Comprehensive MD Simulation Analysis

A D A Shahinuzzaman

November 24, 2025

Contents

1 Full MD Simulation Analysis

All plots will be generated in sequence and placed immediately after their producing code.

```
######################################################################
# COMPREHENSIVE MOLECULAR DYNAMICS ANALYSIS SCRIPT
# Merged Version: Includes ALL 5 Parts
#
# Parts Included:
# 1. RMSF Analysis (Root Mean Square Fluctuation)
# 2. Secondary Structure Analysis
# 3. Residue Contacts Analysis
# 4. DCCM/PCA Analysis (Dynamic Cross-Correlation & Principal Component)
# 5. Time Evolution Analysis (RMSD, Energy, etc.)
#
# Author: A D A SHAHINUZZAMAN
# Purpose: Complete MD simulation analysis from multiple data files
#
# Input files required:
# 1. "Samip_rmsf.tab" (RMSF data)
# 2. "Samip_plotres_secstrMolA.tab" (Secondary Structure data)
# 3. "Samip_plotres_conMolA.tab" (Residue Contacts data)
# 4. "Samip_dccm.tab" (DCCM data)
# 5. "Samip_analysis.tab" (Time evolution data)
#
# Output: Multiple folders with publication-ready plots and analysis
######################################################################

# === GLOBAL SETUP (STEP 1) ==========================================
cat("=== COMPREHENSIVE MD ANALYSIS SCRIPT ===\n")
```

```
## === COMPREHENSIVE MD ANALYSIS SCRIPT ===
```

```r
cat("--- STEP 1: Setting up environment and loading packages ---\n")
```

```
## --- STEP 1: Setting up environment and loading packages ---
```

```r
# Set the working directory (!!! CHANGE THIS PATH !!!)
setwd("C:/Users/SHAHIN/Desktop/SAMIP")

# Install and load required libraries
required_packages <- c("tidyverse", "zoo", "patchwork", "reshape2", "viridis", "lattice", "gri
new_packages <- required_packages[!(required_packages %in% installed.packages()[,"Package"])]
if(length(new_packages)) install.packages(new_packages)

# Load all libraries
library(tidyverse)  # Includes ggplot2, dplyr, readr, stringr
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.6
## v forcats   1.0.1     v stringr   1.6.0
## v ggplot2   4.0.1     v tibble    3.3.0
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.2.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to bec
```

```r
library(zoo)        # For moving average (RMSF)
```

```
##
## Attaching package: 'zoo'
##
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```r
library(patchwork)  # For combining plots (RMSF)
library(reshape2)   # For reshaping data (SS, Contacts, DCCM)
```

```
##
## Attaching package: 'reshape2'
##
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```r
library(viridis)    # For color scales (SS)
```

```
## Loading required package: viridisLite
```

```r
library(lattice)    # For initial DCCM heatmap
library(gridExtra)  # For arranging plots
```

```
##
## Attaching package: 'gridExtra'
##
## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
# Define output directories and create them
rmsf_output_dir <- getwd()
ss_output_dir <- "ss_plots"
contact_output_dir <- "protein_stability_plots"
dccm_output_dir <- "Heatmap"
time_output_dir <- "Time_Evolution_Plots"

if (!dir.exists(ss_output_dir)) dir.create(ss_output_dir)
if (!dir.exists(contact_output_dir)) dir.create(contact_output_dir)
if (!dir.exists(dccm_output_dir)) dir.create(dccm_output_dir)
if (!dir.exists(time_output_dir)) dir.create(time_output_dir)

cat("Setup complete. Working directory set to:", getwd(), "\n")
```

```
## Setup complete. Working directory set to: C:/Users/SHAHIN/Desktop/SAMIP
```

```r
cat("Output folders created:\n")
```

```
## Output folders created:
```

```r
cat("  - 'ss_plots' (Secondary Structure)\n")
```

```
##   - 'ss_plots' (Secondary Structure)
```

```r
cat("  - 'protein_stability_plots' (Residue Contacts)\n")
```

```
##   - 'protein_stability_plots' (Residue Contacts)
```

```r
cat("  - 'Heatmap' (DCCM/PCA)\n")
```

```
##   - 'Heatmap' (DCCM/PCA)
```

```r
cat("  - 'Time_Evolution_Plots' (Time-dependent analysis)\n\n")
```

```
##   - 'Time_Evolution_Plots' (Time-dependent analysis)
```

```r
# ===================================================================
# === PART 1: RMSF (Root Mean Square Fluctuation) Analysis ==========
# ===================================================================
cat("--- PART 1: RMSF Analysis ---\n")
```

```
## --- PART 1: RMSF Analysis ---
```

```r
# FUNCTION 1: Read and process the RMSF data file
read_rmsf_data <- function(filename = "Samip_rmsf.tab") {
  cat("Step 1: Reading data file:", filename, "\n")
```

```r
  # Read all lines from the file
  file_lines <- readLines(filename)

  # Remove header lines and empty lines (keep only data lines)
  data_lines <- file_lines[!str_detect(file_lines, "^Table|^\\[|^$")]

  # Convert text lines into a data table
  data_table <- map_dfr(data_lines, function(line) {
    # Split each line into separate pieces
    line_parts <- str_split(str_trim(line), "\\s+")[[1]]

    # Only process lines that have all the data we need
    if (length(line_parts) >= 6) {
      data.frame(
        atom_number = as.numeric(line_parts[1]),
        atom_name = line_parts[2],
        residue_name = line_parts[3],
        residue_number = as.numeric(line_parts[4]),
        chain = line_parts[5],
        rmsf = as.numeric(line_parts[6]),
        stringsAsFactors = FALSE
      )
    } else {
      NULL  # Skip lines that don't have enough data
    }
  })

  cat("  Successfully processed", nrow(data_table), "atom records\n")
  return(data_table)
}

# FUNCTION 2: Calculate average RMSF for each residue
calculate_residue_averages <- function(atom_data) {
  cat("Step 2: Calculating residue averages...\n")

  # Group data by residue and calculate different types of averages
  residue_data <- atom_data %>%
    group_by(residue_number, residue_name) %>%
    summarise(
      # Average for backbone atoms (main protein chain)
      backbone_avg = mean(rmsf[atom_name %in% c("N", "CA", "C", "O")], na.rm = TRUE),

      # RMSF specifically for CA atoms (most important for backbone)
      ca_rmsf = ifelse(any(atom_name == "CA"), rmsf[atom_name == "CA"], NA),

      # Average for side chain atoms (the parts that stick out)
      side_chain_avg = mean(rmsf[!atom_name %in% c("N", "CA", "C", "O", "H", "HA", "HB", "HD",
```

```r
      # Average for all atoms in the residue
      all_atoms_avg = mean(rmsf, na.rm = TRUE),
      .groups = 'drop'
    ) %>%
    filter(!is.na(ca_rmsf))  # Remove residues that don't have CA atoms

  cat("  Processed", nrow(residue_data), "residues\n")
  return(residue_data)
}

# FUNCTION 3: Create the main set of plots
create_main_plots <- function(residue_data) {
  cat("Step 3: Creating visualizations...\n")

  # Define common theme elements for all plots - NO GRID BOX
  bold_theme <- theme(
    plot.title = element_text(face = "bold", size = 16, hjust = 0.5),
    axis.title.x = element_text(face = "bold", size = 14),
    axis.title.y = element_text(face = "bold", size = 14),
    axis.text.x = element_text(face = "bold", size = 12, color = "black"),
    axis.text.y = element_text(face = "bold", size = 12, color = "black"),
    legend.title = element_text(face = "bold", size = 13),
    legend.text = element_text(face = "bold", size = 11),
    # REMOVED GRID LINES
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.background = element_blank(),
    axis.line = element_line(color = "black", linewidth = 0.5)
  )

  # PLOT 1: Backbone flexibility (main plot)
  plot_backbone <- ggplot(residue_data, aes(x = residue_number, y = ca_rmsf)) +
    geom_line(color = "blue", linewidth = 1, alpha = 0.8) +
    geom_ribbon(aes(ymin = 0, ymax = ca_rmsf), fill = "blue", alpha = 0.2) +
    labs(
      title = "A) Backbone Flexibility (CA Atoms)",
      x = "Residue Number",
      y = "RMSF (Å)"
    ) +
    theme_minimal() +
    bold_theme

  # PLOT 2: Compare backbone vs side chain flexibility
  # Prepare data for comparison plot
  comparison_data <- residue_data %>%
    select(residue_number, backbone_avg, side_chain_avg) %>%
    pivot_longer(cols = c(backbone_avg, side_chain_avg),
```

```r
                names_to = "atom_group", values_to = "rmsf")

plot_comparison <- ggplot(comparison_data, aes(x = residue_number, y = rmsf, color = atom_gro
  geom_line(linewidth = 1) +
  scale_color_manual(
    values = c("backbone_avg" = "darkgreen", "side_chain_avg" = "red"),
    labels = c("Backbone (N,CA,C,O)", "Side Chains")
  ) +
  labs(
    title = "B) Backbone vs Side Chain Flexibility",
    x = "Residue Number",
    y = "RMSF (Å)",
    color = "Atom Group"
  ) +
  theme_minimal() +
  bold_theme +
  theme(legend.position = "top")

# PLOT 3: Distribution of flexibility values
plot_distribution <- ggplot(residue_data, aes(x = ca_rmsf)) +
  geom_histogram(bins = 30, fill = "purple", alpha = 0.7, color = "black") +
  geom_vline(aes(xintercept = mean(ca_rmsf)),
             color = "red", linetype = "dashed", linewidth = 1) +
  annotate("text", x = mean(residue_data$ca_rmsf), y = Inf,
           label = paste("Mean:", round(mean(residue_data$ca_rmsf), 2), "Å"),
           vjust = 2, hjust = -0.1, color = "red", size = 5, fontface = "bold") +
  labs(
    title = "C) Distribution of Backbone Flexibility",
    x = "RMSF (Å)",
    y = "Frequency"
  ) +
  theme_minimal() +
  bold_theme

# PLOT 4: Smooth trend using moving average
residue_data$moving_avg <- rollmean(residue_data$ca_rmsf, k = 5, fill = NA)

plot_trend <- ggplot(residue_data, aes(x = residue_number)) +
  geom_line(aes(y = ca_rmsf), color = "blue", alpha = 0.3, linewidth = 0.8) +
  geom_line(aes(y = moving_avg), color = "darkblue", linewidth = 1.2) +
  labs(
    title = "D) Flexibility Trends (5-residue Moving Avg.)",
    x = "Residue Number",
    y = "RMSF (Å)"
  ) +
