The Open Bioinformatics Journal
DOI: 10.2174/0118750362380098250416071630, 2025, 18, e18750362380098

ISSN: 1875-0362
1

RESEARCH ARTICLE **OPEN ACCESS**

Balancing Accuracy and Recall in Hebbian and Quantum-Inspired Learning Models

Theodoros Kyriazos[1,*] and Mary Poga[2]

[1]*Department of Psychology, Panteion University, Athens, Greece*
[2]*Independent Researcher, Athens, Greece*

* Address correspondence to this author at the Department of Psychology, Panteion University, Athens, Greece; E-mail: th.kyriazos@gmail.com

Cite as: Kyriazos T, Poga M. Balancing Accuracy and Recall in Hebbian and Quantum-Inspired Learning Models. Open Bioinform J, 2025; 18: e18750362380098. http://dx.doi.org/10.2174/0118750362380098250416071630

Published: April 25, 2025

1. APPENDIX

Fix for Deterministic Results

To ensure consistent results across runs, set a random seed at the start of your script using the set.seed() function. For example:

```
# Set seed for reproducibility
set.seed(12345)
```

Add this line near the beginning of the script, right after the library imports and before any operations involving randomness.

Impact on the Script

- **Random Weight Initialization:** This includes calls to rnorm() and runif() for weight and quantum state initialization.
- **Pattern Generation:** The generation of binary patterns and test patterns uses sample().
- **Noise Addition:** The distort_pattern() function uses sample() to add noise.

Setting a seed will fix these random operations to produce the same "random" values across runs, ensuring that your results are reproducible.

