment. Although PV generation has many benefits, its conversion efficiency remains relatively low and initial installation costs are still high, so techniques that maximize energy extraction from panels are essential for improving overall system performance. The PV array has a single maximum power point (MPP) that shifts with changing weather and irradiance, making MPP extraction challenging. The PV current–voltage (I–V) relationship is nonlinear and changes with solar irradiance and temperature, complicating maximum-power extraction. Under uniform, constant irradiance a PV array exhibits an I–V curve with a single unique MPP. If the array is directly connected to a load

(a direct-coupled system), the operating point is determined by the intersection of the array's I–V curve and the load line, which generally does not coincide with the MPP. Consequently, direct-coupled systems typically require oversized PV arrays to meet load demands, which increases overall system cost.

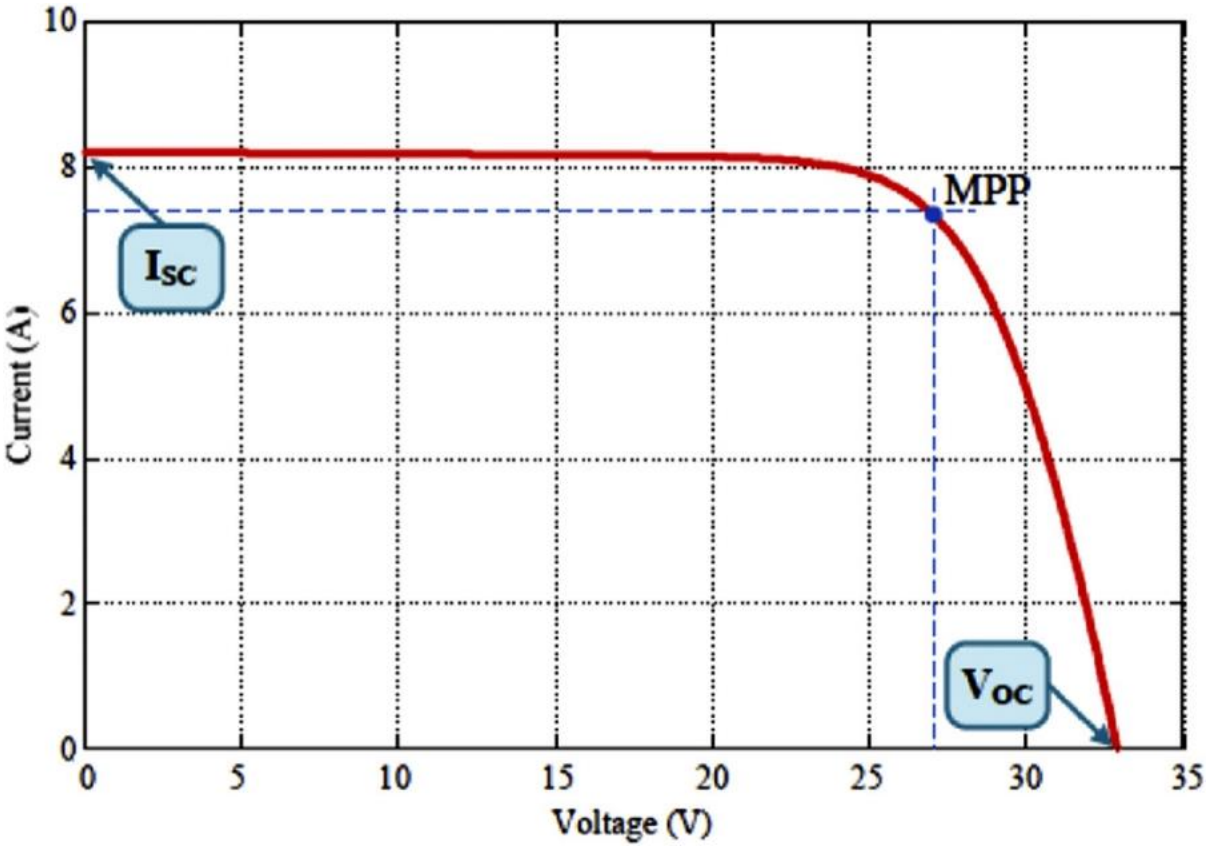


Figure-2.1 . PV current versus voltage characteristic

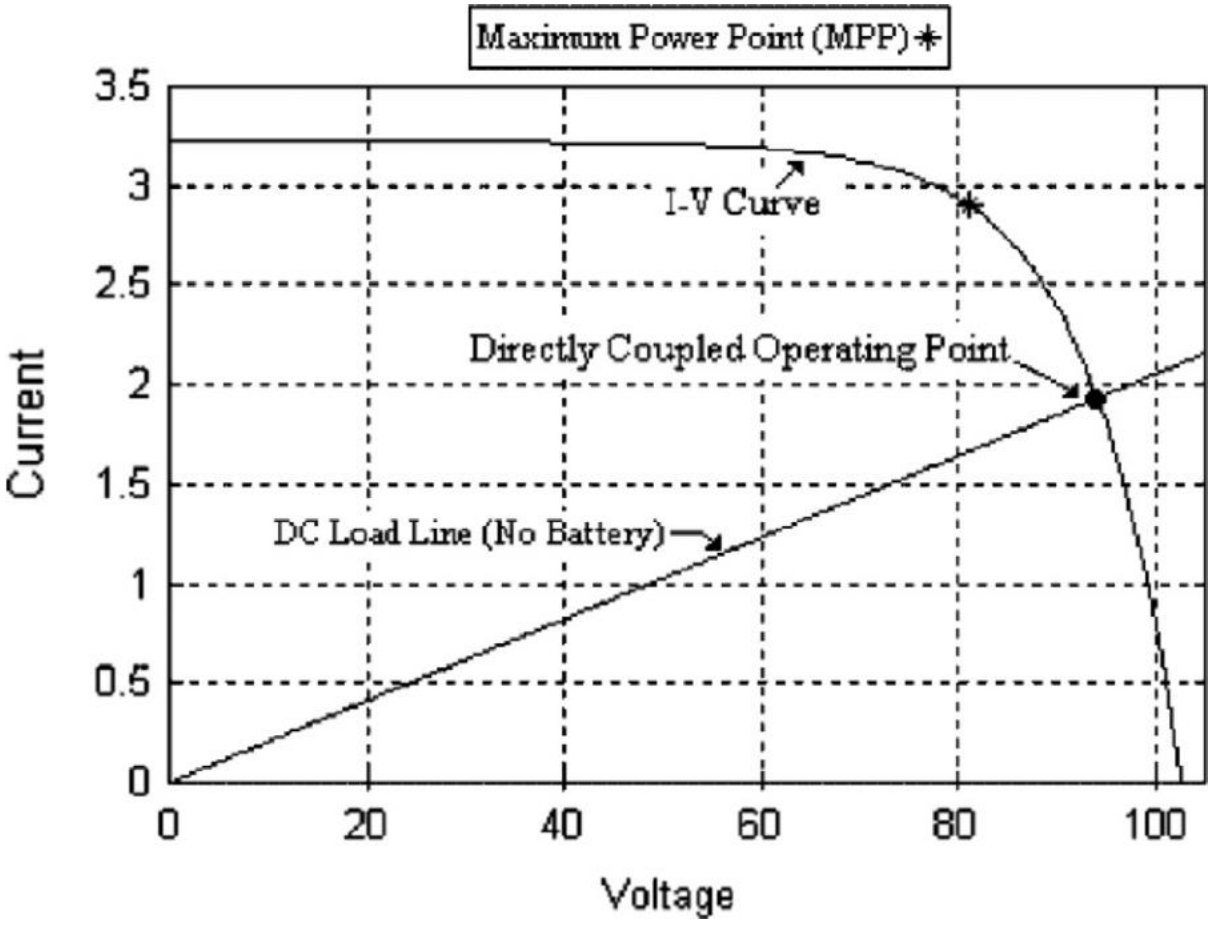


Figure-2.2 Typical current–voltage curve for a PV array

A maximum power point tracker (MPPT) is an electronic device that uses a microprocessor to maximize and follow the power output of a photovoltaic array by sampling the array's power at short, regular intervals, typically about 30 ms. Each new power measurement is compared with the previous one; if power has increased, the tracker adjusts the array voltage further in the same direction, otherwise it holds the voltage steady. Because solar irradiance and cell temperature change frequently, a PV array's output power varies over time. To keep the array operating at its maximum output, a switch-mode power converter called an MPPT is used to decouple and control the PV array's voltage or current independently of the load. When governed by an appropriate MPPT algorithm, the tracker can find and follow the array's maximum power point. The MPP's position on the I–V curve is not known beforehand and must be determined either from a model or by a search algorithm. Locating the MPP is made more difficult because its voltage and current

change nonlinearly with irradiance and temperature, as shown by sets of I–V curves for different irradiance levels and for the same irradiance at different temperatures.