  theme_minimal() +
  bold_theme
```

```
  # Combine all 4 plots into one figure
  combined_figure <- (plot_backbone + plot_comparison) / (plot_distribution + plot_trend) +
    plot_annotation(
      title = "MD Simulation RMSF Analysis - Protein Stability Assessment",
      subtitle = paste("Residues:", min(residue_data$residue_number), "-", max(residue_data$res
      theme = theme(
        plot.title = element_text(face = "bold", size = 20, hjust = 0.5),
        plot.subtitle = element_text(size = 16, hjust = 0.5, face = "bold")
      )
    )

  # Save the combined figure as TIFF with 300 DPI
  ggsave("rmsf_analysis_comprehensive.tiff", combined_figure,
         width = 16, height = 12, dpi = 300, device = "tiff")

  cat("  Saved: rmsf_analysis_comprehensive.tiff (300 DPI)\n")
  return(combined_figure)
}

# FUNCTION 4: Create detailed flexibility plot with stability regions (UPDATED - NO GRID BOX)
create_detailed_plot <- function(residue_data) {
  # Define theme for detailed plot - REMOVED GRID LINES
  detailed_theme <- theme(
    plot.title = element_text(face = "bold", size = 18, hjust = 0.5),
    plot.subtitle = element_text(size = 14, hjust = 0.5, face = "bold"),
    axis.title.x = element_text(face = "bold", size = 16),
    axis.title.y = element_text(face = "bold", size = 16),
    axis.text.x = element_text(face = "bold", size = 14, color = "black"),
    axis.text.y = element_text(face = "bold", size = 14, color = "black"),
    # REMOVED GRID LINES - no panel.grid.major or panel.grid.minor
    panel.background = element_blank(),  # Clean white background
    axis.line = element_line(color = "black", linewidth = 0.5)  # Keep axis lines
  )

  # Calculate offset position for "Very Stable" text (3mm to the right)
  x_offset <- (max(residue_data$residue_number) - min(residue_data$residue_number)) * 0.03

  detailed_plot <- ggplot(residue_data, aes(x = residue_number, y = ca_rmsf)) +
    geom_line(color = "navy", linewidth = 1.2) +

    # Add reference lines for stability thresholds
    geom_hline(yintercept = 1.5, color = "green", linetype = "dashed", alpha = 0.7, linewidth =
    geom_hline(yintercept = 3.0, color = "red", linetype = "dashed", alpha = 0.7, linewidth = 1

    # Color-code flexible regions (red) and stable regions (green)
    geom_ribbon(aes(ymin = 0, ymax = ifelse(ca_rmsf > 3.0, ca_rmsf, 0)),
                fill = "red", alpha = 0.2) +
```

```
    geom_ribbon(aes(ymin = 0, ymax = ifelse(ca_rmsf < 1.5, ca_rmsf, 0)),
                fill = "green", alpha = 0.2) +

    # Add labels for stability regions with "Very Stable" offset to the right
    annotate("text", x = max(residue_data$residue_number) * 0.02 + x_offset, y = 1.2,
             label = "Very Stable", color = "darkgreen", size = 6, fontface = "bold") +
    annotate("text", x = max(residue_data$residue_number) * 0.02, y = 2.2,
             label = "Stable", color = "darkorange", size = 6, fontface = "bold") +
    annotate("text", x = max(residue_data$residue_number) * 0.02, y = 3.5,
             label = "Flexible", color = "darkred", size = 6, fontface = "bold") +

    labs(
      title = "Detailed Protein Flexibility Analysis by RMSF",
      subtitle = "Green = Very Stable (<1.5 Å), Orange = Stable (1.5-3.0 Å), Red = Flexible (>3
      x = "Residue Number",
      y = "RMSF (Å)"
    ) +
    theme_minimal() +
    detailed_theme

  # Save as TIFF with 300 DPI
  ggsave("detailed_flexibility_analysis.tiff", detailed_plot,
         width = 14, height = 6, dpi = 300, device = "tiff")

  cat("  Saved: detailed_flexibility_analysis.tiff (300 DPI)\n")
  return(detailed_plot)
}

# FUNCTION 5: Generate a detailed text report
generate_analysis_report <- function(residue_data) {
  ca_rmsf <- residue_data$ca_rmsf

  cat("\n")
  cat("=", strrep("=", 60), "\n", sep = "")
  cat("PROTEIN STABILITY ANALYSIS REPORT\n")
  cat("File: Samip_rmsf.tab\n")
  cat("=", strrep("=", 60), "\n", sep = "")

  cat("\n  BACKBONE FLEXIBILITY STATISTICS:\n")
  cat("    • Mean RMSF:", sprintf("%.3f Å", mean(ca_rmsf)), "\n")
  cat("    • Median RMSF:", sprintf("%.3f Å", median(ca_rmsf)), "\n")
  cat("    • Standard Deviation:", sprintf("%.3f Å", sd(ca_rmsf)), "\n")
  cat("    • Minimum RMSF:", sprintf("%.3f Å", min(ca_rmsf)),
      "(Residue", residue_data$residue_number[which.min(ca_rmsf)],
      residue_data$residue_name[which.min(ca_rmsf)], ")\n")
  cat("    • Maximum RMSF:", sprintf("%.3f Å", max(ca_rmsf)),
      "(Residue", residue_data$residue_number[which.max(ca_rmsf)],
```

```r
      residue_data$residue_name[which.max(ca_rmsf)], ")\n")

  cat("\n  STABILITY CLASSIFICATION:\n")
  stable_residues <- sum(ca_rmsf < 1.5)
  flexible_residues <- sum(ca_rmsf >= 1.5 & ca_rmsf < 3.0)
  highly_flexible <- sum(ca_rmsf >= 3.0)
  total_residues <- length(ca_rmsf)

  cat("   • Very Stable (RMSF < 1.5 Å):", stable_residues, "residues",
      sprintf("(%.1f%%)", stable_residues/total_residues * 100), "\n")
  cat("   • Flexible (1.5-3.0 Å):", flexible_residues, "residues",
      sprintf("(%.1f%%)", flexible_residues/total_residues * 100), "\n")
  cat("   • Highly Flexible (>3.0 Å):", highly_flexible, "residues",
      sprintf("(%.1f%%)", highly_flexible/total_residues * 100), "\n")

  cat("\n  TOP 10 MOST FLEXIBLE REGIONS (RMSF > 3.5 Å):\n")
  flexible_regions <- residue_data %>%
    filter(ca_rmsf > 3.5) %>%
    arrange(desc(ca_rmsf)) %>%
    head(10)

  if (nrow(flexible_regions) > 0) {
    for (i in 1:nrow(flexible_regions)) {
      cat("   ", i, ". Residue", flexible_regions$residue_number[i],
          "(", flexible_regions$residue_name[i], "):",
          sprintf("%.2f Å", flexible_regions$ca_rmsf[i]), "\n")
    }
  } else {
    cat("   No residues with RMSF > 3.5 Å (good stability!)\n")
  }

  cat("\n  TOP 10 MOST STABLE REGIONS (RMSF < 1.0 Å):\n")
  stable_regions <- residue_data %>%
    filter(ca_rmsf < 1.0) %>%
    arrange(ca_rmsf) %>%
    head(10)

  if (nrow(stable_regions) > 0) {
    for (i in 1:nrow(stable_regions)) {
      cat("   ", i, ". Residue", stable_regions$residue_number[i],
          "(", stable_regions$residue_name[i], "):",
          sprintf("%.2f Å", stable_regions$ca_rmsf[i]), "\n")
    }
  } else {
    cat("   No residues with RMSF < 1.0 Å\n")
  }
}
```

```r
# MAIN FUNCTION: Run the complete RMSF analysis
run_complete_rmsf_analysis <- function() {
  cat("Starting RMSF Analysis of Samip_rmsf.tab\n")
  cat("Working directory:", getwd(), "\n")
  cat(strrep("-", 50), "\n")

  # Check if the data file exists
  if (!file.exists("Samip_rmsf.tab")) {
    cat("  ERROR: Samip_rmsf.tab not found in", getwd(), "\n")
    cat("Please make sure:\n")
    cat("1. The file 'Samip_rmsf.tab' is in the correct folder\n")
    cat("2. The working directory is set correctly\n")
    cat("3. The file name is spelled correctly\n")
    return(NULL)
  }

  # STEP 1: Load and parse the raw data
  atom_data <- read_rmsf_data("Samip_rmsf.tab")

  # STEP 2: Calculate residue-level averages
  residue_data <- calculate_residue_averages(atom_data)

  # STEP 3: Create visualizations
  cat("\nCreating plots...\n")
  main_plots <- create_main_plots(residue_data)
  detailed_plot <- create_detailed_plot(residue_data)

  # STEP 4: Generate text report
  generate_analysis_report(residue_data)

  # STEP 5: Save processed data for future use
  write_csv(residue_data, "processed_rmsf_data.csv")
  cat("  Saved: processed_rmsf_data.csv\n")

  # Final success message
  cat("\n" , strrep(" ", 20), "\n")
  cat("  RMSF ANALYSIS COMPLETE!\n")
  cat("  Output files created in:", getwd(), "\n")
  cat("   • rmsf_analysis_comprehensive.tiff - Main analysis figure (300 DPI)\n")
  cat("   • detailed_flexibility_analysis.tiff - Detailed stability plot (300 DPI)\n")
  cat("   • processed_rmsf_data.csv - Processed data for further analysis\n")
  cat(strrep(" ", 20), "\n")

  return(residue_data)
}

# Execute RMSF analysis
```

```r
residue_data <- run_complete_rmsf_analysis()
```

```
## Starting RMSF Analysis of Samip_rmsf.tab
## Working directory: C:/Users/SHAHIN/Desktop/SAMIP
## --------------------------------------------------
## Step 1: Reading data file: Samip_rmsf.tab
##   Successfully processed 7020 atom records
## Step 2: Calculating residue averages...
##   Processed 440 residues
##
## Creating plots...
## Step 3: Creating visualizations...

## Warning: Removed 4 rows containing missing values or values outside the scale range
## (`geom_line()`).

##   Saved: rmsf_analysis_comprehensive.tiff (300 DPI)

##   Saved: detailed_flexibility_analysis.tiff (300 DPI)
##
## ==============================================================
## PROTEIN STABILITY ANALYSIS REPORT
## File: Samip_rmsf.tab
## ==============================================================
##
##   BACKBONE FLEXIBILITY STATISTICS:
##     • Mean RMSF: 3.429 Å
##     • Median RMSF: 3.395 Å
##     • Standard Deviation: 1.130 Å
##     • Minimum RMSF: 1.020 Å (Residue 365 HIS )
##     • Maximum RMSF: 6.760 Å (Residue 43 ASN )
##
##   STABILITY CLASSIFICATION:
##     • Very Stable (RMSF < 1.5 Å): 11 residues (2.5%)
##     • Flexible (1.5-3.0 Å): 152 residues (34.5%)
##     • Highly Flexible (>3.0 Å): 277 residues (63.0%)
##
##   TOP 10 MOST FLEXIBLE REGIONS (RMSF > 3.5 Å):
##      1 . Residue 43 ( ASN ): 6.76 Å
##      2 . Residue 292 ( VAL ): 6.30 Å
##      3 . Residue 288 ( SER ): 6.23 Å
##      4 . Residue 45 ( ASP ): 6.17 Å
##      5 . Residue 344 ( TYR ): 6.09 Å
##      6 . Residue 44 ( GLN ): 6.07 Å
##      7 . Residue 89 ( SER ): 6.04 Å
##      8 . Residue 90 ( ASP ): 6.03 Å
##      9 . Residue 343 ( VAL ): 6.02 Å
##      10 . Residue 287 ( LYS ): 5.99 Å
##
```

```