1.1. R code of this study

```
# Workspace Setup
# ==========================================================

# Clear existing plots
if (!is.null(dev.list())) dev.off()

# Clean workspace by removing all objects
rm(list = ls())

# Clear console (Note: This may not work in all environments)
cat("\014")

# Load necessary libraries
library(Matrix)
library(caret)
library(pROC)
library(knitr)
library(stats)

# Parameters
N <- 1000 # Number of neurons (fixed)
P <- 100 # Number of patterns (fixed)
num_instances <- 10 # Number of network instances for statistical analysis (fixed)
exc_ratio <- 0.8 # Proportion of excitatory neurons (fixed)
decay_rate <- 0.005 # Fixed decay rate
connectivity <- 0.1 # Fixed connectivity
```

```
# Varying parameters at three levels (low, medium, high)
learning_rates <- c(0.01, 0.05, 0.1)
threshold_levels <- c(0.3, 0.5, 0.7)

# Function to distort patterns for generalization testing
distort_pattern <- function(pattern, distortion_rate = 0.1)
{
noise <- sample(c(-1, 1), length(pattern), replace = TRUE,
prob = c(1 - distortion_rate, distortion_rate))
return(ifelse((pattern + noise) > 0, 1, 0))
}

# Define performance metrics functions
accuracy <- function(predicted, actual) {
sum(predicted == actual) / length(actual)
}

precision <- function(predicted, actual) {
true_positives <- sum(predicted == 1 & actual == 1)
predicted_positives <- sum(predicted == 1)
if (predicted_positives == 0) return(0)
true_positives / predicted_positives
}

recall <- function(predicted, actual) {
true_positives <- sum(predicted == 1 & actual == 1)
actual_positives <- sum(actual == 1)
if (actual_positives == 0) return(0)
true_positives / actual_positives
}

f1_score <- function(precision, recall) {
if (precision + recall == 0) return(0)
2 * (precision * recall) / (precision + recall)
}

# Function to perform statistical analysis
evaluate_model <- function(model, data) {
metrics <- list(accuracy = numeric(), precision =
numeric(), recall = numeric(), f1_score = numeric())
for (i in 1:nrow(data)) {
actual <- data[i, ]
test_input <- matrix(actual, nrow = 1)
recalled <- model(test_input)
model_accuracy <- accuracy(recalled, actual)
model_precision <- precision(recalled, actual)
model_recall <- recall(recalled, actual)
model_f1 <- f1_score(model_precision, model_recall)
metrics$accuracy <- c(metrics$accuracy, model_accuracy)
metrics$precision <- c(metrics$precision,
model_precision)
metrics$recall <- c(metrics$recall, model_recall)
metrics$f1_score <- c(metrics$f1_score, model_f1)
}
return(metrics)
}

# Function to calculate average metrics
avg_performance <- function(metrics) {
list(
accuracy = c(mean = mean(metrics$accuracy), sd =
sd(metrics$accuracy)),
precision = c(mean = mean(metrics$precision), sd =
sd(metrics$precision)),
recall = c(mean = mean(metrics$recall), sd =
sd(metrics$recall)),
f1_score = c(mean = mean(metrics$f1_score), sd =
sd(metrics$f1_score))
)
}

# Enhanced Hebbian learning function
hebbian_update <- function(weights, x, y, eta, n_exc,
n_inh) {
for (i in 1:length(x)) {
for (j in 1:length(y)) {
if (i <= n_exc && j <= n_exc) { # Excitatory to excitatory
weights[i, j] <- weights[i, j] + eta * x[i] * y[j]
} else if (i <= n_exc && j > n_exc) { # Excitatory to
inhibitory
weights[i, j] <- weights[i, j] - eta * x[i] * y[j]
} else if (i > n_exc && j <= n_exc) { # Inhibitory to
excitatory
weights[i, j] <- weights[i, j] - eta * x[i] * y[j]
} else { # Inhibitory to inhibitory
weights[i, j] <- weights[i, j] + eta * x[i] * y[j]
}
}
}
return(weights)
}

# Function to normalize the weights
normalize_weights <- function(weights) {
max_weight <- max(abs(weights))
if (max_weight > 0) {
weights <- weights / max_weight
}
return(weights)
}

# Function to calculate weight decay
decay_weights <- function(weights, decay_rate) {
return(weights * (1 - decay_rate))
}

# Quantum encoding function
quantum_encoding <- function(quantum_states, patterns,
alpha = 0.1) {
for (p in 1:nrow(patterns)) {
pattern <- patterns[p, ]
delta_q <- alpha * (outer(pattern, pattern) +
complex(real = rnorm(length(pattern)^2, 0, 1), imaginary
= rnorm(length(pattern)^2, 0, 1)))
quantum_states <- quantum_states + delta_q
# Normalize quantum states
norm <- sqrt(rowSums(Mod(quantum_states)^2))
quantum_states <- quantum_states / norm
}
return(quantum_states)
```

```
}

# Measure the time taken to execute the entire simulation
execution_time <- system.time({
# Iterate over different parameter levels
results <- list()