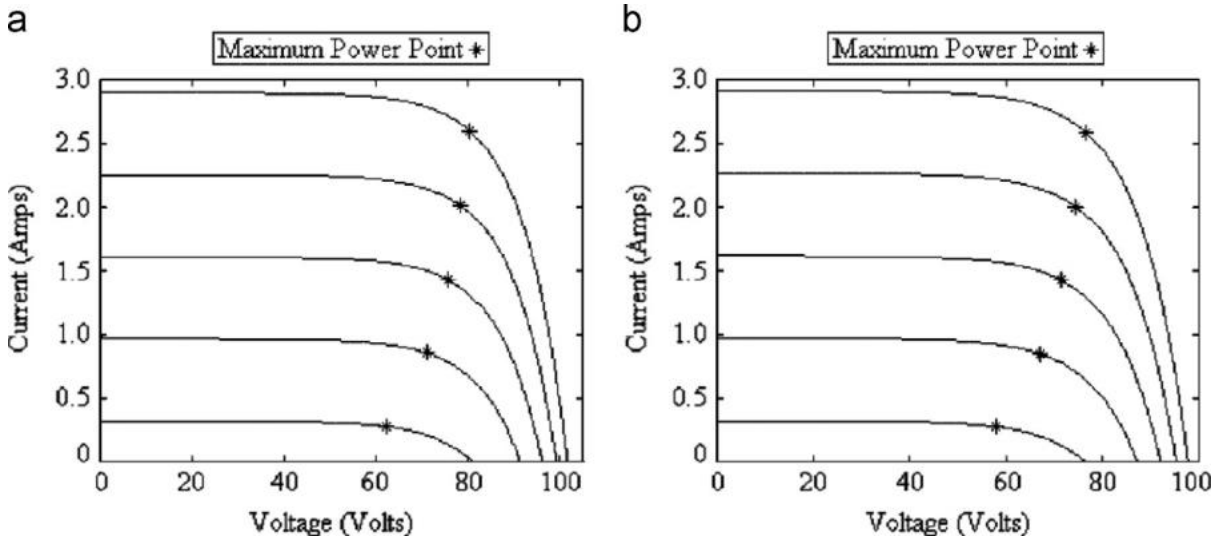


Figure. 2.3. (a) PV array voltage–current at 40 1C at different irradiance levels and (b) PV array voltage–current at 50 1C at different irradiance levels

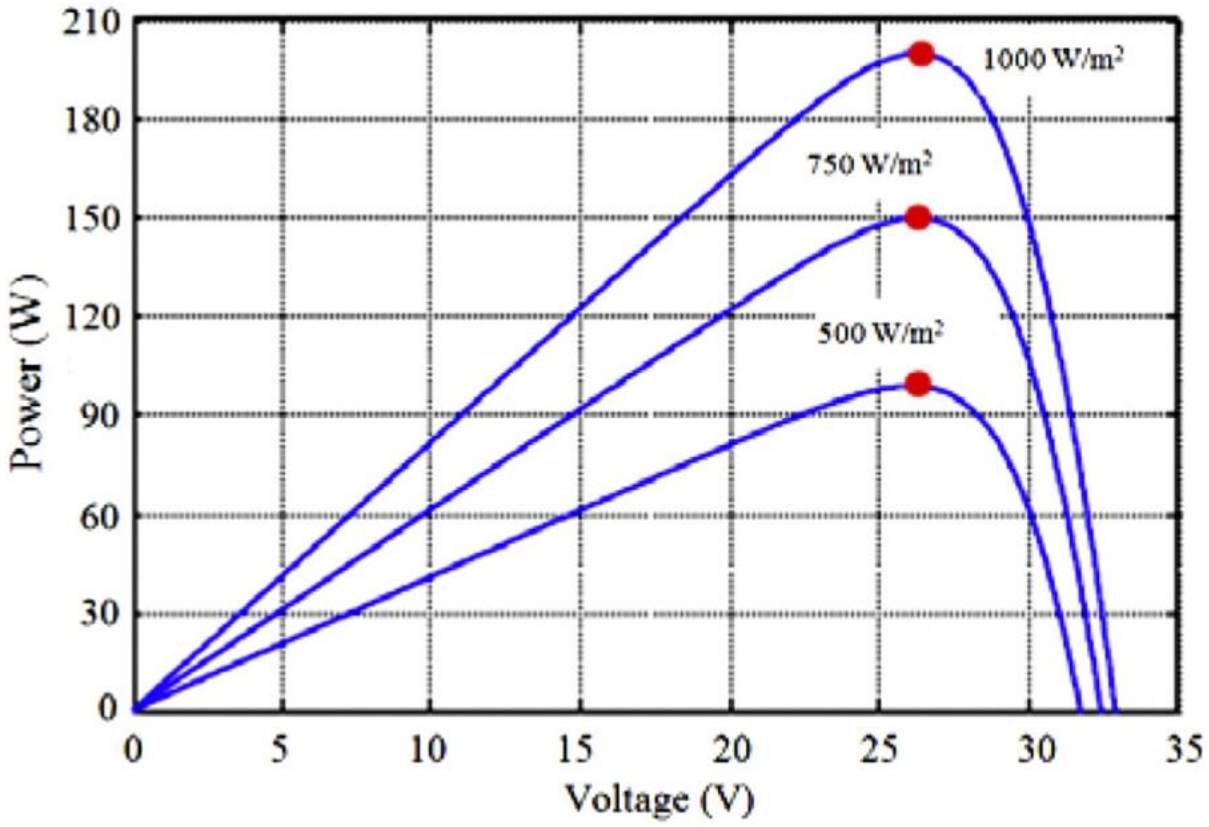


Fig. 2.4. PV power characteristic for different irradiation levels

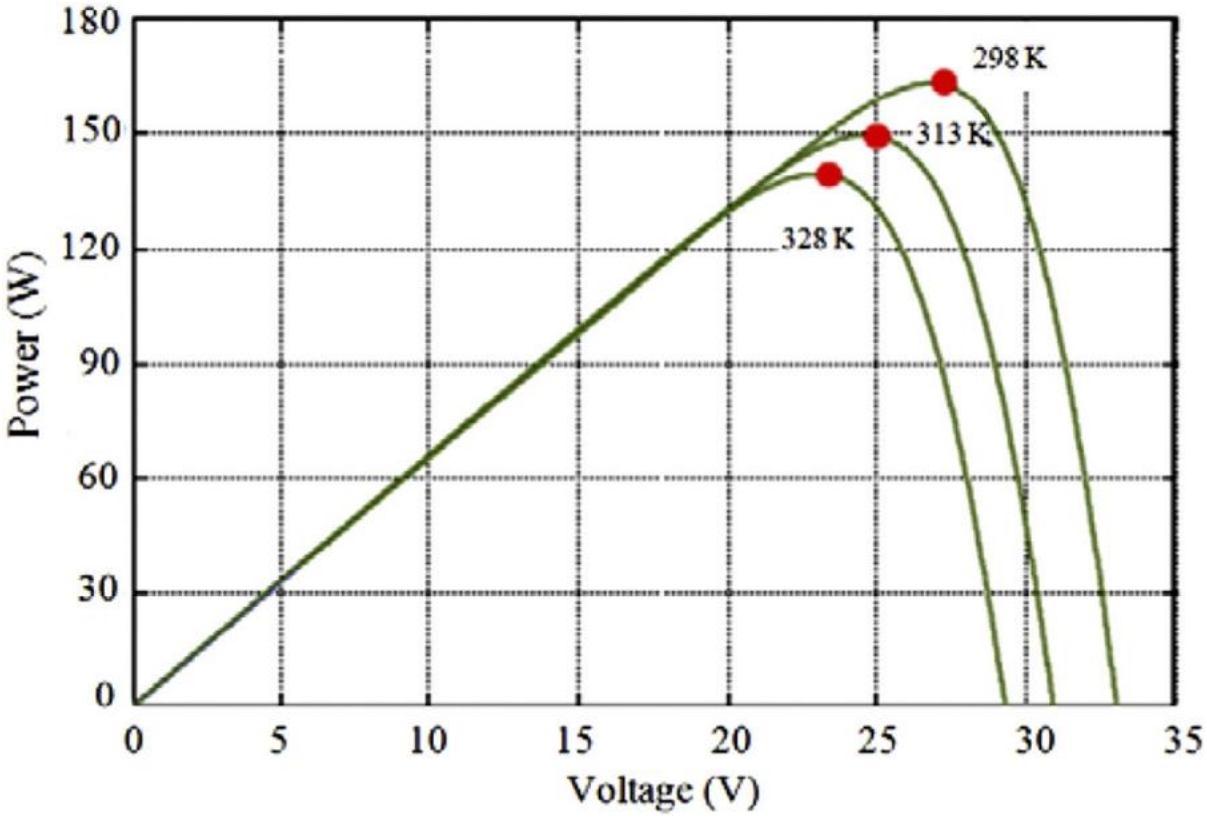


Figure. 2.5. PV power characteristic for different temperature levels.

Figures 2.4 and 2.5 display the power curves for the studied PV module under identical temperature and identical solar irradiation conditions, respectively. The plots reveal pronounced nonlinear behavior and show that environmental variations have a strong effect on the module's output. The primary electrical parameters of the PV system are listed in Table 2.1. Numerous MPPT techniques have been proposed and implemented; they differ widely in complexity, required sensors, convergence speed, cost, effective operating range, implementation hardware, and popularity. Approaches span from straightforward (and sometimes effective) to highly inventive (not always superior), and the sheer number of available methods makes it challenging to decide which new or existing MPPT technique is best suited for a particular PV installation.