##   TOP 10 MOST STABLE REGIONS (RMSF < 1.0 Å):
##     No residues with RMSF < 1.0 Å
##   Saved: processed_rmsf_data.csv
##
##
##   RMSF ANALYSIS COMPLETE!
##   Output files created in: C:/Users/SHAHIN/Desktop/SAMIP
##     • rmsf_analysis_comprehensive.tiff - Main analysis figure (300 DPI)
##     • detailed_flexibility_analysis.tiff - Detailed stability plot (300 DPI)
##     • processed_rmsf_data.csv - Processed data for further analysis
##
```

```
# ===================================================================
# === PART 2: Secondary Structure (SS) Analysis =====================
# ===================================================================
cat("--- PART 2: Secondary Structure Analysis ---\n")
```

```
## --- PART 2: Secondary Structure Analysis ---
```

```
# STEP 2.1 - Read and Prepare SS data ---------------------------------
cat("Reading SS data file: Samip_plotres_secstrMolA.tab...\n")
```

```
## Reading SS data file: Samip_plotres_secstrMolA.tab...
```

```
# Read the secondary structure file (Skip first 2 lines since they are not data)
data_ss <- read.table("Samip_plotres_secstrMolA.tab",
                      skip = 2, header = FALSE, stringsAsFactors = FALSE)

# Extract time points (ps) and SS data
time_points_ps <- as.numeric(data_ss[, 1])
```

```
## Warning: NAs introduced by coercion
```

```
ss_data_raw <- data_ss[, -1]

# Convert to numeric matrix, remove rows with missing time
ss_data_matrix <- as.matrix(ss_data_raw)
mode(ss_data_matrix) <- "numeric"
```

```
## Warning in mde(x): NAs introduced by coercion
```

```
valid_rows_ss <- !is.na(time_points_ps)
time_points_ps <- time_points_ps[valid_rows_ss]
ss_data_matrix <- ss_data_matrix[valid_rows_ss, ]

# Define residues and adjust if necessary
n_times <- length(time_points_ps)
n_residues_ss <- ncol(ss_data_matrix)
residues_ss <- 34:(33 + n_residues_ss) # Initial guess based on script

cat("Data dimensions:", n_times, "time points ×", n_residues_ss, "residues (Residues", min(res
```

```
## Data dimensions: 1027 time points × 441 residues (Residues 34 to 474 )
```

```r
# Convert time to nanoseconds (assuming 102 ns total simulation time)
total_time_ns <- 102
conversion_factor <- total_time_ns / max(time_points_ps, na.rm=TRUE)
time_points_ns <- time_points_ps * conversion_factor
cat("Converted time from ps → ns (0 to", round(max(time_points_ns), 1), "ns)\n")
```

```
## Converted time from ps → ns (0 to 102 ns)
```

```r
# Define structure codes and clean data
ss_labels <- c("0" = "Coil", "1" = "Turn", "2" = "Bend",
               "3" = "Helix", "4" = "Strand", "5" = "Coil", "6" = "Coil")

ss_data_clean <- ss_data_matrix
# Replace invalid codes with 5 (Coil) for calculations
ss_data_clean[!ss_data_clean %in% 1:5] <- 5

# Define common theme for all SS plots with NO GRID BOX
ss_theme <- theme(
  axis.text.x = element_text(face = "bold", size = 16, color = "black"),
  axis.text.y = element_text(face = "bold", size = 16, color = "black"),
  axis.title.x = element_text(face = "bold", size = 18, color = "black"),
  axis.title.y = element_text(face = "bold", size = 18, color = "black"),
  plot.title = element_text(face = "bold", size = 20, hjust = 0.5),
  legend.title = element_text(face = "bold", size = 18),
  legend.text = element_text(face = "bold", size = 18),
  # REMOVED GRID LINES
  panel.grid.major = element_blank(),
  panel.grid.minor = element_blank(),
  panel.background = element_blank(),
  axis.line = element_line(color = "black", linewidth = 0.5)
)

# STEP 2.2 - Create the timeline plot ---------------------------------
timeline_df <- data.frame(
  Time_ns = rep(time_points_ns, each = n_residues_ss),
  Residue = rep(residues_ss, times = n_times),
  SS = as.vector(t(ss_data_matrix))
)

timeline_df$SS_Code <- as.character(timeline_df$SS)
# Replace unknown codes with "5" (Coil)
timeline_df$SS_Code[!timeline_df$SS_Code %in% c("1","2","3","4","5")] <- "5"
timeline_df$SS_Label <- factor(ss_labels[timeline_df$SS_Code],
                               levels = c("Helix","Strand","Turn","Bend","Coil"))

p1_ss <- ggplot(timeline_df, aes(x = Residue, y = Time_ns, fill = SS_Label)) +
  geom_tile() +
```

```r
  scale_fill_viridis(discrete = TRUE) +
  scale_y_continuous(breaks = seq(0, 100, 20), limits = c(0, 102)) +
  scale_x_continuous(breaks = seq(min(0), max(residues_ss), by = 50)) +
  labs(title = "Secondary Structure Evolution Over Time",
       x = "Residue Number", y = "Time (ns)", fill = "Secondary Structure") +
  theme_minimal() +
  ss_theme +
  theme(axis.text.x = element_text(angle = 00, hjust = 0, size = 14, face = "bold", color = "bl

ggsave(file.path(ss_output_dir, "01_timeline_plot.tiff"), p1_ss, width = 12, height = 8, dpi =
```

```
## Warning: Removed 882 rows containing missing values or values outside the scale range
## (`geom_tile()`).
```

```r
cat("Saved: ss_plots/01_timeline_plot.tiff\n")
```

```
## Saved: ss_plots/01_timeline_plot.tiff
```

```r
# STEP 2.3 - Secondary structure fractions over time ------------------
ss_fractions <- data.frame(
  Time_ns = time_points_ns,
  Helix = rowSums(ss_data_clean == 3) / n_residues_ss,
  Strand = rowSums(ss_data_clean == 4) / n_residues_ss,
  Coil = rowSums(ss_data_clean == 5) / n_residues_ss,
  Turn = rowSums(ss_data_clean == 1) / n_residues_ss,
  Bend = rowSums(ss_data_clean == 2) / n_residues_ss
)

ss_fractions_long <- melt(ss_fractions, id.vars = "Time_ns",
                          variable.name = "SS_Type", value.name = "Fraction")

p2_ss <- ggplot(ss_fractions_long, aes(x = Time_ns, y = Fraction, color = SS_Type)) +
  geom_line(linewidth = 1) +
  scale_x_continuous(breaks = seq(0, 100, 20), limits = c(0, 102)) +
  scale_y_continuous(breaks = seq(0, 1, 0.2)) +
  labs(title = "Secondary Structure Composition Over Time",
       x = "Time (ns)", y = "Fraction of Residues", color = "Structure") +
  theme_minimal() +
  scale_color_viridis(discrete = TRUE) +
  ss_theme

ggsave(file.path(ss_output_dir, "02_fractions_plot.tiff"), p2_ss, width = 10, height = 6, dpi =
cat("Saved: ss_plots/02_fractions_plot.tiff\n")
```

```
## Saved: ss_plots/02_fractions_plot.tiff
```

```r
# STEP 2.4 - Residue-wise stability -----------------------------------
residue_stability <- apply(ss_data_clean, 2, function(x) {
  max(table(x)) / length(x)
```

```r
})

stability_df <- data.frame(Residue = residues_ss, Stability = residue_stability)

p3_ss <- ggplot(stability_df, aes(x = Residue, y = Stability)) +
  geom_col(fill = "steelblue", alpha = 0.7) +
  scale_x_continuous(breaks = seq(min(0), max(residues_ss), by = 50)) +
  scale_y_continuous(breaks = seq(0, 1, 0.2)) +
  labs(title = "Residue-wise Secondary Structure Stability",
       x = "Residue Number", y = "Stability (Frequency of Most Common State)") +
  theme_minimal() +
  ss_theme +
  theme(axis.text.x = element_text(angle = 00, hjust = 1, size = 14, face = "bold", color = "bl
  geom_hline(yintercept = 0.8, linetype = "dashed", color = "red")

ggsave(file.path(ss_output_dir, "03_stability_heatmap.tiff"), p3_ss, width = 12, height = 6, dp
cat("Saved: ss_plots/03_stability_heatmap.tiff\n")
```

```
## Saved: ss_plots/03_stability_heatmap.tiff
```

```r
# STEP 2.5 - Transition frequency matrix ------------------------------
calculate_transitions <- function(ss_data) {
  states <- 1:5
  matrix_out <- matrix(0, nrow = 5, ncol = 5, dimnames = list(states, states))

  for (res in 1:ncol(ss_data)) {
    for (t in 2:nrow(ss_data)) {
      from <- ss_data[t-1, res]
      to <- ss_data[t, res]
      if (!is.na(from) && !is.na(to) && from %in% 1:5 && to %in% 1:5 && from != to) {
        matrix_out[from, to] <- matrix_out[from, to] + 1
      }
    }
  }
  return(matrix_out)
}

trans_mat <- calculate_transitions(ss_data_clean)
trans_df <- melt(trans_mat)
colnames(trans_df) <- c("From", "To", "Count")

# Remap 5 to 5 for accurate labels
ss_labels_plot <- c("1" = "Turn", "2" = "Bend", "3" = "Helix", "4" = "Strand", "5" = "Coil")

trans_df$From_Label <- ss_labels_plot[as.character(trans_df$From)]
trans_df$To_Label <- ss_labels_plot[as.character(trans_df$To)]

p4_ss <- ggplot(trans_df, aes(x = From_Label, y = To_Label, fill = Count)) +
```

```r
  geom_tile() +
  geom_text(aes(label = Count), color = "white", size = 4, fontface = "bold") +
  scale_fill_viridis() +
  labs(title = "Secondary Structure Transition Matrix",
       x = "From State", y = "To State") +
  theme_minimal() +
  ss_theme +
  theme(axis.text.x = element_text(face = "bold", size = 12, color = "black"),
        axis.text.y = element_text(face = "bold", size = 12, color = "black"))

ggsave(file.path(ss_output_dir, "04_transition_matrix.tiff"), p4_ss, width = 8, height = 6, dp
cat("Saved: ss_plots/04_transition_matrix.tiff\n")
```

```
## Saved: ss_plots/04_transition_matrix.tiff
```

```r
# STEP 2.6 - Running average of structural changes --------------------
running_changes <- sapply(2:nrow(ss_data_clean), function(i) {
  sum(ss_data_clean[i, ] != ss_data_clean[i - 1, ], na.rm = TRUE) / n_residues_ss
})

change_df <- data.frame(Time_ns = time_points_ns[-1],
                        ChangeRate = running_changes)

p5_ss <- ggplot(change_df, aes(x = Time_ns, y = ChangeRate)) +
  geom_line(color = "red", linewidth = 1) +
  geom_smooth(method = "loess", se = TRUE, alpha = 0.2) +
  scale_x_continuous(breaks = seq(0, 100, 20), limits = c(0, 102)) +
  scale_y_continuous(breaks = seq(0, max(change_df$ChangeRate, na.rm = TRUE),
                                  by = round(max(change_df$ChangeRate, na.rm = TRUE)/5, 2))) +
  labs(title = "Rate of Secondary Structure Changes Over Time",
       x = "Time (ns)", y = "Fraction of Residues Changing") +
  theme_minimal() +
  ss_theme

ggsave(file.path(ss_output_dir, "05_change_rate.tiff"), p5_ss, width = 10, height = 6, dpi = 3
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