for (lr in learning_rates) {
for (threshold in threshold_levels) {
instance_results <- list(hebbian = list(), quantum = list())

for (instance in 1:num_instances) {
# Initialize weights matrix with small random values
weights <- matrix(rnorm(N * N, mean = 0, sd = 0.1), nrow = N, ncol = N)
weights <- weights * (matrix(runif(N * N) < connectivity, nrow = N, ncol = N))
diag(weights) <- 0 # No self-connections

# Initialize quantum states with small random complex values
Q_states <- matrix(complex(real = rnorm(N * N, 0, 1), imaginary = rnorm(N * N, 0, 1)), nrow = N, ncol = N)
Q_states <- Q_states * (matrix(runif(N * N) < connectivity, nrow = N, ncol = N))
diag(Q_states) <- 0 # No self-connections

# Determine number of excitatory and inhibitory neurons
n_exc <- round(N * exc_ratio)
n_inh <- N - n_exc

# Generate sparse binary patterns
patterns <- matrix(sample(c(0, 1), P * N, replace = TRUE, prob = c(0.9, 0.1)), nrow = P, ncol = N)

# Hebbian learning with normalization
hebbian_learning <- function(weights, patterns, learning_rate, decay, n_exc, n_inh) {
for (p in 1:nrow(patterns)) {
pattern <- patterns[p, ]
weights <- hebbian_update(weights, pattern, pattern, learning_rate, n_exc, n_inh)
weights <- decay_weights(weights, decay)
}
return(weights)
}

# Encode patterns using Hebbian learning and quantum encoding
weights <- hebbian_learning(weights, patterns, lr, decay_rate, n_exc, n_inh)
Q_states <- quantum_encoding(Q_states, patterns)

# Define the recall function for traditional Hebbian model
recall_pattern <- function(input_pattern, weights, N, threshold) {
state <- input_pattern
for (iter in 1:100) { # Iterate until convergence
for (i in 1:N) {
net_input <- sum(weights[i, ] * state) - threshold
state[i] <- ifelse(net_input > 0, 1, 0)
}
}
return(state)
}

# Define the recall function for quantum-inspired model
retrieve_quantum <- function(quantum_states, input) {
recalled_pattern <- Re(input %*% quantum_states)
recalled_pattern <- abs(recalled_pattern)
recalled_pattern[recalled_pattern >= 0.5] <- 1
recalled_pattern[recalled_pattern < 0.5] <- 0
return(recalled_pattern)
}

# Create distorted test patterns for generalization
test_patterns <- t(apply(patterns, 1, distort_pattern))

# Hebbian model evaluation
evaluate_hebbian <- function(test_input) {
recall_pattern(test_input, weights, N, threshold)
}

# Quantum model evaluation
evaluate_quantum <- function(test_input) {
retrieve_quantum(Q_states, test_input)
}

hebbian_metrics <- evaluate_model(evaluate_hebbian, test_patterns)
quantum_metrics <- evaluate_model(evaluate_quantum, test_patterns)

instance_results$hebbian[[instance]] <- hebbian_metrics
instance_results$quantum[[instance]] <- quantum_metrics
}

# Aggregate results over multiple instances
hebbian_metrics_agg <- list(
accuracy = unlist(lapply(instance_results$hebbian, function(x) x$accuracy)),
precision = unlist(lapply(instance_results$hebbian, function(x) x$precision)),
recall = unlist(lapply(instance_results$hebbian, function(x) x$recall)),
f1_score = unlist(lapply(instance_results$hebbian, function(x) x$f1_score))
)

quantum_metrics_agg <- list(
accuracy = unlist(lapply(instance_results$quantum, function(x) x$accuracy)),
precision = unlist(lapply(instance_results$quantum, function(x) x$precision)),
recall = unlist(lapply(instance_results$quantum, function(x) x$recall)),
f1_score = unlist(lapply(instance_results$quantum, function(x) x$f1_score))
)

# Calculate average metrics for both models
```

```
hebbian_avg_metrics <- avg_performance(hebbian_metrics_agg)
quantum_avg_metrics <- avg_performance(quantum_metrics_agg)

# Store results for comparison
results[[paste("LR:", lr, "Thresh:", threshold)]] <- list(
Hebbian = hebbian_avg_metrics,
Quantum = quantum_avg_metrics,
Hebbian_Detailed = instance_results$hebbian,
Quantum_Detailed = instance_results$quantum
)
}
}
})

# Print the execution time
print(execution_time)

# Function to print results in a table format
print_results_table <- function(results) {
for (key in names(results)) {
cat("\nParameters: ", key, "\n")
hebbian_metrics <- results[[key]]$Hebbian
quantum_metrics <- results[[key]]$Quantum
data <- data.frame(
Metric = c("Accuracy", "Precision", "Recall", "F1-Score"),
Hebbian_Mean = c(hebbian_metrics$accuracy["mean"],
hebbian_metrics$precision["mean"],
hebbian_metrics$recall["mean"],
hebbian_metrics$f1_score["mean"]),
Hebbian_SD = c(hebbian_metrics$accuracy["sd"],
hebbian_metrics$precision["sd"],