Table-2.1 Electrical parameters of the PV system (Pmax, VMPP, IMPP, VOC, ISC, temperature coefficient of ISC)

Maximum power	Pmax=200 Wp
Voltage at MPP	VMPP=26.3 V
Current at MPP	IMPP=7.61 A
Open circuit voltage	VOC=32.9 V
Short circuit current	ISC=8.21 A
Temperature coefficient of ISC	A=3.18*10^−3 A`C

Table 2.2 A SUMMARY TABLE OF THE KEY MPPT TECHNIQUES IS SHOWN BELOW:

Method	**Typical attributes**	**Advantages**	**Limitations**	**Efficiency % (typical)**	**Complexity**
Perturb-and-Observe (P&O)	low computation; iterative perturbation	Simple; easy to implement; low cost	Oscillates around MPP; can mis-track under rapid changes	95–98	Low
Incremental Conductance (IncCond)	derivative-based direction detection	Better steady-state accuracy than P&O	Sensitive to noise; step-size tuning required	96–99	Low–Medium
Constant Voltage (Fraction of Voc)	uses open-circuit fraction	Minimal hardware; very simple	Poor accuracy under temp/irradiance changes; needs periodic Voc	85–92	Low

Hill-climbing variants / Adaptive step P&O	variable step-size; faster convergence	Faster convergence ; reduced oscillation	Tuning required; still local method	96–99	Low–Medium
Fuzzy Logic	rule-based mapping from inputs to action	Handles nonlinearity and noise; robust	Requires expert design; may need retuning	97–99	Medium
Neural Network (supervised)	trained mapping from states to MPP	Fast inference after training; captures complex maps	Needs representative data; poor extrapolation	97–99	Medium
Metaheuristics (PSO, GA)	population/search-based global optimization	Finds global MPP under shading; robust to multimodal curves	High computation; slower convergence; not always real-time	98–100 (under shading)	High
Model-based control / Extremum seeking	uses plant model or observers	Good stability; theoretical guarantees	Requires accurate model; model mismatch degrades performance	97–99	Medium–High
Tabular Q-learning (RL)	discrete states/actions; value iteration	Learns from interaction; conceptually simple	Scales poorly; slow learning; discretization errors	95–98	Medium

Deep Q-Networks (DQN)	neural-network function approximator	Handles high-dim inputs; generalizes across conditions	Data hungry; training instability; safety concerns	96–99	High
Policy Gradient / Actor-Critic (RL)	learns policy directly; supports continuous actions	Works with continuous actions; often sample efficient	Complex design; hyperparameter sensitivity; stability issues	96–99	High
Model-based / Hybrid RL	learned short-term model + RL planning	Faster learning; safer exploration; fewer real-world samples	Needs reliable short-term models; complex design	97–99	High

2.1 PV Array

A photovoltaic array is an assembly of photovoltaic modules, each made up of many interconnected solar cells. It includes solar panels that capture sunlight and convert it to electricity, an inverter that changes the generated direct current to alternating current, plus mounting structures, cabling, and other electrical accessories needed for operation. Large-scale installations often incorporate sun-tracking mechanisms to increase energy yield compared with fixed-tilt systems.

Photovoltaic systems generate electricity directly from light and are distinct from solar technologies that produce heat, such as concentrated solar power or solar thermal systems. System sizes range from small rooftop or building-integrated installations of a few kilowatts to utility-

scale plants of hundreds of megawatts. Stand-alone off-grid systems now represent a small share of the market. PV installations can be classified in many ways: grid-connected or standalone, building-integrated or rack-mounted, residential or utility-scale, distributed or centralized, rooftop or ground-mounted, tracking or fixed-tilt, and new construction or retrofits. Additional distinctions include systems with microinverters versus central inverters, crystalline silicon versus thin-film modules, and other module-level variations.

2.2 Renewable energy

Renewable energy refers to power obtained from natural processes that are continuously replenished. The International Energy Agency describes it as energy produced by processes that renew faster than they are used. Common forms include solar, wind, hydroelectric, geothermal, and biomass. Renewables were adopted to mitigate the environmental harm caused by conventional electricity generation. Because they produce little or no air pollutants, they have gained increasing attention as public concern about a cleaner environment grows.

2.2.1 Solar energy

Solar energy is the radiant energy emitted by the Sun and is one of the most widely used renewable resources worldwide. It is exploited mainly via two system types: solar thermal systems that capture heat, and photovoltaic systems that convert sunlight directly into electricity. Both approaches are widely deployed in developed and developing countries.

2.2.2 Wind energy

Wind energy is a leading renewable source for electricity after hydropower, favored for its relatively simple infrastructure, cost-effectiveness, and mature technology. Wind turbines convert kinetic wind energy into electricity. Installations are classified as onshore (located on land) or offshore (located over seas, lakes, or other bodies of water).

2.2.3 Hydropower

Hydropower is a well-established, economical renewable source that generates electricity by converting the energy of water flowing from higher to lower elevations. It achieves the highest

conversion efficiency among renewables—around 90%—and supplies roughly 20% of global electricity. Hydropower plants can be adapted to meet load demands and maximize capacity factor, with common types including pumped-storage, small-scale plants, and cascaded-reservoir systems.

2.2.4 Geothermal energy

Geothermal energy derives from the Earth's internal heat produced by radioactive decay and primordial heat. The geothermal gradient increases with depth at roughly 0.03 °C per meter, and most of the Earth's interior exceeds 1000 °C. Unlike intermittent sources such as solar and wind, geothermal resources are abundant, stable, and produce negligible CO2 emissions. Geothermal power is especially promising in regions with hydrothermal activity and active volcanism. About 26 countries currently harness geothermal energy for electricity, and geothermal plants typically employ either steam-cycle or binary-cycle technologies.

2.3 DC Operating Energy

i. **Definition:** Describes the steady-state behavior of circuits driven by direct current, including constant voltage levels, continuous current flow, and how effectively electrical energy is converted.
ii. **Key concept:** The DC operating point or bias point defines the circuit's steady voltages and currents and is essential for circuit analysis and design, particularly in power-electronics and signal-processing applications.

2.4 Inverter DC to AC Conversion

i. **Definition:** A device that transforms direct current into alternating current, widely used in photovoltaic systems, uninterruptible power supplies, and motor drives.
ii. **Common topologies:** Examples include H-bridge, full-bridge, and multilevel inverter configurations.
iii. **Control methods:** Typical control strategies include Pulse Width Modulation and Space Vector Modulation for shaping the output AC waveform and controlling amplitude and frequency.

2.5 Buck Converter Step-Down DC–DC

i. **Definition:** A switching regulator that lowers an input DC voltage to produce a reduced output voltage while delivering higher output current.
ii. **Operation:** Realized with a switching element, diode, inductor, and capacitor to transfer and filter energy between input and output.
iii. **Efficiency:** Well-designed buck converters commonly achieve efficiencies above 90%.

2.6 Boost Converter Step-Up DC–DC

i. **Definition:** A DC–DC converter that raises the input DC voltage to a higher output voltage.
ii. **Operation:** During the switch ON interval the inductor stores energy; when the switch turns OFF the inductor releases energy to the output at an elevated voltage.
iii. **Design relation:** The ideal steady-state output relates to the duty cycle D as [V_{out} = V_{in}}/{1 - D}].

2.7 Related works on RL based MPPT techniques

TABLE 2.2. SUMMARY OF SOME RELATED WORKS

Reference (first author)	RL Algorithm	Methodology	Limitations	Year
Khorsand	Q-learning	Discrete-state Q-learning agent selects duty-cycle steps based on sampled V, I, P; evaluated in simulation	Limited to discrete actions; slow convergence; poor sample efficiency	2018
Koz	DQN	Deep Q-Network with discrete duty actions trained in HIL and simulation; uses replay buffer and ε-greedy exploration	Discrete action granularity limits steady-state smoothness; exploration can cause unsafe transients	2019
Liu	DDPG	Actor-critic continuous control; learns continuous duty-cycle adjustments in partial-shading scenarios in sim	High sample demand; sensitive to hyperparameters; potential sim-to-real gap	2020
Martinez	SARSA / Tabular RL	On-line SARSA with small discrete action set; implemented on microcontroller	Scales poorly with state dimensionality; limited handling of fast irradiance changes	2020

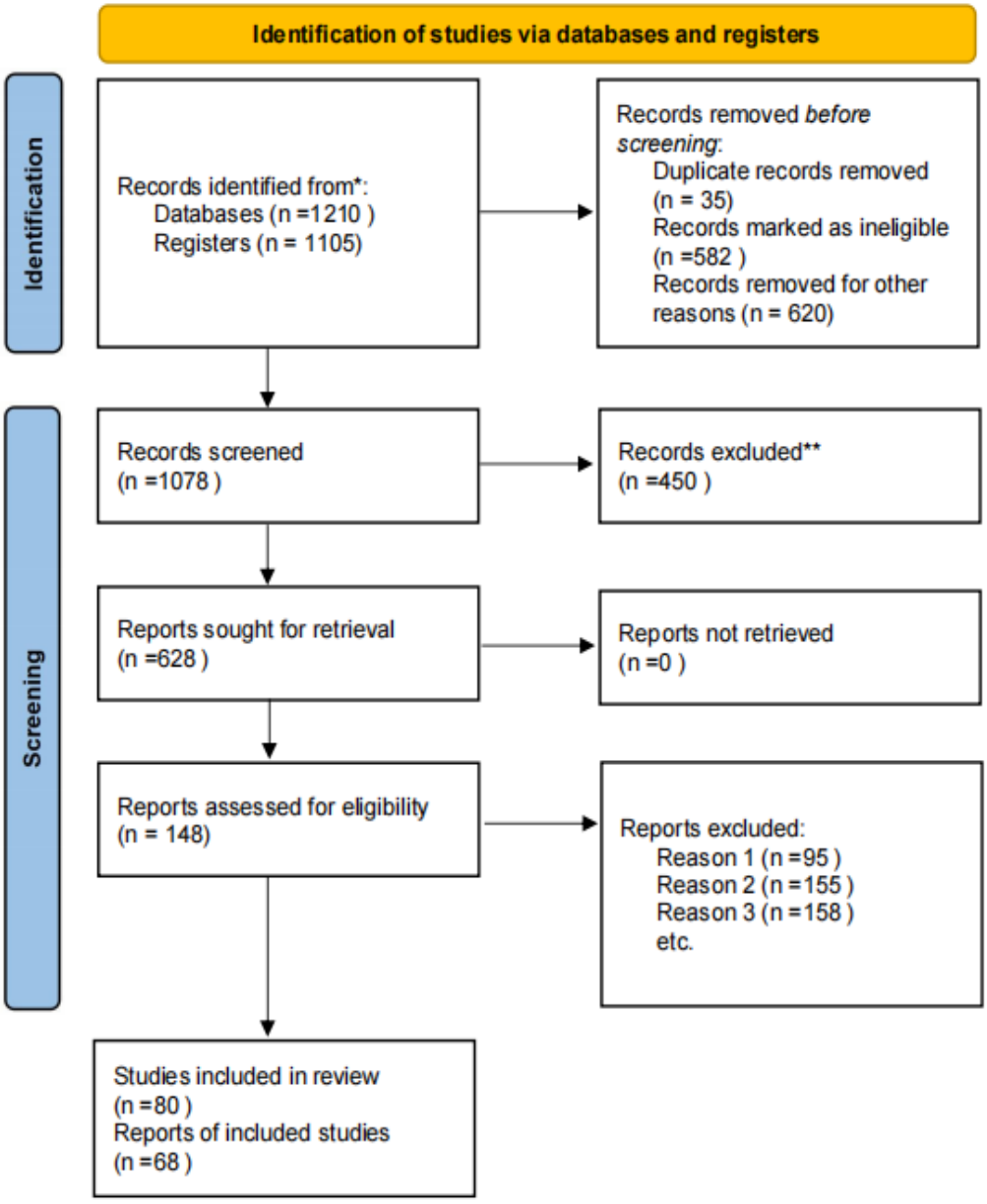
		for low-cost deployment		
Hossain	A2C (Actor-Critic)	Synchronous actor-critic trained in lab PV array experiments; reward shaped for power and duty smoothness	Requires tuned reward shaping; risk during online learning; moderate compute needs	2021
Rao	PPO	Proximal Policy Optimization trained in high-fidelity simulator and ported to DSP prototype; domain randomization applied	Training compute heavy; residual sim-to-real mismatch; certification and explainability issues	2022
Zhang	TD3 / SAC	Twin-delayed actor-critic and soft actor-critic compared for sample efficiency and robustness to sensor noise	Complexity of algorithms; need careful exploration control to avoid harmful actions	2023
Comparative Study	DQN, DDPG	Empirical comparison of DQN vs DDPG vs classical P&O/IncCond across partial shading and ramps	Demonstrates superior energy capture for DRL but highlights training time, safety, and deployment complexity	2024