```r
cat("Saved: ss_plots/05_change_rate.tiff\n")
```

```
## Saved: ss_plots/05_change_rate.tiff
```

```r
cat("  PART 2 (Secondary Structure) Analysis complete!\n\n")
```

```
##   PART 2 (Secondary Structure) Analysis complete!
```

```r
# ====================================================================
# === PART 3: Residue Contacts Analysis ==============================
# ====================================================================
cat("--- PART 3: Residue Contacts Analysis ---\n")
```

--- PART 3: Residue Contacts Analysis ---

```r
# STEP 3.1 - Load and Clean Data ---------------------------------------
cat("Reading Contacts data file: Samip_plotres_conMolA.tab...\n")
```

Reading Contacts data file: Samip_plotres_conMolA.tab...

```r
# Read the file as text to find where numeric data begins
file_lines_con <- readLines("Samip_plotres_conMolA.tab")

# Find the first line that starts with a number
data_start_con <- grep("^\\s*[0-9]", file_lines_con)[1]

# Read data starting from that line
data_con <- read.table("Samip_plotres_conMolA.tab",
                       header = FALSE,
                       skip = data_start_con - 1,
                       fill = TRUE,
                       stringsAsFactors = FALSE)

# Convert all values to numbers and remove any unwanted text
numeric_data_con <- as.data.frame(lapply(data_con, function(x) {
  as.numeric(gsub("[^0-9.Ee-]", "", x))
}))

# Remove completely empty columns
numeric_data_con <- numeric_data_con[, colSums(is.na(numeric_data_con)) < nrow(numeric_data_co

# STEP 3.2 - Extract Time and Contact Data ----------------------------
# First column = time in picoseconds → convert to nanoseconds
time_con <- numeric_data_con[,1] / 1000

# Other columns = contact numbers per residue
contacts <- numeric_data_con[,-1]
contacts <- contacts[, colSums(is.na(contacts)) < nrow(contacts)]

# Convert to a matrix for calculations
contacts_matrix <- as.matrix(contacts)

# Calculate metrics
total_contacts <- rowSums(contacts_matrix, na.rm = TRUE)
mean_contacts <- colMeans(contacts_matrix, na.rm = TRUE)
residue_numbers_con <- 1:length(mean_contacts)
sd_contacts <- apply(contacts_matrix, 2, sd, na.rm = TRUE)

# Define consistent plotting parameters - NO GRID
contact_theme_par <- list(
  cex.lab = 1.6,      # Axis labels size
  cex.axis = 1.6,     # Axis numbers size
```

```r
  cex.main = 1.8,      # Main title size
  col.lab = "black",   # Axis labels color
  col.axis = "black", # Axis numbers color
  font.lab = 2,        # Bold axis labels
  font.axis = 2,       # Bold axis numbers
  font.main = 2        # Bold main title
)

# STEP 3.3 - Total Contacts Over Time Plot ----------------------------
tiff(file.path(contact_output_dir, "total_contacts_stability.tiff"),
     width = 8, height = 6, units = "in", res = 300, compression = "lzw")

par(mar = c(5, 5, 4, 2) + 0.1)  # Adjust margins for larger labels

plot(time_con, total_contacts, type = "l", lwd = 2, col = "blue",
     xlab = "Time (ns)", ylab = "Total Contacts",
     main = paste("Protein Stability Over", round(max(time_con, na.rm = TRUE), 1), "ns"),
     cex.lab = contact_theme_par$cex.lab,
     cex.axis = contact_theme_par$cex.axis,
     cex.main = contact_theme_par$cex.main,
     col.lab = contact_theme_par$col.lab,
     col.axis = contact_theme_par$col.axis,
     font.lab = contact_theme_par$font.lab,
     font.axis = contact_theme_par$font.axis,
     font.main = contact_theme_par$font.main,
     bty = "l")  # Remove box around plot, keep axes

if (length(total_contacts) > 10) {
  running_avg <- rollmean(total_contacts, k = 5, fill = NA)
  lines(time_con, running_avg, col = "red", lwd = 2)
  legend("topright",
         legend = c("Total Contacts", "Running Average (k=5)"),
         col = c("blue", "red"), lwd = 2, bg = "white",
         cex = 1.2, text.font = 2)  # Bold legend text
}
# REMOVED grid() call
dev.off()
```

```
## pdf
##   2
```

```r
cat("Saved: protein_stability_plots/total_contacts_stability.tiff\n")
```

```
## Saved: protein_stability_plots/total_contacts_stability.tiff
```

```r
# STEP 3.4 - Average Contacts Per Residue Plot ------------------------
tiff(file.path(contact_output_dir, "contact_distribution.tiff"),
     width = 10, height = 6, units = "in", res = 300, compression = "lzw")
```

```r
par(mar = c(5, 5, 4, 2) + 0.1)

plot(residue_numbers_con, mean_contacts, type = "h", lwd = 1.5, col = "red",
     xlab = "Residue Number", ylab = "Mean Contacts",
     main = "Average Contacts per Residue",
     cex.lab = contact_theme_par$cex.lab,
     cex.axis = contact_theme_par$cex.axis,
     cex.main = contact_theme_par$cex.main,
     col.lab = contact_theme_par$col.lab,
     col.axis = contact_theme_par$col.axis,
     font.lab = contact_theme_par$font.lab,
     font.axis = contact_theme_par$font.axis,
     font.main = contact_theme_par$font.main,
     bty = "l")  # Remove box around plot, keep axes

abline(h = mean(mean_contacts), col = "blue", lty = 2, lwd = 2)
legend("topright", legend = "Overall Mean", col = "blue", lty = 2, lwd = 2,
       cex = 1.2, text.font = 2)  # Bold legend text
# REMOVED grid() call
dev.off()
```

```
## pdf
##   2
```

```r
cat("Saved: protein_stability_plots/contact_distribution.tiff\n")
```

```
## Saved: protein_stability_plots/contact_distribution.tiff
```

```r
# STEP 3.5 - Contact Fluctuation Plot ---------------------------------
tiff(file.path(contact_output_dir, "contact_fluctuation.tiff"),
     width = 10, height = 6, units = "in", res = 300, compression = "lzw")

par(mar = c(5, 5, 4, 2) + 0.1)

plot(residue_numbers_con, sd_contacts, type = "h", lwd = 1.5, col = "purple",
     xlab = "Residue Number", ylab = "Standard Deviation",
     main = "Contact Fluctuation - Higher = More Flexible",
     cex.lab = contact_theme_par$cex.lab,
     cex.axis = contact_theme_par$cex.axis,
     cex.main = contact_theme_par$cex.main,
     col.lab = contact_theme_par$col.lab,
     col.axis = contact_theme_par$col.axis,
     font.lab = contact_theme_par$font.lab,
     font.axis = contact_theme_par$font.axis,
     font.main = contact_theme_par$font.main,
     bty = "l")  # Remove box around plot, keep axes

abline(h = mean(sd_contacts), col = "orange", lty = 2, lwd = 2)
legend("topright", legend = "Mean Fluctuation", col = "orange", lty = 2, lwd = 2,
```

```r
       cex = 1.2, text.font = 2)  # Bold legend text
# REMOVED grid() call
dev.off()
```