hebbian_metrics$recall["sd"],
hebbian_metrics$f1_score["sd"]),
Quantum_Mean = c(quantum_metrics$accuracy["mean"],
quantum_metrics$precision["mean"],
quantum_metrics$recall["mean"],
quantum_metrics$f1_score["mean"]),
Quantum_SD = c(quantum_metrics$accuracy["sd"],
quantum_metrics$precision["sd"],
quantum_metrics$recall["sd"],
quantum_metrics$f1_score["sd"])
)
print(kable(data, format = "markdown"))
}
}

# Print the comparison tables
print_results_table(results)

# Function to perform paired t-tests
perform_statistical_tests <- function(results) {
for (key in names(results)) {
cat("\nParameters: ", key, "\n")
hebbian_detailed <- results[[key]]$Hebbian_Detailed
quantum_detailed <- results[[key]]$Quantum_Detailed

hebbian_accuracy <- unlist(lapply(hebbian_detailed, function(x) x$accuracy))
quantum_accuracy <- unlist(lapply(quantum_detailed, function(x) x$accuracy))

hebbian_precision <- unlist(lapply(hebbian_detailed, function(x) x$precision))
quantum_precision <- unlist(lapply(quantum_detailed, function(x) x$precision))

hebbian_recall <- unlist(lapply(hebbian_detailed, function(x) x$recall))
quantum_recall <- unlist(lapply(quantum_detailed, function(x) x$recall))

hebbian_f1 <- unlist(lapply(hebbian_detailed, function(x) x$f1_score))
quantum_f1 <- unlist(lapply(quantum_detailed, function(x) x$f1_score))

# Perform paired t-tests
accuracy_test <- t.test(hebbian_accuracy, quantum_accuracy, paired = TRUE)
precision_test <- t.test(hebbian_precision, quantum_precision, paired = TRUE)
recall_test <- t.test(hebbian_recall, quantum_recall, paired = TRUE)
f1_test <- t.test(hebbian_f1, quantum_f1, paired = TRUE)

cat("Paired t-test results:\n")
cat("Accuracy: p-value =", accuracy_test$p.value, "\n")
cat("Precision: p-value =", precision_test$p.value, "\n")
cat("Recall: p-value =", recall_test$p.value, "\n")
cat("F1-Score: p-value =", f1_test$p.value, "\n")

# Interpret results
cat("Interpretation:\n")
if (accuracy_test$p.value < 0.05) {
cat("There is a significant difference in accuracy between the Hebbian and Quantum models.\n")
} else {
cat("There is no significant difference in accuracy between the Hebbian and Quantum models.\n")
}

if (precision_test$p.value < 0.05) {
cat("There is a significant difference in precision between the Hebbian and Quantum models.\n")
} else {
cat("There is no significant difference in precision between the Hebbian and Quantum models.\n")
}

if (recall_test$p.value < 0.05) {
cat("There is a significant difference in recall between the Hebbian and Quantum models.\n")
} else {
cat("There is no significant difference in recall between the Hebbian and Quantum models.\n")
}

if (f1_test$p.value < 0.05) {
```

```
cat("There is a significant difference in F1-Score between the Hebbian and Quantum models.\n")
} else {
cat("There is no significant difference in F1-Score between the Hebbian and Quantum models.\n")
}
}
}

# Perform statistical tests
perform_statistical_tests(results)

# Function to interpret and summarize results
interpret_results <- function(results) {
for (key in names(results)) {
cat("\nParameters: ", key, "\n")
hebbian_metrics <- results[[key]]$Hebbian
quantum_metrics <- results[[key]]$Quantum

cat("Summary of Results:\n")

if (hebbian_metrics$accuracy["mean"] > quantum_metrics$accuracy["mean"]) {
cat("Hebbian model has higher accuracy.\n")
} else {
cat("Quantum model has higher accuracy.\n")
}

if (hebbian_metrics$precision["mean"] > quantum_metrics$precision["mean"]) {
cat("Hebbian model has higher precision.\n")
} else {
cat("Quantum model has higher precision.\n")
}

if (hebbian_metrics$recall["mean"] > quantum_metrics$recall["mean"]) {
cat("Hebbian model has higher recall.\n")
} else {
cat("Quantum model has higher recall.\n")
}

if (hebbian_metrics$f1_score["mean"] > quantum_metrics$f1_score["mean"]) {
cat("Hebbian model has higher F1-Score.\n")
} else {
cat("Quantum model has higher F1-Score.\n")
}

# Determine overall better model
overall_better_model <- ifelse(
sum(c(hebbian_metrics$accuracy["mean"],
hebbian_metrics$precision["mean"],
hebbian_metrics$recall["mean"],
hebbian_metrics$f1_score["mean"])) >
sum(c(quantum_metrics$accuracy["mean"],
quantum_metrics$precision["mean"],
quantum_metrics$recall["mean"],
quantum_metrics$f1_score["mean"])),
"Hebbian Model",
"Quantum Model"
)

cat("Overall, the better model is: ", overall_better_model, "\n")
}
}

# Interpret and summarize results
interpret_results(results)
```