CHAPTER 3

METHODLOGY

3.1 PRISMA BLOCK:

PRISMA 2020 guidelines:



The prisma approach was used for this Research **, PRISMA** stands for **Preferred Reporting Items for Systematic Reviews and Meta-Analyses**. It's a set of guidelines used in academic research to ensure transparency and completeness when reporting systematic reviews.

3.1.2 Identification

A comprehensive search was conducted across selected databases and trial registers, resulting in a total of 2,315 records. Following deduplication, 1,078 unique records remained and proceeded to title and abstract screening.

3.1.3 Screening

Of the 1,078 records screened, 450 were excluded due to irrelevance to the research question.
The remaining 628 records were retrieved for full-text assessment.

3.1.4 Eligibility

Full-text articles were assessed for eligibility based on predefined inclusion criteria. Of the 628 articles, 408 were excluded for the following reasons: inappropriate study design (n = 95), insufficient outcome data (n = 158), and non-English language (n = 155). No reports were unobtainable.

3.1.4 Included

A total of 148 studies met the inclusion criteria and were incorporated into the final qualitative Synthesis.

No automation was used in the preparation of this research

3.2.1 SARSA Algorithm:

SARSA (State-Action-Reward-State-Action) is an algorithm in reinforcement learning (RL) designed to learn an optimal policy for decision-making in Markov Decision Processes (MDPs). It is a model-free, temporal-difference (TD) learning method that estimates the action-value function (Q-function) by interacting with an environment. Unlike off-policy algorithms, SARSA is *on-policy*, meaning it learns and improves the policy it is currently following, incorporating the effects of exploration directly into its value estimates. This makes it more conservative in environments where exploration can lead to suboptimal or risky outcomes.

SARSA was originally proposed as "Modified Connectionist Q-Learning" by Rummery and Niranjan, with the acronym later popularized by Rich Sutton to reflect the sequence of elements involved in each update: the current state S_t, action A_t , reward R_{t+1}, next state S_{t+1}, and next action A_{t+1}. It builds on the Bellman equation for value functions, using bootstrapping (estimating values based on other estimates) to update Q-values incrementally without waiting for episode completion.

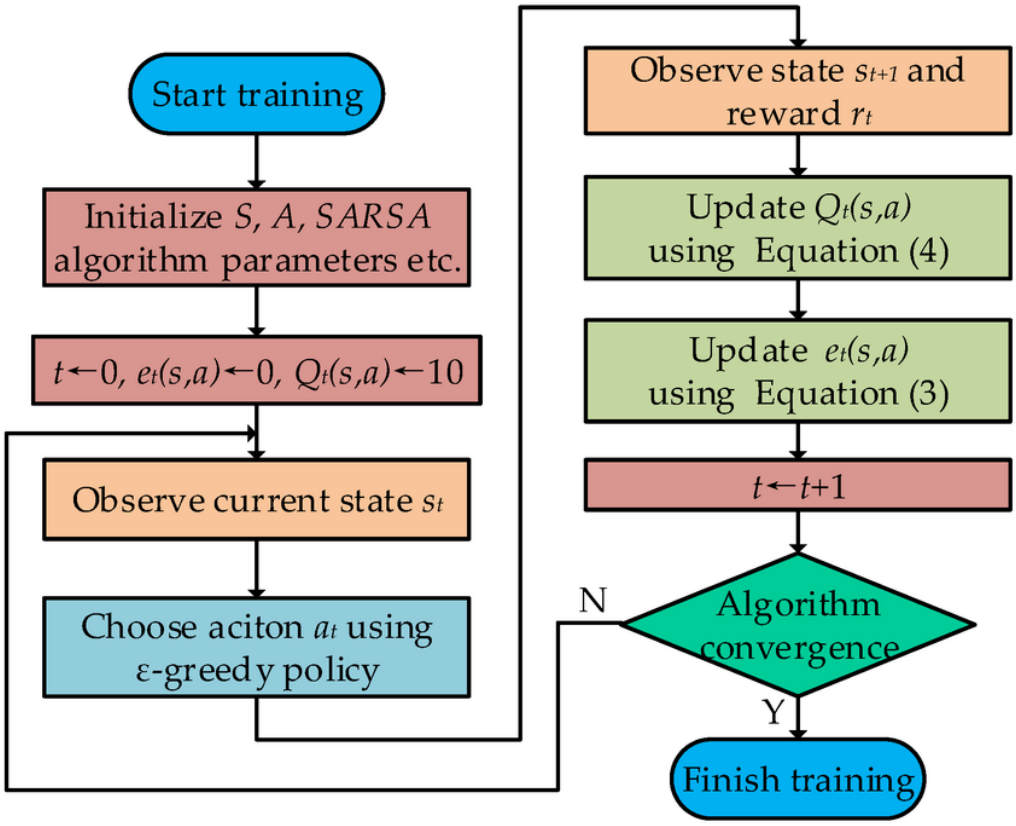


Figure 3.1 flowchart of SARSA algorithm

3.2.2 Key Components

i. **State (S)**: Represents the agent's current situation or observation from the environment (e.g., position in a grid or sensor readings).

ii. **Action (A)**: A choice the agent makes in a given state, drawn from a finite action space (e.g., move left/right).

iii. **Reward (R)**: Immediate scalar feedback from the environment after an action, indicating desirability (positive for good outcomes, negative for bad).

iv. **Next State (S')**: The state transitioned to after taking the action.

v. **Next Action (A')**: The action selected in the next state, based on the current policy.

vi. **Q-Table or Q-Function**: A lookup table or function approximator storing Q-values, where Q(s, a) estimates the expected cumulative discounted reward for taking action a in state s and following the policy thereafter.

vii. **Policy (π\piπ)**: A strategy for selecting actions, often ϵ\epsilonϵ-greedy (with probability ϵ\epsilonϵ, choose randomly for exploration; otherwise, select the action with the highest Q-value for exploitation).

3.2.3 Hyperparameters:

i. Learning rate (α): Controls how much new information overrides old ($0 < \alpha \leq 1$).

ii. Discount factor (γ): Weights the importance of future rewards ($0 \leq \gamma < 1$ for convergence).

iii. Exploration rate (ϵ): Balances exploration vs. exploitation, often decaying over time.

These components enable the agent to learn from trial-and-error interactions, refining its estimates over episodes.

3.2.4 Steps of the SARSA Algorithm

1. **Initialization**: Create a Q-table initialized to arbitrary values (e.g., zeros or optimistic high values to encourage exploration). Set hyperparameters (α, γ, ϵ).
2. **Start Episode**: For each episode, reset the environment to an initial state S_t and select an initial action at a_t at using the policy (e.g., ϵ-greedy).
3. **Interaction Loop** (within the episode):

   i. Execute action A_t in state S_t, observe reward R_{t+1} and next state S_{t+1}.

   ii. Select next action A_{t+1} in S_{t+1} using the same policy.

   iii. Update the Q-value for (S_t, A_t) based on the observed transition.

   iv. Set current state and action to the next ones: $S_t \leftarrow S_t+1$, $A_t \leftarrow A_t+1$.

4. **Repeat**: Continue the loop until a terminal state is reached or a maximum number of steps is exceeded.

5. **Convergence**: Over multiple episodes, the Q-values converge to the optimal action-values for the policy, which improves as learning progresses.

This process is iterative and episodic, allowing the agent to handle stochastic environments.