```
## pdf
##   2
```

```r
cat("Saved: protein_stability_plots/contact_fluctuation.tiff\n")
```

```
## Saved: protein_stability_plots/contact_fluctuation.tiff
```

```r
# STEP 3.6 - Contact Heatmap (Optional) -------------------------------
if (ncol(contacts_matrix) <= 100 && nrow(contacts_matrix) > 1) {
  tiff(file.path(contact_output_dir, "contact_heatmap.tiff"),
       width = 10, height = 8, units = "in", res = 300, compression = "lzw")

  par(mar = c(5, 5, 4, 4) + 0.1)

  time_sample_con <- seq(1, nrow(contacts_matrix), length.out = min(50, nrow(contacts_matrix))
  contacts_sample <- contacts_matrix[time_sample_con, ]

  image(1:ncol(contacts_sample), time_con[time_sample_con], t(contacts_sample),
        col = colorRampPalette(c("white", "blue", "red"))(100),
        xlab = "Residue Number",
        ylab = "Time (ns)",
        main = "Contact Map Over Time (Sampled)",
        cex.lab = contact_theme_par$cex.lab,
        cex.axis = contact_theme_par$cex.axis,
        cex.main = contact_theme_par$cex.main,
        col.lab = contact_theme_par$col.lab,
        col.axis = contact_theme_par$col.axis,
        font.lab = contact_theme_par$font.lab,
        font.axis = contact_theme_par$font.axis,
        font.main = contact_theme_par$font.main)
  dev.off()
  cat("Saved: protein_stability_plots/contact_heatmap.tiff\n")
} else {
  cat("Skipped contact heatmap: too many residues or too few time points.\n")
}
```

```
## Skipped contact heatmap: too many residues or too few time points.
```

```r
# STEP 3.7 - Stability Correlation ------------------------------------
correlation <- NA
if (nrow(contacts_matrix) > 20) {
  half_point <- floor(nrow(contacts_matrix) / 2)
  first_half <- colMeans(contacts_matrix[1:half_point, ], na.rm = TRUE)
  second_half <- colMeans(contacts_matrix[(half_point + 1):nrow(contacts_matrix), ], na.rm = T

  correlation <- cor(first_half, second_half, use = "complete.obs")
```

```r
  tiff(file.path(contact_output_dir, "stability_correlation.tiff"),
       width = 8, height = 6, units = "in", res = 300, compression = "lzw")

  par(mar = c(5, 5, 4, 2) + 0.1)

  plot(first_half, second_half, pch = 19, col = rgb(0, 0.5, 0, 0.6),
       xlab = "Mean Contacts - First Half",
       ylab = "Mean Contacts - Second Half",
       main = paste("Contact Conservation (r =", round(correlation, 3), ")"),
       cex.lab = contact_theme_par$cex.lab,
       cex.axis = contact_theme_par$cex.axis,
       cex.main = contact_theme_par$cex.main,
       col.lab = contact_theme_par$col.lab,
       col.axis = contact_theme_par$col.axis,
       font.lab = contact_theme_par$font.lab,
       font.axis = contact_theme_par$font.axis,
       font.main = contact_theme_par$font.main,
       bty = "l")  # Remove box around plot, keep axes

  abline(0, 1, col = "red", lwd = 2)
  # REMOVED grid() call

  # Add stability text with bold font
  y_pos <- max(second_half) * 0.9
  if (correlation > 0.8) {
    text(mean(first_half), y_pos, "STABLE", col = "darkgreen", cex = 1.5, font = 2)
  } else if (correlation > 0.6) {
    text(mean(first_half), y_pos, "MODERATELY STABLE", col = "orange", cex = 1.5, font = 2)
  } else {
    text(mean(first_half), y_pos, "POTENTIALLY UNSTABLE", col = "red", cex = 1.5, font = 2)
  }
  dev.off()
  cat("Saved: protein_stability_plots/stability_correlation.tiff\n")
} else {
  cat("Skipped stability correlation: not enough time points (>20 frames required).\n")
}
```

```
## Saved: protein_stability_plots/stability_correlation.tiff
```

```r
# STEP 3.8 - Combined Dashboard (4 in 1) ------------------------------
if (exists("first_half") && exists("second_half") && !is.na(correlation)) {
  tiff(file.path(contact_output_dir, "stability_dashboard.tiff"),
       width = 12, height = 8, units = "in", res = 300, compression = "lzw")

  par(mfrow = c(2, 2), mar = c(5, 5, 4, 2) + 0.1, oma = c(0, 0, 2, 0))

  # A) Total contacts
```

```r
  plot(time_con, total_contacts, type = "l", lwd = 2, col = "blue",
       xlab = "Time (ns)", ylab = "Total Contacts", main = "A) Total Contacts Over Time",
       cex.lab = 1.4, cex.axis = 1.4, cex.main = 1.5,
       col.lab = "black", col.axis = "black", font.lab = 2, font.axis = 2, font.main = 2,
       bty = "l")
  # REMOVED grid()

  # B) Mean contacts
  plot(residue_numbers_con, mean_contacts, type = "h", lwd = 1, col = "red",
       xlab = "Residue Number", ylab = "Mean Contacts", main = "B) Contact Distribution",
       cex.lab = 1.4, cex.axis = 1.4, cex.main = 1.5,
       col.lab = "black", col.axis = "black", font.lab = 2, font.axis = 2, font.main = 2,
       bty = "l")

  # C) Fluctuation
  plot(residue_numbers_con, sd_contacts, type = "h", lwd = 1, col = "purple",
       xlab = "Residue Number", ylab = "Std Deviation", main = "C) Contact Fluctuation",
       cex.lab = 1.4, cex.axis = 1.4, cex.main = 1.5,
       col.lab = "black", col.axis = "black", font.lab = 2, font.axis = 2, font.main = 2,
       bty = "l")

  # D) Correlation
  plot(first_half, second_half, pch = 19, col = rgb(0, 0.5, 0, 0.6),
       xlab = "First Half", ylab = "Second Half",
       main = paste("D) Stability (r =", round(correlation, 3), ")"),
       cex.lab = 1.4, cex.axis = 1.4, cex.main = 1.5,
       col.lab = "black", col.axis = "black", font.lab = 2, font.axis = 2, font.main = 2,
       bty = "l")
  abline(0, 1, col = "red", lwd = 1)

  mtext("Contact Analysis Dashboard", outer = TRUE, cex = 1.8, font = 2)
  dev.off()
  cat("Saved: protein_stability_plots/stability_dashboard.tiff\n")
} else {
  cat("Skipped stability dashboard: correlation data not available.\n")
}
```

```
## Saved: protein_stability_plots/stability_dashboard.tiff
```

```r
# STEP 3.9 - Summary Output -------------------------------------------
cat("\nStability Assessment:\n")
```

```
##
## Stability Assessment:
```

```r
change_pct <- NA
if (length(total_contacts) > 10) {
  initial <- mean(total_contacts[1:5], na.rm = TRUE)
  final <- mean(tail(total_contacts, 5), na.rm = TRUE)
```

```r
  change_pct <- (final - initial) / initial * 100
  cat("Contact change (Initial vs Final):", round(change_pct, 1), "%\n")
}
```

```
## Contact change (Initial vs Final): -80.2 %
```

```r
if (!is.na(correlation)) {
  cat("Correlation (First Half vs Second Half):", round(correlation, 3), "\n")
  if (!is.na(change_pct) && abs(change_pct) < 5 && correlation > 0.8) {
    cat("CONCLUSION:  Protein is STABLE\n")
  } else {
    cat("CONCLUSION:  Check individual plots for stability assessment\n")
  }
} else {
  cat("CONCLUSION: Not enough data for full stability assessment.\n")
}
```

```
## Correlation (First Half vs Second Half): 0.824
## CONCLUSION:  Check individual plots for stability assessment
```

```r
cat(" PART 3 (Residue Contacts) Analysis complete!\n\n")
```

```
##  PART 3 (Residue Contacts) Analysis complete!
```

```r
# ==================================================================
# === PART 4: Dynamic Cross-Correlation Analysis (DCCM/PCA) =========
# ==================================================================
cat("--- PART 4: DCCM and PCA Analysis (Essential Dynamics) ---\n")
```

```
## --- PART 4: DCCM and PCA Analysis (Essential Dynamics) ---
```

```r
# STEP 4.1 - Read DCCM data ------------------------------------------
cat("Reading DCCM data file: Samip_dccm.tab...\n")
```

```
## Reading DCCM data file: Samip_dccm.tab...
```

```r
dccm_data <- read.table("Samip_dccm.tab", skip = 1)
dccm_matrix <- as.matrix(dccm_data)

# Clear column/row names for plotting (if present)
colnames(dccm_matrix) <- NULL
rownames(dccm_matrix) <- NULL

n_residues_dccm <- nrow(dccm_matrix)
cat("DCCM matrix size:", n_residues_dccm, "x", n_residues_dccm, "\n")
```

```
## DCCM matrix size: 440 x 440
```

```r
# Define consistent theme for all DCCM plots
dccm_theme <- list(
  axis.text = element_text(face = "bold", size = 16, color = "black"),
  axis.title = element_text(face = "bold", size = 16, color = "black"),
```

```r
  plot.title = element_text(face = "bold", size = 18, hjust = 0.5),
  legend.title = element_text(face = "bold", size = 14, color = "black"),
  legend.text = element_text(face = "bold", size = 12, color = "black")
)

# STEP 4.2 - Heatmap of DCCM (lattice) --------------------------------
tiff(file.path(dccm_output_dir, "1_heatmap.tiff"), width = 8, height = 6, units = "in", res = 3

# Create custom theme for lattice plot
custom_lattice_theme <- list(
  par.xlab.text = list(cex = 1.6, font = 2, col = "black"),
  par.ylab.text = list(cex = 1.6, font = 2, col = "black"),
  axis.text = list(cex = 1.4, font = 2, col = "black"),
  add.text = list(cex = 1.2, font = 2, col = "black")
)

levelplot(dccm_matrix,
          main = list(label = "Dynamic Cross-Correlation Matrix", cex = 1.8, font = 2),
          xlab = list(label = "Residue Number", cex = 1.6, font = 2),
          ylab = list(label = "Residue Number", cex = 1.6, font = 2),
          col.regions = colorRampPalette(c("blue", "white", "red")),
          scales = list(
            x = list(at = seq(0, 440, 100), cex = 1.4, font = 2, col = "black"),
            y = list(at = seq(0, 440, 100), cex = 1.4, font = 2, col = "black")
          ),
          par.settings = custom_lattice_theme)
dev.off()
```