1.2. R code Plots

```
    #
#####################################
# 1. Workspace Setup
#
#####################################

# Clear existing plots
if (!is.null(dev.list())) dev.off()

# Clean workspace by removing all objects
rm(list = ls())

# Clear console (Note: This may not work in all environments)
cat("\014")

#
#####################################
# 2. Install and Load Required Libraries
#
#####################################

# Define the required packages
required_packages <- c("ggplot2", "dplyr", "DiagrammeR", "tidyr")

# Install any missing packages
installed_packages <- rownames(installed.packages())
for (pkg in required_packages) {
if (!(pkg %in% installed_packages)) {
install.packages(pkg, dependencies = TRUE)
}
}

# Load the libraries
library(ggplot2)
library(dplyr)
library(DiagrammeR)
library(tidyr)

#
#####################################
```

```
# 3. Hebbian Neural Network Architecture Diagram
#
==================================
================

# Define the Hebbian Neural Network Diagram using DOT
language
heb_diagram <- "
digraph HebbianNN {
rankdir=LR;

label = 'Hebbian Neural Network Architecture';
labelloc = t;
fontsize = 20;

node [shape=circle, style=filled, color=lightblue]
Input1 [label='Input 1']
Input2 [label='Input 2']
Input3 [label='Input 3']
Input4 [label='Input 4']

node [shape=circle, style=filled, color=lightgreen]
Excitatory1 [label='Excitatory 1']
Excitatory2 [label='Excitatory 2']
Excitatory3 [label='Excitatory 3']
Excitatory4 [label='Excitatory 4']

node [shape=circle, style=filled, color=yellow]
Inhibitory1 [label='Inhibitory 1']
Inhibitory2 [label='Inhibitory 2']

node [shape=circle, style=filled, color=lightcoral]
Output1 [label='Output 1']
Output2 [label='Output 2']

# Input to Excitatory connections
Input1 -> Excitatory1
Input1 -> Excitatory2
Input1 -> Excitatory3
Input1 -> Excitatory4
Input2 -> Excitatory1
Input2 -> Excitatory2
Input2 -> Excitatory3
Input2 -> Excitatory4
Input3 -> Excitatory1
Input3 -> Excitatory2
Input3 -> Excitatory3
Input3 -> Excitatory4
Input4 -> Excitatory1
Input4 -> Excitatory2
Input4 -> Excitatory3
Input4 -> Excitatory4

# Input to Inhibitory connections
Input1 -> Inhibitory1
Input1 -> Inhibitory2
Input2 -> Inhibitory1
Input2 -> Inhibitory2
Input3 -> Inhibitory1
Input3 -> Inhibitory2
Input4 -> Inhibitory1
Input4 -> Inhibitory2

# Excitatory to Output connections
Excitatory1 -> Output1
Excitatory1 -> Output2
Excitatory2 -> Output1
Excitatory2 -> Output2
Excitatory3 -> Output1
Excitatory3 -> Output2
Excitatory4 -> Output1
Excitatory4 -> Output2

# Inhibitory to Output connections
Inhibitory1 -> Output1
Inhibitory1 -> Output2
Inhibitory2 -> Output1
Inhibitory2 -> Output2
}
"

# Render the Hebbian Neural Network Diagram
DiagrammeR::grViz(heb_diagram)

#
==================================
================
# 4. Quantum-Inspired Neural Network Architecture
Diagram
#
==================================
================

# Define the Quantum-Inspired Neural Network Diagram
using DOT language
quantum_diagram <- "
digraph QuantumNN {
rankdir=LR;

label = 'Quantum-Inspired Neural Network Architecture';
labelloc = t;
fontsize = 20;

node [shape=circle, style=filled, color=lightblue]
QInput1 [label='Q Input 1']
QInput2 [label='Q Input 2']
QInput3 [label='Q Input 3']
QInput4 [label='Q Input 4']

node [shape=circle, style=filled, color=lightgreen]
QState1 [label='Q State 1']
QState2 [label='Q State 2']
QState3 [label='Q State 3']
QState4 [label='Q State 4']

node [shape=circle, style=filled, color=lightcoral]
QOutput1 [label='Q Output 1']
QOutput2 [label='Q Output 2']

# Input to Quantum State connections
```

```
QInput1 -> QState1
QInput1 -> QState2
QInput1 -> QState3
QInput1 -> QState4
QInput2 -> QState1
QInput2 -> QState2
QInput2 -> QState3
QInput2 -> QState4
QInput3 -> QState1
QInput3 -> QState2