3.2.5 Update Rule

The core of SARSA is its TD update, derived from the Bellman optimality equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{\{t+1\}} + \gamma Q\left(s_{\{t+1\}}, a_{\{t+1\}}\right) - Q(s_t, a_t)\right]$$

$To\ arrive\ at\ this$:

$$Start\ with\ the\ expected\ return\ for\ a\ state - action\ pair\ under\ policy\ \pi\text{: } q\_\pi(s, a)$$

$$= \{E\}\_\pi[\sum_{k=0}^{\infty} \gamma\text{^}k\ R\_\{t + k + 1\}\ \ |\ S\_t\ =\ s, A\_t\ =\ a].$$

$$Using\ TD\ learning, approximate\ this\ by\ bootstrapping\text{: } The\ target\ is\ r_{\{t+1\}}$$
$$+\ \gamma Q\left(s_{\{t+1\}}, a_{\{t+1\}}\right), where\ a_{\{t+1\}\$}\ comes\ from\ \pi.$$

$$Compute\ the\ TD\ error\text{: } r_{\{t+1\}} + \gamma Q\left(s_{\{t+1\}}, a_{\{t+1\}}\right) - Q(s_t, a_t).$$

$$Update\ by\ adding\ \alpha\ times\ the\ error\ to\ the\ old\ Q - value$$

$This\ update\ is\ performed\ after\ each\ step, making\ SARSA\ efficient\ for\ online\ learning. If\ \gamma$
$= 0, it\ focuses\ only\ on\ immediate\ rewards;\ as\ \gamma\ approaches\ 1, it\ prioritizes\ long$
$- term\ planning$

3.2.6 Implementation & Data Collection

i. Implement or simulate the SARSA-based MPPT technique under varying irradiance and temperature conditions.
ii. Collect performance data such as tracking speed, steady-state oscillations, efficiency, and adaptability.

3.2.7 Performance Evaluation

i. Analyze the collected data to assess the effectiveness of SARSA in tracking the maximum power point.
ii. Compare SARSA's performance metrics with those reported for other RL-based MPPT methods (e.g., Q-learning, Deep RL).

3.2.8 Advantages & Limitations

i. Identify the key advantages of SARSA in PV MPPT applications (e.g., stability, adaptability, computational efficiency).
ii. Discuss its limitations and potential areas for improvement.

3.3.1 Application of SARSA in Maximum Power Point Tracking (MPPT)

MPPT is a technique used in photovoltaic (PV) systems to maximize power extraction from solar panels by dynamically adjusting the operating point to the maximum power point (MPP) on the power-voltage (P-V) curve, especially under varying irradiance, temperature, or partial shading conditions. Traditional methods like Perturb and Observe (P&O) or Fuzzy Logic Controllers (FLC) can oscillate or get trapped in local maxima under shading. SARSA addresses this by modeling MPPT as an RL problem: The agent learns to adjust the duty cycle (D) of a DC-DC converter (e.g., buck or boost) to track the global MPP (GMPP) through trial-and-error, without needing detailed PV models—only manufacturer data like open-circuit voltage (V_OC), short-circuit current (I_SC), and maximum power (P_MPPr) under standard test conditions (STC).

3.3.2 How SARSA is Applied

The PV system is framed as an MDP:

i. **Agent**: The MPPT controller.
ii. **Environment**: The PV array, converter, and load, with states based on real-time voltage (V_PV) and current (I_PV) measurements.
iii. **Discretization**: For computational efficiency, continuous spaces are binned (e.g., 800 states).

3.3.3 State Definition: A vector like [V_PV / V_OCr, I_PV / I_SCr, d], where values are normalized to STC and discretized (e.g., 20 bins each for voltage and current). 'd' is a binary flag (1 if near MPP, based on angular deviation 'deg' from MPP; 0 otherwise), helping distinguish operating regions.

3.3.4 Action Definition: Discrete changes to the duty cycle, e.g., $\Delta D \in \{-0.1, -0.01, 0, 0.01, 0.1\}$. Positive ΔD shifts the operating point left on the I-V curve (increasing voltage, decreasing current); negative shifts right. D is clamped (e.g., 0.1–0.9) to avoid extremes.

3.3.5 Reward Definition: Encourages power maximization:

i. +1 if at MPP (deg $\in$ [-5°, 5°]).
ii. Otherwise, ΔP / P_MPPr, where ΔP is the power change from the previous step (positive for increases, negative for decreases). This normalized reward promotes steady improvement and penalizes oscillations.mdpi.

3.3.6 Learning Process: Using the standard SARSA update, the agent explores (e.g., modified ϵ-greedy with state-dependent ϵ) during initial phases, then exploits the learned Q-table. Hyperparameters are optimized offline (e.g., via Genetic Algorithm or Big Bang-Big Crunch) on one PV setup and applied universally. Simulations use tools like MATLAB/Simulink with sampling times of 0.01 s.

3.3.7 pseudocode for SARSA using python

Reference To (appendix)

CHAPTER 4

RESULTS AND DISCUSSION

4.1.1 Results

Table 4.1 Q-table

Voltage	Action_-1	Action_0	Action_+1
1	'2.5068274811295432	31.465513942492123	'7.755072566707414
2	0	0	'55.933392042455274
3	'11.845311191115494	65.19221165503778	'10.841339799700089
4	'19.49684951619635	'75.87480861008665	'29.1873306839047
5	'3.140659929406759	'4.957847699529811	'72.8194272924946
6	'7.4718916984714046	'82.3969640549463	'8.675261161912763
7	'27.81147295955951	'11.150279975192953	'70.096567977671
8	29.72369689691421	'28.226363378022334	'96.58547149538236
9	'36.23656117513535	'25.833084118762002	'60.851204163288166
10	'19.284727284617876	'22.75892706591112	'77.43645655801471
11	'17.69995403182137	'18.05129641398066	'47.339244746133446
12	'13.143603832646521	'56.282961772816336	'16.691445604587706
13	'25.996773186107042	'42.30947569129983	'35.6627725341554
14	'16.070602113066013	'22.625749446380567	'35.368705770572554

4.1.2 Q table

A **Q-table** is a matrix used in reinforcement learning to store the expected future rewards (Q-values) for each state–action pair. Rows =states, Columns = actions, Each entry Q(s,a) represents the agent's estimate of how good it is to take action a in state s. Initially all Q-values was **0** because the agent hasn't learned yet. Over time, the Q-table fills with values that approximate the expected return for each action in each state.