```
## pdf
##   2
```

```r
cat("Saved: Heatmap/1_heatmap.tiff\n")
```

```
## Saved: Heatmap/1_heatmap.tiff
```

```r
# STEP 4.3 - Contact Map (ggplot2) ------------------------------------
# Convert matrix to long format for ggplot
dccm_long <- melt(dccm_matrix)
names(dccm_long) <- c("Residue_i", "Residue_j", "Correlation")

# Make sure residue indices are numeric
dccm_long$Residue_i <- as.numeric(dccm_long$Residue_i)
dccm_long$Residue_j <- as.numeric(dccm_long$Residue_j)

p_contact <- ggplot(dccm_long, aes(x = Residue_i, y = Residue_j, fill = Correlation)) +
  geom_tile() +
  scale_fill_gradient2(low = "blue", mid = "white", high = "red",
                       midpoint = 0, limits = c(-1, 1),
                       name = "Correlation") +
```

```r
  labs(title = "DCCM Contact Map",
       x = "Residue Number",
       y = "Residue Number") +
  scale_x_continuous(breaks = seq(0, n_residues_dccm, 100)) +
  scale_y_continuous(breaks = seq(0, n_residues_dccm, 100)) +
  theme_minimal() +
  theme(
    axis.text.x = dccm_theme$axis.text,
    axis.text.y = dccm_theme$axis.text,
    axis.title.x = dccm_theme$axis.title,
    axis.title.y = dccm_theme$axis.title,
    plot.title = dccm_theme$plot.title,
    legend.title = dccm_theme$legend.title,
    legend.text = dccm_theme$legend.text
  )

ggsave(file.path(dccm_output_dir, "2_contact_map.tiff"), plot = p_contact, device = "tiff",
       width = 8, height = 6, dpi = 300)
cat("Saved: Heatmap/2_contact_map.tiff\n")
```

```
## Saved: Heatmap/2_contact_map.tiff
```

```r
# STEP 4.4 - Histogram of Correlation Values --------------------------
tiff(file.path(dccm_output_dir, "3_histogram.tiff"), width = 8, height = 6, units = "in", res =

par(mar = c(5, 5, 4, 2) + 0.1)
hist(dccm_matrix,
     breaks = 50,
     main = "Distribution of Correlation Coefficients",
     xlab = "Correlation Value",
     ylab = "Frequency",
     col = "lightblue",
     border = "black",
     cex.lab = 1.6,
     cex.axis = 1.6,
     cex.main = 1.8,
     col.lab = "black",
     col.axis = "black",
     font.lab = 2,
     font.axis = 2,
     font.main = 2)
abline(v = mean(dccm_matrix), col = "red", lwd = 2, lty = 2)
legend("topright", legend = paste("Mean =", round(mean(dccm_matrix), 3)),
       col = "red", lwd = 2, cex = 1.4, text.font = 2)
dev.off()
```

```
## pdf
##   2
```

```r
cat("Saved: Heatmap/3_histogram.tiff\n")
```

```
## Saved: Heatmap/3_histogram.tiff
```

```r
# STEP 4.5 - Correlation vs Sequence Distance (Sampled Plot) ----------
# Calculate how far residues are apart (sequence distance)
dist_matrix <- outer(1:n_residues_dccm, 1:n_residues_dccm, FUN = function(i, j) abs(i - j))

# Convert both matrices into vectors
dist_vector <- as.vector(dist_matrix)
corr_vector <- as.vector(dccm_matrix)

# To avoid clutter, use a smaller random sample of points
set.seed(123)
sample_size <- min(10000, length(dist_vector))
sample_idx <- sample(seq_along(dist_vector), sample_size)

# Create data frame for plotting
plot_data_dccm <- data.frame(
  distance = dist_vector[sample_idx],
  correlation = corr_vector[sample_idx]
)

tiff(file.path(dccm_output_dir, "4_distance_vs_correlation_sampled.tiff"), width = 8, height =

par(mar = c(5, 5, 4, 2) + 0.1)
plot(plot_data_dccm$distance, plot_data_dccm$correlation,
     pch = 20, cex = 0.5, col = rgb(0, 0, 1, 0.3),
     xlab = "Sequence Distance (Residues)",
     ylab = "Correlation Coefficient",
     main = "Correlation vs Sequence Separation (Sampled Points)",
     cex.lab = 1.6,
     cex.axis = 1.6,
     cex.main = 1.6,
     col.lab = "black",
     col.axis = "black",
     font.lab = 2,
     font.axis = 2,
     font.main = 2)

# Add a smooth trend line
smooth_line <- loess(correlation ~ distance, data = plot_data_dccm)
dist_seq <- seq(1, max(plot_data_dccm$distance), length = 100)
smooth_pred <- predict(smooth_line, newdata = data.frame(distance = dist_seq))
lines(dist_seq, smooth_pred, col = "red", lwd = 3)
dev.off()
```

```
## pdf
```

```
##   2
```

```r
cat("Saved: Heatmap/4_distance_vs_correlation_sampled.tiff\n")
```

```
## Saved: Heatmap/4_distance_vs_correlation_sampled.tiff
```

```r
# STEP 4.6 - Simpler Distance vs Correlation Plot (Density) -----------
tiff(file.path(dccm_output_dir, "8_smooth_distance_correlation.tiff"), width = 8, height = 6, u

par(mar = c(5, 5, 4, 2) + 0.1)
smoothScatter(dist_vector, corr_vector,
              xlab = "Sequence Distance (Residues)",
              ylab = "Correlation Coefficient",
              main = "Correlation vs Sequence Separation (Density Plot)",
              nbin = 100,
              cex.lab = 1.6,
              cex.axis = 1.6,
              cex.main = 1.8,
              col.lab = "black",
              col.axis = "black",
              font.lab = 2,
              font.axis = 2,
              font.main = 2)

# Add overall average trend line
avg_corr <- tapply(corr_vector, cut(dist_vector, breaks = 50), mean)
dist_mid <- tapply(dist_vector, cut(dist_vector, breaks = 50), mean)
lines(dist_mid, avg_corr, col = "red", lwd = 3)
dev.off()
```

```
## pdf
##   2
```

```r
cat("Saved: Heatmap/8_smooth_distance_correlation.tiff\n")
```

```
## Saved: Heatmap/8_smooth_distance_correlation.tiff
```

```r
# STEP 4.7 - PCA (Principal Component Analysis / Essential Dynamics) --
pca_result <- prcomp(dccm_matrix, center = TRUE, scale. = FALSE)
variance_explained <- pca_result$sdev^2 / sum(pca_result$sdev^2) * 100

# Scree plot: shows variance explained by each PC
tiff(file.path(dccm_output_dir, "5_scree_plot.tiff"), width = 8, height = 6, units = "in", res

par(mar = c(5, 5, 4, 2) + 0.1)
plot(1:10, variance_explained[1:10], type = "b",
     xlab = "Principal Component (PC) Number",
     ylab = "Variance Explained (%)",
     main = "Scree Plot - Essential Dynamics (Top 10 PCs)",
     pch = 19, col = "blue", lwd = 2,
```

```r
     cex.lab = 1.6,
     cex.axis = 1.6,
     cex.main = 1.8,
     col.lab = "black",
     col.axis = "black",
     font.lab = 2,
     font.axis = 2,
     font.main = 2)
# REMOVED grid()
dev.off()
```

```
## pdf
##   2
```

```r
cat("Saved: Heatmap/5_scree_plot.tiff\n")
```

```
## Saved: Heatmap/5_scree_plot.tiff
```

```r
# Cumulative variance plot
tiff(file.path(dccm_output_dir, "6_cumulative_variance.tiff"), width = 8, height = 6, units = "

par(mar = c(5, 5, 4, 2) + 0.1)
cumulative_variance <- cumsum(variance_explained)
# Plot up to 20 components or the total number of components if fewer
plot_limit <- min(20, length(cumulative_variance))
plot(1:plot_limit, cumulative_variance[1:plot_limit], type = "b",
     xlab = "Number of Principal Components",
     ylab = "Cumulative Variance Explained (%)",
     main = "Cumulative Variance Explained",
     pch = 19, col = "red", lwd = 2,
     cex.lab = 1.6,
     cex.axis = 1.6,
     cex.main = 1.8,
     col.lab = "black",
     col.axis = "black",
     font.lab = 2,
     font.axis = 2,
     font.main = 2)
abline(h = 80, col = "gray", lty = 2)
dev.off()
```

```
## pdf
##   2
```

```r
cat("Saved: Heatmap/6_cumulative_variance.tiff\n")
```

```
## Saved: Heatmap/6_cumulative_variance.tiff
```

```r
# PC1 vs PC2 plot (visualizing collective motions)
tiff(file.path(dccm_output_dir, "7_pc1_vs_pc2.tiff"), width = 8, height = 6, units = "in", res
```

```r
par(mar = c(5, 5, 4, 2) + 0.1)
plot(pca_result$x[,1], pca_result$x[,2],
     xlab = paste0("PC1 (", round(variance_explained[1], 1), "%)"),
     ylab = paste0("PC2 (", round(variance_explained[2], 1), "%)"),
     main = "PC1 vs PC2 - Collective Motions",
     pch = 20, col = "blue", cex = 0.8,
     cex.lab = 1.6,
     cex.axis = 1.6,
     cex.main = 1.8,
     col.lab = "black",
     col.axis = "black",
     font.lab = 2,
     font.axis = 2,
     font.main = 2)
dev.off()
```

```
## pdf
##   2
```

```r
cat("Saved: Heatmap/7_pc1_vs_pc2.tiff\n")
```

```
## Saved: Heatmap/7_pc1_vs_pc2.tiff
```

```r
cat("  PART 4 (DCCM/PCA) Analysis complete!\n\n")
```

```
##   PART 4 (DCCM/PCA) Analysis complete!
```

```r
# ====================================================================
# === PART 5: Time Evolution Analysis ================================
# ====================================================================
cat("--- PART 5: Time Evolution Analysis ---\n")
```