QInput3 -> QState3
QInput3 -> QState4
QInput4 -> QState1
QInput4 -> QState2
QInput4 -> QState3
QInput4 -> QState4

# Quantum State to Output connections
QState1 -> QOutput1
QState1 -> QOutput2
QState2 -> QOutput1
QState2 -> QOutput2
QState3 -> QOutput1
QState3 -> QOutput2
QState4 -> QOutput1
QState4 -> QOutput2
}
"

# Render the Quantum-Inspired Neural Network Diagram
DiagrammeR::grViz(quantum_diagram)

#
==========================================
================
# 5. Data Preparation
#
==========================================
================

# Create the 'results' data frame with the correct data
results <- data.frame(
Learning_Rate = rep(c(0.01, 0.01, 0.01, 0.05, 0.05, 0.05,
0.1, 0.1, 0.1), each = 2),
Threshold = rep(c(0.3, 0.5, 0.7, 0.3, 0.5, 0.7, 0.3, 0.5, 0.7),
each = 2),
Model = rep(c("Hebbian", "Quantum"), 9),
Accuracy = c(0.260, 0.872, 0.261, 0.874, 0.318, 0.871,
0.261, 0.872,
0.260, 0.873, 0.260, 0.873, 0.260, 0.873, 0.261, 0.871,
0.260, 0.873),
Precision = c(0.100, 0.102, 0.099, 0.106, 0.093, 0.103,
0.101, 0.104,
0.100, 0.101, 0.100, 0.103, 0.101, 0.106, 0.101, 0.103,
0.101, 0.106),
Recall = c(0.799, 0.037, 0.798, 0.036, 0.735, 0.037, 0.802,
0.037,
0.801, 0.035, 0.799, 0.035, 0.801, 0.037, 0.801, 0.038,
0.799, 0.038),
F1_Score = c(0.178, 0.053, 0.176, 0.052, 0.165, 0.054,
0.179, 0.054,
0.178, 0.052, 0.177, 0.052, 0.179, 0.054, 0.179, 0.055,
0.178, 0.055)
)

#
==========================================
================
# 6. Visualization: Bar Charts for Metrics
#
==========================================
================

# Define a function to create and save bar charts for a
given metric
create_and_save_plot <- function(data, metric, title,
filename) {
# Check if the metric exists in the data frame
if (!metric %in% colnames(data)) {
stop(paste("Metric", metric, "not found in the data
frame."))
}

# Create the bar chart
p <- ggplot(data, aes(x = factor(Threshold), y =
.data[[metric]], fill = Model)) +
geom_bar(stat = "identity", position =
position_dodge(width = 0.8),
width = 0.7, color = "black") +
facet_wrap(~ Learning_Rate, labeller = label_both) +
labs(title = title, x = "Threshold", y = metric) +
scale_fill_manual(values = c("Hebbian" = "white",
"Quantum" = "black")) +
theme_minimal() +
theme(
text = element_text(size = 15),
strip.text = element_text(size = 15), # Facet labels
plot.title = element_text(size = 18, hjust = 0.5), #
Centered title
legend.title = element_blank(), # Remove legend title
legend.text = element_text(size = 16), # Legend text size
axis.title = element_text(size = 16), # Axis titles
axis.text = element_text(size = 15) # Axis text
)

# To ensure white bars are visible, set a light gray
background
p <- p + theme(
panel.background = element_rect(fill = "grey90", color =
NA),
plot.background = element_rect(fill = "grey95", color =
NA)
)

# Save the plot as a high-resolution PNG file
ggsave(filename, plot = p, width = 8.5, height = 11, dpi =
500, units = "in", device = "png")

# Optionally, display the plot in the R session
print(p)
```

```
}

# Define metrics and corresponding filenames
metrics_info <- list(
list(metric = "Accuracy", title = "Accuracy Comparison for Hebbian and Quantum-Inspired Models", filename = "Figure3_Accuracy.png"),
list(metric = "Precision", title = "Precision Comparison for Hebbian and Quantum-Inspired Models", filename = "Figure4_Precision.png"),
list(metric = "Recall", title = "Recall Comparison for Hebbian and Quantum-Inspired Models", filename = "Figure5_Recall.png"),
list(metric = "F1_Score", title = "F1-Score Comparison for Hebbian and Quantum-Inspired Models", filename = "Figure6_F1_Score.png")
)

# Iterate through each metric and create/save the corresponding plot
for (info in metrics_info) {
create_and_save_plot(results, info$metric, info$title, info$filename)
}

#
==========
# End of Script
#
==========
```