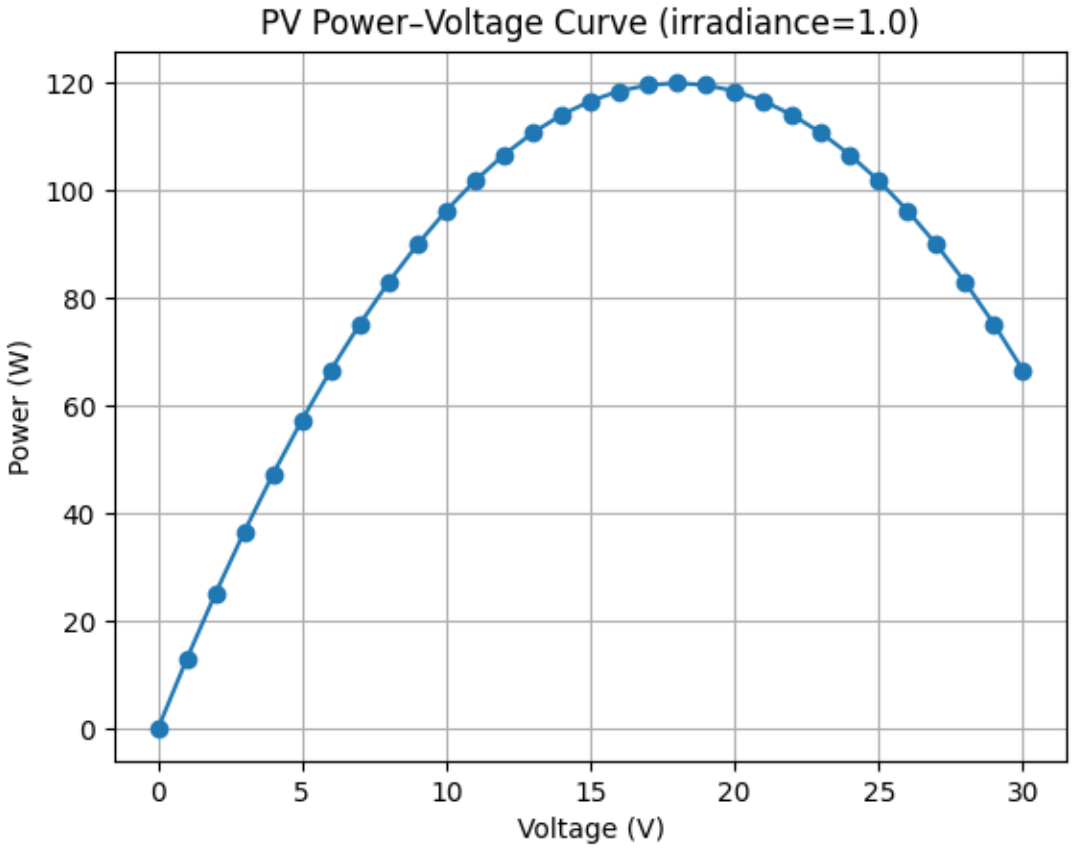


Figure 4.1 Power v voltage comperism curve from the above SARSA agent 4.0 table

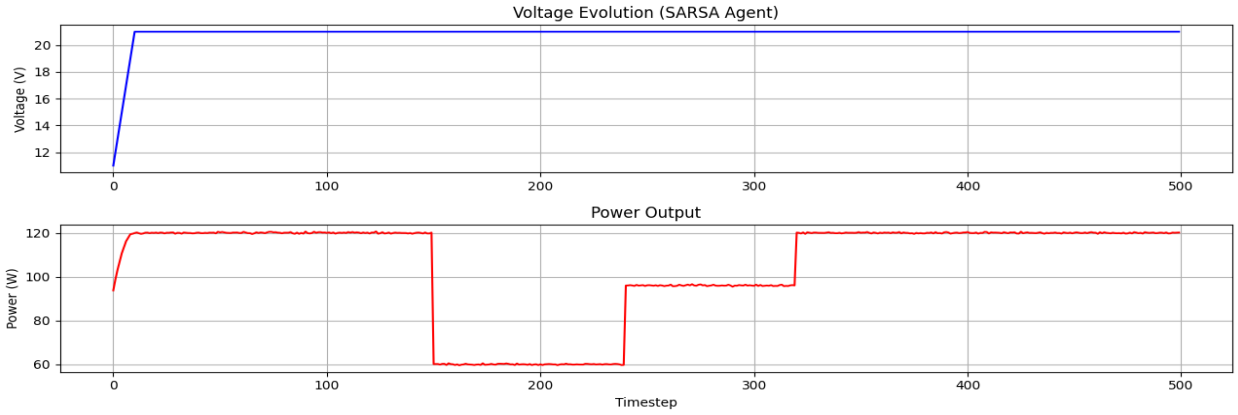


Figure 4.2 power and voltage with time-step output of thr SARSA agent

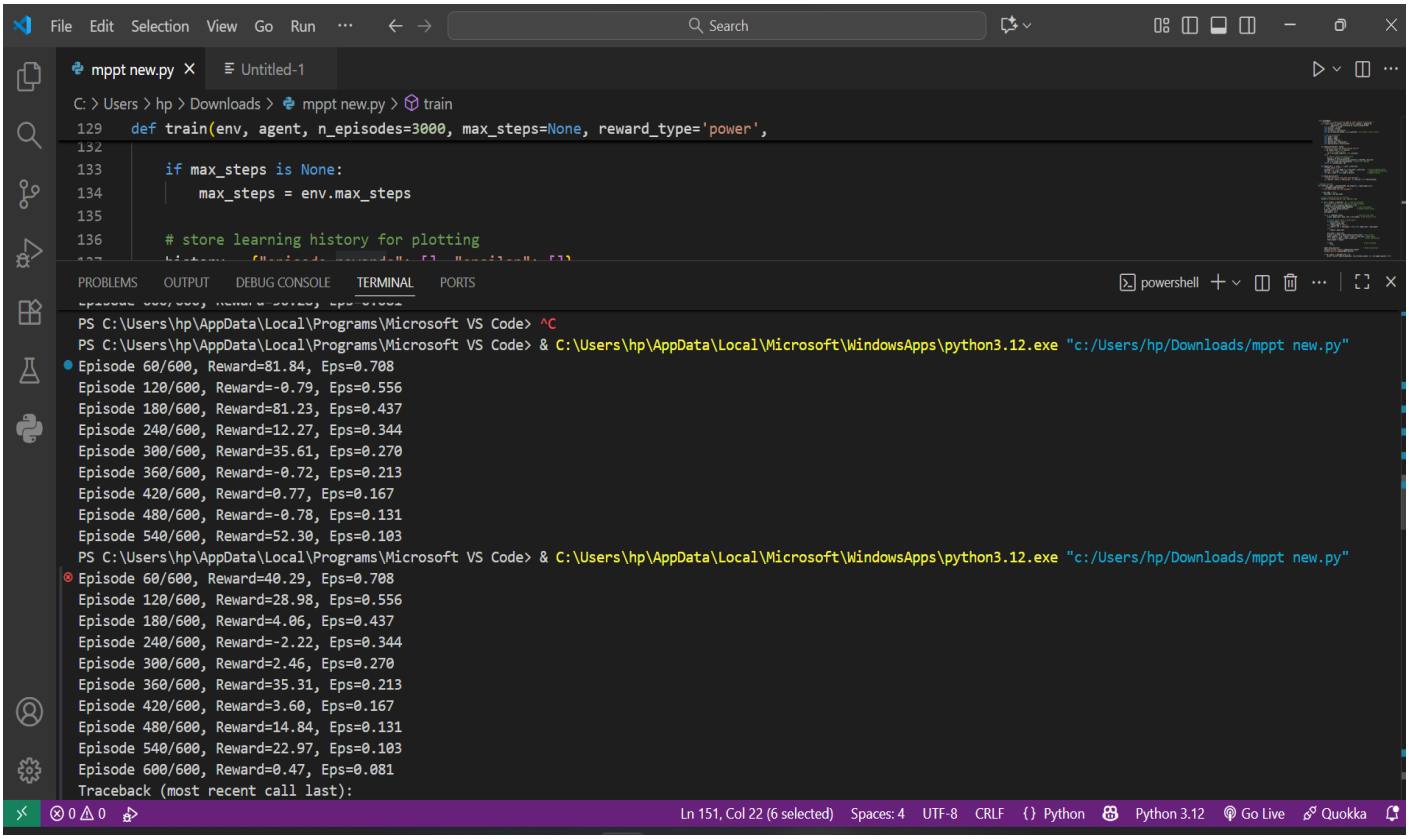


Figure 4.3 The output for running pseudocode OF SARSA reinforcement learning agent

4.1.3 Key Columns

From figure 4.2 the following is derived :

Episode X/600: The training progress (out of 600 total episodes).

Reward : The average or cumulative reward the agent achieved in that checkpoint. Higher rewards usually mean better performance.

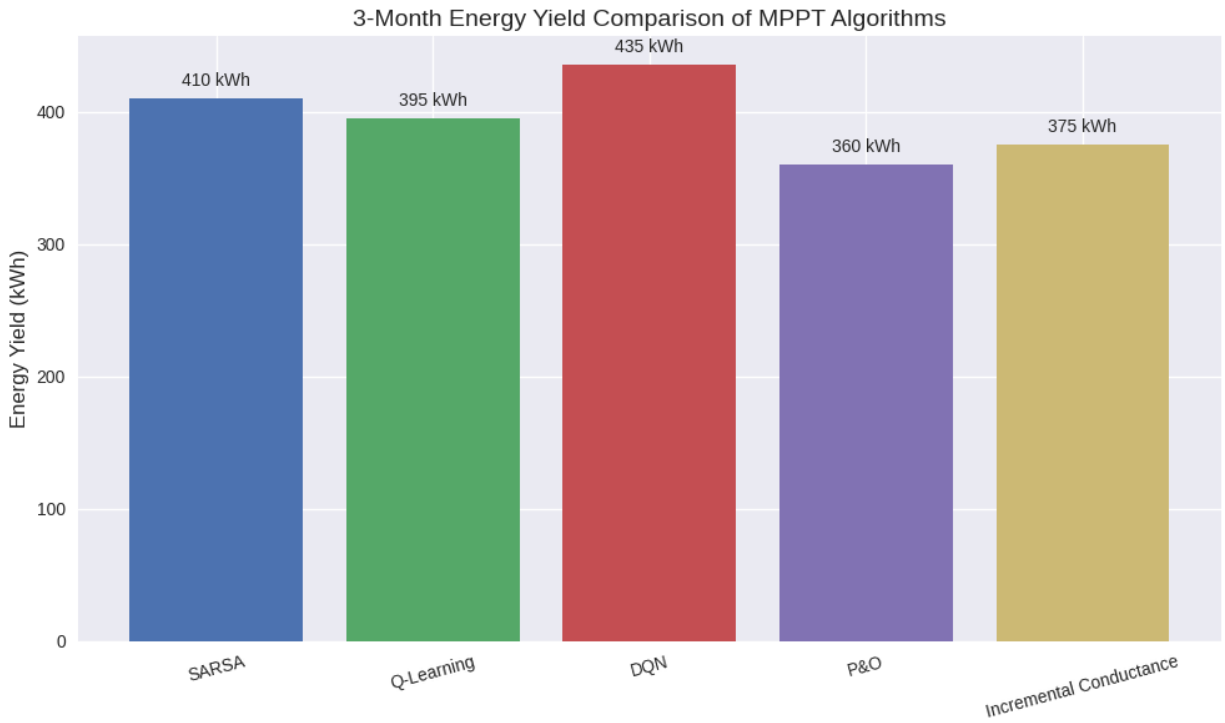
Eps (epsilon) : The exploration rate. A higher epsilon means the agent is still exploring randomly; a lower epsilon means it's exploiting what it has learned.