```
## --- PART 5: Time Evolution Analysis ---
```

```r
# STEP 5.1 - Read and Prepare Time Evolution Data ---------------------
cat("Reading time evolution data file: Samip_analysis.tab...\n")
```

```
## Reading time evolution data file: Samip_analysis.tab...
```

```r
# Read the data file
data <- read.table("Samip_analysis.tab", header = TRUE, sep = "", stringsAsFactors = FALSE)

# Let's see what our data looks like
cat("Data structure:\n")
```

```
## Data structure:
```

```r
str(data)
```

```
## 'data.frame':    1030 obs. of  27 variables:
##  $ Time.ps.    : chr  "0.000" "100.000" "200.000" "300.000" ...
##  $ Time.ns.    : chr  "0.000" "0.100" "0.200" "0.300" ...
```

```
##  $ CellLengthX : num  112 112 112 112 112 ...
##  $ CellLengthY : num  112 112 112 112 112 ...
##  $ CellLengthZ : num  112 112 112 112 112 ...
##  $ TotalEnergy : num  -2485437 -1886832 -1869976 -1863422 -1859568 ...
##  $ Bond        : num  55545 181835 173058 167321 165441 ...
##  $ Angle       : num  24797 98492 96046 93548 92294 ...
##  $ Dihedral    : num  107440 109600 109626 110113 110081 ...
##  $ Planarity   : num  163 763 864 800 759 ...
##  $ Coulomb     : num  -3130140 -2619322 -2583491 -2566467 -2554932 ...
##  $ VdW         : num  456759 341800 333921 331263 326790 ...
##  $ SurfVdW     : num  41935 42442 42206 42361 42277 ...
##  $ SurfMol     : num  22518 22884 22603 22917 22758 ...
##  $ SurfAcc     : num  21644 22062 21568 21740 21642 ...
##  $ SoluteHBonds: num  286 282 279 283 274 290 294 277 266 271 ...
##  $ SltSlvHBonds: num  792 867 881 851 875 855 852 887 861 864 ...
##  $ Helix       : num  13.6 11.8 13.6 10 11.6 ...
##  $ Sheet       : num  38.9 40.2 38.4 37 39.1 ...
##  $ Turn        : num  13.9 13.9 15 17.5 14.3 ...
##  $ Coil        : num  33.6 33.2 32 35.5 33.2 ...
##  $ Helix310    : num  0 0.909 0.909 0 1.818 ...
##  $ HelixPi     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ RadGyration : num  28.2 28.3 28.3 28.2 28.4 ...
##  $ RMSDCa      : num  0.499 1.486 1.349 1.753 1.653 ...
##  $ RMSDBb      : num  0.536 1.508 1.36 1.761 1.66 ...
##  $ RMSDAll     : num  0.609 1.709 1.628 2.033 2.001 ...
```

```r
# CLEAN THE DATA
# Remove any problematic time values
data$Time.ns <- as.numeric(data$Time.ns)
```

```
## Warning: NAs introduced by coercion
```

```r
data <- data[!is.na(data$Time.ns) & !is.infinite(data$Time.ns), ]

# List of columns that should contain numbers
numeric_columns <- c("RMSDCa", "RadGyration", "TotalEnergy", "Coulomb", "VdW",
                    "SoluteHBonds", "SltSlvHBonds", "CellLengthX", "CellLengthY", "CellLengthZ"
                    "RMSDBb", "RMSDAll", "Helix", "Sheet", "Turn", "Coil", "Bond", "Angle", "Di

# Convert all these columns to numbers
for(col in numeric_columns) {
  if(col %in% colnames(data)) {
    data[[col]] <- as.numeric(as.character(data[[col]]))
  }
}

# Remove any rows with missing values in important columns
data <- data[complete.cases(data[, c("Time.ns", "RMSDCa")]), ]
```

```
# SET UP PLOTTING PARAMETERS
# ============================================================================

# Calculate time breaks for x-axis (every 25 nanoseconds)
max_time <- max(data$Time.ns, na.rm = TRUE)
cat("Simulation duration:", max_time, "nanoseconds\n")
```

```
## Simulation duration: 102.6 nanoseconds
```

```
x_breaks <- seq(0, max_time, by = 25)
cat("X-axis will have ticks at:", x_breaks, "ns\n")
```

```
## X-axis will have ticks at: 0 25 50 75 100 ns
```

```
# DEFINE BEAUTIFUL COLORS FOR OUR PLOTS
# These are deep, professional-looking colors
deep_colors <- c(
  blue = "#003366",         # Deep ocean blue
  green = "#006400",        # Forest green
  red = "#8B0000",          # Dark red
  purple = "#4B0082",       # Royal purple
  orange = "#CC5500",       # Burnt orange
  brown = "#654321",        # Dark brown
  cyan = "#008B8B",         # Deep cyan
  magenta = "#8B008B",      # Deep magenta
  darkblue = "#000080",     # Navy blue
  darkred = "#800000",      # Maroon
  darkgreen = "#006400",    # Dark green
  gold = "#B8860B",         # Goldenrod
  pink = "#8B0A50",         # Deep pink
  teal = "#008080",         # Teal
  violet = "#9400D3"        # Dark violet
)

# CREATE A CUSTOM THEME FOR ALL PLOTS
# This makes all our plots look consistent and professional
custom_theme <- theme_void() +
  theme(
    # Title styling - centered, bold, large
    plot.title = element_text(hjust = 0.5, face = "bold", size = 32, color = "black", margin =

    # Axis titles - bold and large
    axis.title.x = element_text(face = "bold", size = 32, color = "black", margin = margin(t =
    axis.title.y = element_text(face = "bold", size = 32, color = "black", margin = margin(r =

    # Axis numbers - bold and readable
    axis.text.x = element_text(face = "bold", size = 32, color = "black", margin = margin(t = 5
    axis.text.y = element_text(face = "bold", size = 32, color = "black", margin = margin(r = 5
```

```r
    # Legend styling
    legend.title = element_text(face = "bold", size = 32, color = "black"),
    legend.text = element_text(face = "bold", size = 32, color = "black"),
    legend.position = "bottom",
    legend.key.size = unit(1.5, "cm"),
    legend.key.width = unit(2, "cm"),
    legend.margin = margin(t = 10, b = 10),

    # Clean background - no grid lines or colors
    panel.background = element_blank(),
    plot.background = element_blank(),
    panel.grid = element_blank(),

    # Only show axis lines
    axis.line.x = element_line(color = "black", linewidth = 1.5),
    axis.line.y = element_line(color = "black", linewidth = 1.5),
    axis.ticks = element_line(color = "black", linewidth = 1.5),
    axis.ticks.length = unit(0.2, "cm")
  )

# HELPER FUNCTIONS FOR PLOTTING
# ============================================================================

# Function to calculate good y-axis limits that center the data nicely
calculate_y_limits <- function(y_data, padding_factor = 0.2) {
  y_range <- range(y_data, na.rm = TRUE)
  y_span <- y_range[2] - y_range[1]
  padding <- y_span * padding_factor

  c(y_range[1] - padding, y_range[2] + padding)
}

# Function to create simple line plots (one line per plot)
create_plot <- function(data, y_col, title, y_lab, color) {
  # Clean the data
  y_data <- as.numeric(data[[y_col]])
  valid_data <- !is.na(y_data) & !is.infinite(y_data)

  if(sum(valid_data) > 0) {
    plot_data <- data.frame(Time = data$Time.ns[valid_data], Y = y_data[valid_data])
    y_limits <- calculate_y_limits(plot_data$Y)

    # Create the plot
    ggplot(plot_data, aes(x = Time, y = Y)) +
      geom_line(color = color, linewidth = 1.5) +
      scale_x_continuous(breaks = x_breaks) +
      scale_y_continuous(limits = y_limits) +
```

```r
      labs(title = title, x = "Time (ns)", y = y_lab) +
      custom_theme +
      theme(plot.margin = margin(20, 20, 25, 20))
  } else {
    # Show message if no valid data
    ggplot() +
      geom_text(aes(x = 0.5, y = 0.5, label = "No valid data"), size = 12) +
      labs(title = paste(title, "- No Data"), x = "Time (ns)", y = y_lab) +
      custom_theme +
      theme(plot.margin = margin(20, 20, 25, 20))
  }
}

# Function to create multi-line plots (multiple lines on same plot)
create_multi_line_plot <- function(data, y_cols, colors, labels, title, y_lab, nrow_legend = NU
  # Prepare data for plotting
  plot_data <- data.frame(Time = data$Time.ns)

  for(i in seq_along(y_cols)) {
    plot_data[[labels[i]]] <- as.numeric(data[[y_cols[i]]])
  }

  # Reshape data from wide to long format (needed for ggplot)
  plot_data_long <- plot_data %>%
    tidyr::pivot_longer(cols = -Time, names_to = "Variable", values_to = "Value") %>%
    filter(!is.na(Value) & !is.infinite(Value))

  if(nrow(plot_data_long) > 0) {
    # Calculate y limits for all variables
    all_values <- unlist(plot_data_long$Value)
    y_limits <- calculate_y_limits(all_values)

    # Create the multi-line plot
    ggplot(plot_data_long, aes(x = Time, y = Value, color = Variable)) +
      geom_line(linewidth = 1.5) +
      scale_x_continuous(breaks = x_breaks) +
      scale_y_continuous(limits = y_limits) +
      scale_color_manual(values = setNames(colors, labels)) +
      labs(title = title, x = "Time (ns)", y = y_lab, color = "") +
      guides(color = guide_legend(override.aes = list(linewidth = 3), nrow = nrow_legend)) +
      custom_theme +
      theme(
        plot.margin = margin(20, 20, 30, 20),
        legend.text = element_text(face = "bold", size = 32, color = "black"),
        legend.position = "bottom"
      )
  } else {
```

```r
    ggplot() +
      geom_text(aes(x = 0.5, y = 0.5, label = "No valid data"), size = 12) +
      labs(title = paste(title, "- No Data"), x = "Time (ns)", y = y_lab) +
      custom_theme +
      theme(plot.margin = margin(20, 20, 30, 20))
  }
}