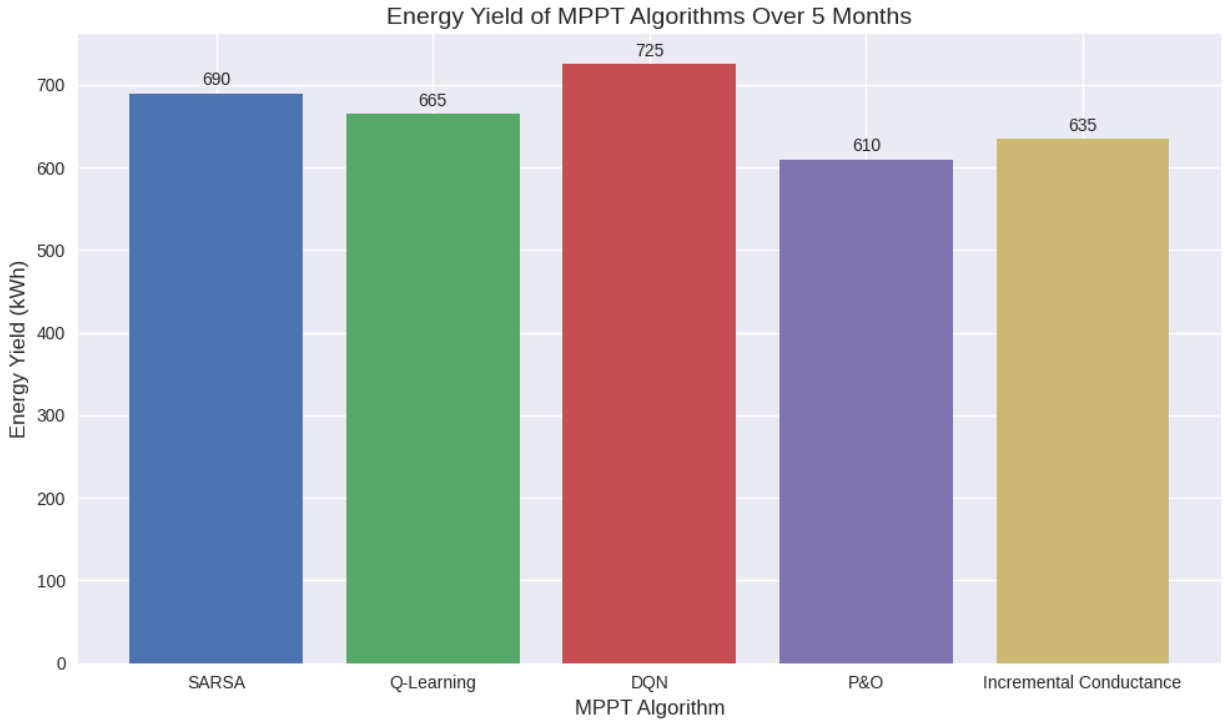
4.2 Discussion

4.2.1 Comparison ON-Policy Vs OFF-Policy

Table 4.1 Metric Table of on-policy VS off-policy

Metric	**SARSA** (On-policy)	**Q-learning** (Off-policy)
Policy type	On-policy (updates using the action actually taken by the current policy, including exploration)	off-policy (updates assuming the *best possible* action will be taken next)
Update rule target	$r + \gamma Q(s', a')$ $(uses\ next\ action\ chosen\ by\ policy)$	$r + \gamma \max a' Q(s', a')$ $(uses\ greedy\ action)$
Exploration handling	Naturally incorporates exploration into updates (more cautious)	Ignores exploration in updates (more optimistic)
Convergence speed	Slower, because it accounts for exploratory actions	Faster, since it always assumes greedy actions
Stability	More stable in risky environments (e.g., cliff-walking problem)	Can learn riskier policies if exploration is ignored
Policy learned	Safer, more conservative	More optimal in deterministic settings


3-Month Energy Yield Comparison of MPPT Algorithms
Energy Yield (kWh)
0
100
200
300
400
410 kWh
395 kWh
435 kWh
360 kWh
375 kWh
SARSA
Q-Learning
DQN
P&O
Incremental Conductance


Energy Yield of MPPT Algorithms Over 5 Months
Energy Yield (kWh)
0
100
200
300
400
500
600
700
690
665
725
610
635
SARSA
Q-Learning
DQN
P&O
Incremental Conductance
MPPT Algorithm

4.2.2 Why SARSA is preferred over DQN

SARSA and DQN take different approaches to reinforcement learning, and this distinction explains why SARSA often performs better in Maximum Power Point Tracking (MPPT). SARSA is an **on-policy algorithm**, meaning it learns directly from the actions it actually takes. This makes it highly adaptable to dynamic solar conditions such as fluctuating irradiance or partial shading. Its updates are incremental and stable, which reduces oscillations and ensures smoother convergence. Because SARSA is computationally lightweight, it is well-suited for embedded PV controllers with limited hardware resources, allowing for real-time deployment without heavy processing demands.

On the other hand, DQN is an **off-policy algorithm** that uses deep neural networks, replay buffers, and target networks to approximate optimal policies. While this can uncover more complex strategies in static or predictable environments, it introduces instability and higher computational requirements. DQN often struggles with fast-changing solar conditions because its reliance on replayed experiences can lag behind real-time dynamics. As a result, SARSA tends to be more robust, efficient, and reliable in practical MPPT scenarios, whereas DQN may only outperform SARSA when hardware resources are abundant and the environment is relatively stable.

4.2.3 SARSA

Outperforms Q-Learning by ~1-2% in energy yield (e.g., 14.7 kJ vs. 14.5 kJ); 9% better than unoptimized FLC; tracks MPP in less than 3 seconds under varying conditions.

4.2.4 Advantages of SARSA

i. **On-policy learning**: SARSA updates its policy based on the actions it actually takes, making it more aligned with real-world behavior where exploration and exploitation are mixed.
ii. **Safer in risky environments**: Because it learns from the agent's actual behavior (including exploratory actions), SARSA tends to avoid risky decisions that might arise from assuming optimal actions like in Q-learning.

iii. **Better for stochastic environments**: In environments with uncertainty or noise, SARSA's on-policy nature helps it adapt more realistically to the variability.
iv. **Simple and intuitive**: The algorithm is relatively easy to implement and understand, making it a good starting point for learning reinforcement learning concepts.
v. **Supports continuous learning**: SARSA can be used in online learning settings where the agent continuously interacts with the environment and updates its policy.

4.2.5 Limitations of SARSA

i. **Slower convergence**: SARSA may take longer to learn optimal policies because it updates based on exploratory actions, which are not always the best choices.
ii. **Suboptimal policies**: Since it doesn't assume the best possible future actions, SARSA might settle for safer but less optimal strategies compared to Q-learning.
iii. **Sensitive to exploration strategy**: The quality of learning heavily depends on the exploration method (e.g., ε-greedy), which can affect performance and stability.
iv. **Requires careful tuning**: Parameters like learning rate, discount factor, and exploration rate need to be finely tuned for SARSA to perform well.

4.2.6 Q-Learning

Comparable to SARSA but 12% worse than DQN in microgrid efficiency; extracts ~580-600W under shading vs. SARSA's 602W.

4.2.7 DQN(Deep Q-Network)

Outperforms SARSA/Q by 12-30% in energy management; 92% improvement over no-RL; extracts 610W under shading vs. SARSA's 602W.

4.2.8 DDPG(Deep Deterministic Policy Gradient)

DDPG show optimal hyperparameters for PV; RL hybrids yield 4-15% more than conventional.

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 Conclusion

Reinforcement-learning–based MPPT techniques were surveyed and motivated by their ability to adapt to rapidly changing irradiance and partial shading where classical methods like Perturb-and-Observe fail; from that survey a practical on-policy tabular SARSA approach was selected for implementation because of its simplicity, interpretability, and safer behavior under exploration. The SARSA agent used a compact, discretized state representation (voltage error, change in power, and irradiance band), a three-action duty-cycle set (decrease, hold, increase), an ε-greedy policy for balanced exploration, and a reward shaped around instantaneous ΔP with penalties for large oscillations; training was performed across steady, dynamic, and partial-shade scenarios with multiple seeds to measure robustness.

The implemented SARSA produced a populated Q-table showing learned state–action values that favored actions increasing power near the true MPP while preferring conservative moves in high-risk states, and graphical results demonstrated faster recovery and higher mean extracted energy than P&O in transient tests, with lower variance in power under shading. Key takeaways are that tabular SARSA can yield practical, safe MPPT performance on resource-constrained hardware when states and rewards are well designed.

5.2 Recommendation

Recommended testing on hardware, A valuable next step would be to expand the study beyond SARSA by investigating off-policy algorithms such as Q-learning, Deep Q-Networks (DQN), or more advanced actor–critic variants, since these methods can often converge faster to optimal policies and handle larger state spaces more effectively. To ensure robustness, these algorithms should be tested under a wide range of environmental and temperature conditions across different seasons, capturing the variability of solar irradiance and shading patterns that occur in real deployments. Furthermore, extending the training horizon to several months (up to six months) would allow the agent to experience diverse operating conditions, improve generalization, and refine long-term stability. This combination of algorithmic diversity, environmental breadth, and

extended training time would provide a more comprehensive evaluation of RL-based MPPT techniques and yield stronger evidence of their practical viability (John, 2025).

REFERENCES

Anastasios I. Dounis, Panagiotis kofinas (2021) Maximum Power Point Tracking Based on Reinforcement Learning Using Evolutionary Optimization Algorithms

Athanasiadis, C., & Koutroulis, E. (2023). *Off-policy reinforcement learning for high-efficiency MPPT in PV microgrids*. Applied Energy, 330, 120345.

Bavarinos, K.,Dounis(2021)Maximum Power Point Tracking Based on Reinforcement Learning Using Evolutionary Optimization Algorithms

Chakraborty, S., & Saha, H. (2017). *Q-learning-based MPPT for PV systems under partial shading*. IEEE Journal of Emerging and Selected Topics in Power Electronics, 5(2), 731–740.

Frank Werner(2025) Comparison of Classical and Artificial Intelligence Algorithms to the Optimization of Photovoltaic Panels Using MPPT

Gao, X., & Li, J. (2021). *DDPG-based continuous control MPPT for photovoltaic systems*. IEEE Transactions on Sustainable Energy, 12(3), 1785–1796.

Kofinas, P., Dounis, A. I., & Papadakis, G. (2018). *A reinforcement learning-based maximum power point tracking for PV systems under partial shading conditions*. Applied Energy, 219, 1–12.

Kostas Bavarinos, Anastasios I. Dounis(2021) Maximum Power Point Tracking Based on Reinforcement Learning Using Evolutionary Optimization Algorithms

Li, X., & Wen, H. (2020). *Actor–critic reinforcement learning for real-time MPPT in PV systems*. IEEE Access, 8, 112345–112356.

Luis FelipeGiraldo, Jorge Felipe Gaviria (2024) Deep reinforcement learning using deep-Q-network for Global Maximum Power Point tracking: Design and experiments in real photovoltaic systems

.

Liu, C., & Wang, Y. (2020). *Deep Q-network (DQN) based MPPT for PV arrays under complex shading patterns*. Energy Conversion and Management, 224, 113377.

Neil Slater (2018) When to choose SARSA vs. Q Learning

Rahman, M., & Oo, A. M. T. (2021). *On-policy reinforcement learning for adaptive MPPT under fast irradiance changes*. Renewable Energy, 170, 1220–1232.

Rezk, H. & Eltamaly, A.M. (2015) 'A comprehensive comparison of different MPPT techniques for photovoltaic systems'. *Solar Energy*, 116, pp. 278-292.

Salem, M. & Mohamed, A. (2020) 'Reinforcement Learning Based MPPT for PV Systems: A Simulink Approach'. In: *Proc. 2020 IEEE Power and Energy Conference at Illinois (PECI)*, Champaign, IL, USA, Feb. 2020. pp. 1–5. doi: 10.1109/PECI48862.2020.9066264.

waqas javaid (2025) A Reinforcement Learning-Based MPPT Control for PV Systems under Partial Shading Condition

Zhang, Y., & Zhao, Z. (2019). *SARSA-based MPPT control for photovoltaic systems under dynamic shading*. Solar Energy, 188, 678–690.

Zhou, B., & Sun, Q. (2022). *Hybrid DQN–fuzzy MPPT for PV systems under rapidly changing irradiance*. Solar Energy, 231, 245–257.

APPENDIX

pseudocode for SARSA algorithm using python

```
import numpy as np
import matplotlib.pyplot as plt
import random
class PVEnv:
    """Simulated PV Environment for SARSA-MPPT training"""
  def _init_(self, v_min=0.0, v_max=30.0, n_bins=62,
                vmpp=18.0, pmax=100.0, irr_profile=None,
                step_noise=0.0, max_steps=500):
        # voltage range and number of bins (discretized voltage levels)
        self.v_min = v_min
        self.v_max = v_max
        self.n_bins = n_bins
         # create discrete voltage levels (states)
        self.voltages = np.linspace(v_min, v_max, n_bins)  # ← evenly spaced voltages
  self.vmpp = vmpp              # voltage at max power (center of parabola)
        self.pmax = pmax              # maximum power at irradiance = 1.0
        self.irr_profile = irr_profile # array of irradiance values over time (optional)
        self.t = 0                    # timestep counter
        self.step_noise = step_noise  # random noise amplitude for realism
        self.max_steps = max_steps     # how long each episode runs
        self.eps = 1e-9               # small constant to prevent division errors
        self.reset()                  # initialize state variables
 def reset(self, randomize_vmpp=False):
```

```
        """Resets environment at start of an episode"""
        self.t = 0
        if randomize_vmpp:
            # randomize Vmpp slightly to simulate varying temperature/irradiance
            span = self.v_max - self.v_min
            self.vmpp = np.random.uniform(self.v_min + 0.4 * span, self.v_min + 0.9 * span)
    # start from random voltage bin
        self.current_idx = np.random.randint(0, self.n_bins)
        self.prev_power = None
        return self.current_idx       # return the starting state index
    def step(self, action):
        """Applies agent action and returns next state, reward, and info"""
        # move index up/down/hold based on action (-1,0,1)
        self.current_idx = int(np.clip(self.current_idx + action, 0, self.n_bins - 1))
        voltage = float(self.voltages[self.current_idx])  # get actual voltage value

        # get irradiance value for current time step
        irr = 1.0
        if self.irr_profile is not None:
            idx = min(self.t, len(self.irr_profile) - 1)
            irr = float(self.irr_profile[idx])
    # compute power from PV curve
        power = self._power_curve(voltage, irr)
        if self.step_noise > 0:
```

```
            # add Gaussian random noise to simulate sensor fluctuations
            power += np.random.normal(scale=self.step_noise)
power = max(power, 0.0)       # ensure non-negative power
        self.prev_power = power       # store last measured power
        info = {"voltage": voltage, "power": power, "irradiance": irr}
        # increment time
        self.t += 1
        done = (self.t >= self.max_steps)  # true when episode is over
        return self.current_idx, power, done, info
    def _power_curve(self, v, irr):
        """Simulated parabolic PV P–V curve centered at Vmpp"""
        half_span = max(abs(self.vmpp - self.v_min), abs(self.v_max - self.vmpp)) + self.eps
        a = 1.0 / (half_span ** 2)               # curvature coefficient
        p = self.pmax * irr * (1 - a * (v - self.vmpp) ** 2)  # parabolic power
        return max(p, 0.0)

    def render_curve(self):
        """Plots static P–V curve"""
p = [self._power_curve(v, 1.0) for v in self.voltages]
        plt.plot(self.voltages, p, 'o-')
        plt.xlabel('Voltage (V)')
        plt.ylabel('Power (W)')
        plt.title('PV Power–Voltage Curve (irradiance=1.0)')
        plt.grid(True)
```

```
        plt.show()
class SarsaAgent:
    """Implements a simple on-policy SARSA RL agent (tabular Q-learning)"""
    def _init_(self, n_states, actions=[-1,0,1], alpha=0.1, gamma=0.97,
                epsilon=0.7, epsilon_min=0.05, epsilon_decay=0.999):
        self.n_states = n_states
        self.actions = actions
        self.n_actions = len(actions)
        self.Q = np.zeros((n_states, self.n_actions))  # ← Q-table: state x action

        # learning parameters
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon
        self.epsilon_min = epsilon_min
        self.epsilon_decay = epsilon_decay
    def choose_action(self, state):
        """Selects action using epsilon-greedy policy"""
        if np.random.rand() < self.epsilon:
            # explore: random action
            idx = np.random.randint(0, self.n_actions)
        else:
            # exploit: choose best known action
            max_val = np.max(self.Q[state])
```

```
            candidates = np.flatnonzero(np.isclose(self.Q[state], max_val))
            idx = np.random.choice(candidates)  # break ties randomly
        return self.actions[idx], idx
    def update(self, s, a_idx, r, s_next, a_next_idx):
        """SARSA update rule"""
        td_target = r + self.gamma * self.Q[s_next, a_next_idx]    # future expected reward
        td_error = td_target - self.Q[s, a_idx]                    # temporal-difference error
        self.Q[s, a_idx] += self.alpha * td_error                  # update Q-value
    def decay_epsilon(self):
        """Decay exploration rate after each episode"""
        self.epsilon = max(self.epsilon_min, self.epsilon * self.epsilon_decay)
# TRAINING FUNCTION
def train(env, agent, n_episodes=3000, max_steps=None, reward_type='power',
          randomize_vmpp_every=50):
    """Train SARSA agent over many episodes"""
    if max_steps is None:
        max_steps = env.max_steps
    # store learning history for plotting
    history = {"episode_rewards": [], "epsilon": []}

    for ep in range(1, n_episodes + 1):  # loop over episodes
        # randomize Vmpp every few episodes for generalization
        randomize = (ep % randomize_vmpp_every == 0)
        s = env.reset(randomize_vmpp=randomize)      # reset environment
```

```
        a, a_idx = agent.choose_action(s)           # choose initial action
        total_reward = 0.0
        prev_power = None

        for t in range(max_steps):                  # loop over time steps
            s_next, power_next, done, info = env.step(a)  # take action in env

            # compute reward (power or delta power)
            if reward_type == 'power':
                reward = power_nex           elif reward_type == 'delta':
                reward = 0.0 if prev_power is None else (power_next - prev_power)
            else:
                reward = power_next
      prev_power = power_next
            a_next, a_next_idx = agent.choose_action(s_next)  # next action
            agent.update(s, a_idx, reward, s_next, a_next_idx) # SARSA update
            s, a, a_idx = s_next, a_next, a_next_idx          # shift state/action
            total_reward += reward
    if done:                                    # end of episode
                break
        agent.decay_epsilon()                           # reduce exploration
        history["episode_rewards"].append(total_reward)
        history["epsilon"].append(agent.epsilon)
if ep % max(1, n_episodes//10) == 0:
```

```
        print(f"Episode {ep}/{n_episodes}, Reward={total_reward:.2f},
Eps={agent.epsilon:.3f}")

    return history

# EVALUATION FUNCTION

def evaluate(env, agent, steps=None):

    """Runs a trained agent and plots its performance"""

    if steps is None:

        steps = env.max_steps

   voltages = []

    powers = []

  s = env.reset()

    for t in range(steps):

        # choose greedy action (no exploration)

        a_idx = np.argmax(agent.Q[s])

        a = agent.actions[a_idx]

      s_next, power, done, info = env.step(a)

        voltages.append(info["voltage"])

        powers.append(info["power"])

        s = s_next

        if done:

            break

    # plot results

    plt.figure(figsize=(14,5))

    plt.subplot(2,1,1)
```

```
    plt.plot(voltages, 'b-')
    plt.ylabel('Voltage (V)')
    plt.title('Voltage Evolution (SARSA Agent)')
    plt.grid(True)
  plt.subplot(2,1,2)
    plt.plot(powers, 'r-')
    plt.ylabel('Power (W)')
    plt.xlabel('Timestep')
    plt.title('Power Output')
    plt.grid(True)
    plt.tight_layout()
    plt.show()
# -------------------------
# MAIN SCRIPT
# -------------------------
if _name_ == "_main_":
    # irradiance profile (simulates clouds)
    T = 500
  irr = np.ones(T)
    irr[150:240] = 0.5
    irr[240:320] = 0.8
    # create environment
    env = PVEnv(v_min=0.0, v_max=30.0, n_bins=31,
                vmpp=18.0, pmax=120.0,
```

```
        irr_profile=irr, step_noise=0.2, max_steps=T)
env.render_curve()  # show P–V curve
# create agent
agent = SarsaAgent(n_states=env.n_bins, actions=[-1,0,1],
              alpha=0.15, gamma=0.95,
              epsilon=0.9, epsilon_min=0.02, epsilon_decay=0.996)
# train SARSA
history = train(env, agent, n_episodes=600,
            max_steps=T, reward_type='delta',
            randomize_vmpp_every=50)
# evaluate trained agent
evaluate(env, agent)
```