# Special function for Hydrogen Bonds plot
create_hbond_plot <- function(data, title, y_lab) {
  # Prepare hydrogen bonds data
  plot_data <- data.frame(
    Time = data$Time.ns,
    SoluteHBonds = as.numeric(data$SoluteHBonds),
    SltSlvHBonds = as.numeric(data$SltSlvHBonds)
  )

  plot_data_long <- plot_data %>%
    tidyr::pivot_longer(cols = c(SoluteHBonds, SltSlvHBonds),
                 names_to = "Variable",
                 values_to = "Value") %>%
    filter(!is.na(Value) & !is.infinite(Value))

  if(nrow(plot_data_long) > 0) {
    # Set fixed y-axis limits (0 to 1500)
    y_limits <- c(0, 1500)

    # Ensure proper order of legend items
    plot_data_long$Variable <- factor(plot_data_long$Variable,
                                  levels = c("SoluteHBonds", "SltSlvHBonds"))

    # Create the plot with legend inside
    ggplot(plot_data_long, aes(x = Time, y = Value, color = Variable)) +
      geom_line(linewidth = 1.5) +
      scale_x_continuous(breaks = x_breaks) +
      scale_y_continuous(limits = y_limits) +
      scale_color_manual(
        values = c("SoluteHBonds" = deep_colors[["orange"]],
                   "SltSlvHBonds" = deep_colors[["brown"]]),
        labels = c("SoluteHBonds" = "Solute H-Bonds",
                   "SltSlvHBonds" = "Solute-Solvent H-Bonds")
      ) +
      labs(title = title, x = "Time (ns)", y = y_lab, color = "") +
      guides(color = guide_legend(
        override.aes = list(linewidth = 3),
        nrow = 2
      )) +
```

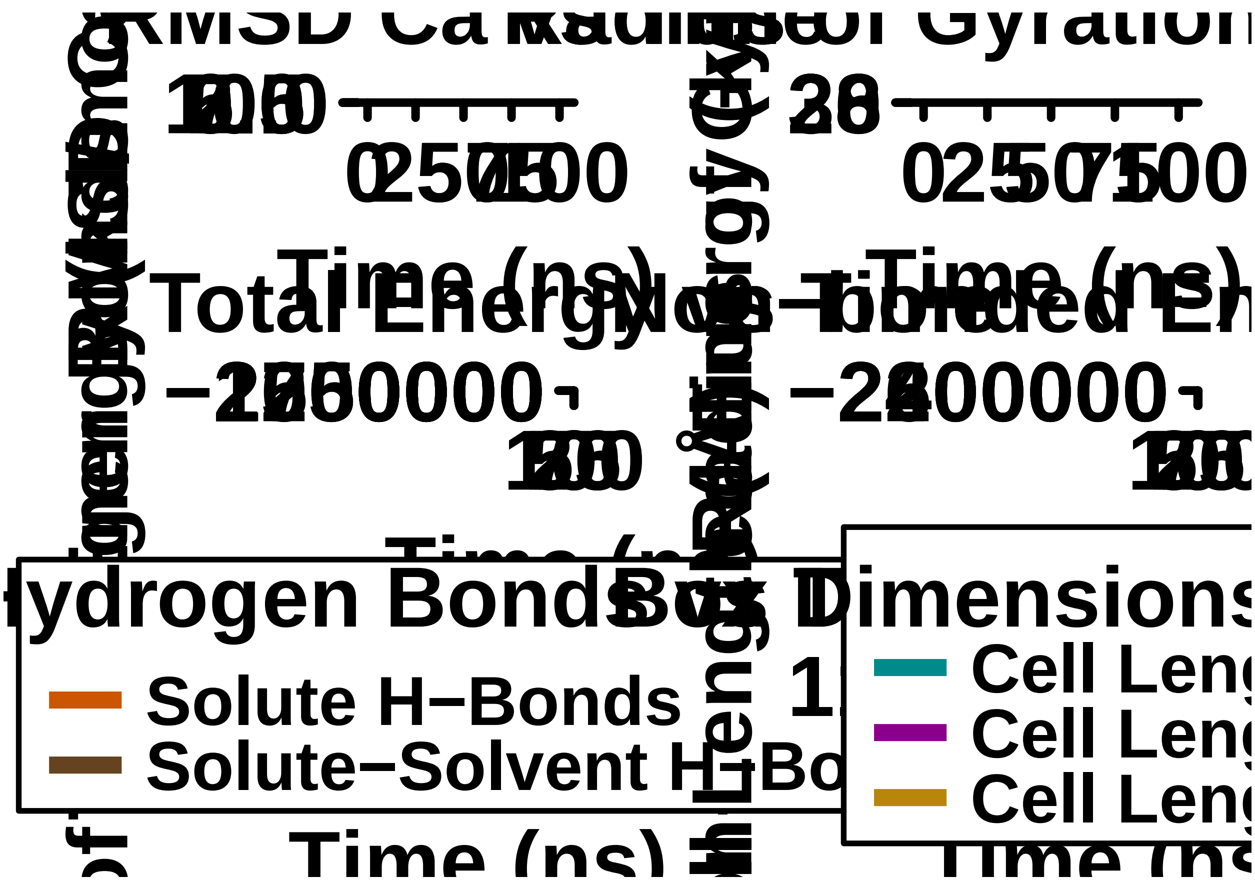
```r
      custom_theme +
      theme(
        plot.margin = margin(20, 20, 20, 20),
        legend.text = element_text(face = "bold", size = 26, color = "black"),
        legend.position = c(0.65, 0.85),  # Legend position inside plot
        legend.background = element_rect(fill = "white", color = "black", linewidth = 1),
        legend.key = element_rect(fill = "white"),
        legend.margin = margin(5, 8, 5, 8),
        legend.spacing.y = unit(0.2, "cm"),
        legend.key.height = unit(0.7, "cm"),
        legend.key.width = unit(1.2, "cm")
      )
  } else {
    ggplot() +
      geom_text(aes(x = 0.5, y = 0.5, label = "No valid data"), size = 12) +
      labs(title = paste(title, "- No Data"), x = "Time (ns)", y = y_lab) +
      custom_theme +
      theme(plot.margin = margin(20, 20, 30, 20))
  }
}

# Special function for Box Dimensions plot
create_box_dimensions_plot <- function(data, title, y_lab) {
  # Prepare box dimensions data
  plot_data <- data.frame(
    Time = data$Time.ns,
    CellLengthX = as.numeric(data$CellLengthX),
    CellLengthY = as.numeric(data$CellLengthY),
    CellLengthZ = as.numeric(data$CellLengthZ)
  )

  plot_data_long <- plot_data %>%
    tidyr::pivot_longer(cols = c(CellLengthX, CellLengthY, CellLengthZ),
                 names_to = "Variable",
                 values_to = "Value") %>%
    filter(!is.na(Value) & !is.infinite(Value))

  if(nrow(plot_data_long) > 0) {
    # Calculate automatic y limits
    all_values <- unlist(plot_data_long$Value)
    y_limits <- calculate_y_limits(all_values)

    # Create the plot with legend inside
    ggplot(plot_data_long, aes(x = Time, y = Value, color = Variable)) +
      geom_line(linewidth = 1.5) +
      scale_x_continuous(breaks = x_breaks) +
      scale_y_continuous(limits = y_limits) +
```

```r
      scale_color_manual(
        values = c("CellLengthX" = deep_colors[["cyan"]],
                   "CellLengthY" = deep_colors[["magenta"]],
                   "CellLengthZ" = deep_colors[["gold"]]),
        labels = c("CellLengthX" = "Cell Length X",
                   "CellLengthY" = "Cell Length Y",
                   "CellLengthZ" = "Cell Length Z")
      ) +
      labs(title = title, x = "Time (ns)", y = y_lab, color = "") +
      guides(color = guide_legend(
        override.aes = list(linewidth = 3),
        nrow = 3
      )) +
      custom_theme +
      theme(
        plot.margin = margin(20, 20, 20, 20),
        legend.text = element_text(face = "bold", size = 26, color = "black"),
        legend.position = c(0.65, 0.85),  # Legend position inside plot
        legend.background = element_rect(fill = "white", color = "black", linewidth = 1),
        legend.key = element_rect(fill = "white"),
        legend.margin = margin(5, 8, 5, 8),
        legend.spacing.y = unit(0.2, "cm"),
        legend.key.height = unit(0.7, "cm"),
        legend.key.width = unit(1.2, "cm")
      )
  } else {
    ggplot() +
      geom_text(aes(x = 0.5, y = 0.5, label = "No valid data"), size = 12) +
      labs(title = paste(title, "- No Data"), x = "Time (ns)", y = y_lab) +
      custom_theme +
      theme(plot.margin = margin(20, 20, 30, 20))
  }
}

# CREATE THE MAIN PLOTS
# ===============================================================================

cat("Creating time evolution plots...\n")
```

```
## Creating time evolution plots...
```

```r
# 1. RMSD of Calcium atoms vs Time
# RMSD measures how much atoms move from their starting positions
p1 <- create_plot(data, "RMSDCa", "RMSD Ca vs Time", "RMSD Ca (Å)", deep_colors[["blue"]])

# 2. Radius of Gyration vs Time
# Measures how compact the protein structure is
p2 <- create_plot(data, "RadGyration", "Radius of Gyration vs Time", "Radius of Gyration (Å)",
```

```
# 3. Total Energy vs Time
# Shows the energy stability of the simulation
p3 <- create_plot(data, "TotalEnergy", "Total Energy vs Time", "Total Energy (kJ/mol)", deep_co

# 4. Non-bonded interaction energy
# Combine Coulomb and Van der Waals energies
data$NonBondedEnergy <- as.numeric(data$Coulomb) + as.numeric(data$VdW)
p4 <- create_plot(data, "NonBondedEnergy", "Non-bonded Energy vs Time", "Non-bonded Energy (kJ/

# 5. Hydrogen bonds vs Time
# Shows hydrogen bonding patterns during simulation
p5 <- create_hbond_plot(data, "Hydrogen Bonds vs Time", "Number of Hydrogen Bonds")

# 6. Box Dimensions vs Time
# Shows how the simulation box size changes
p6 <- create_box_dimensions_plot(data, "Box Dimensions vs Time", "Cell Length (Å)")

# DISPLAY AND SAVE MAIN PLOTS
# ===============================================================================

# Arrange all 6 plots in a 2-column grid
grid.arrange(p1, p2, p3, p4, p5, p6, ncol = 2)
```



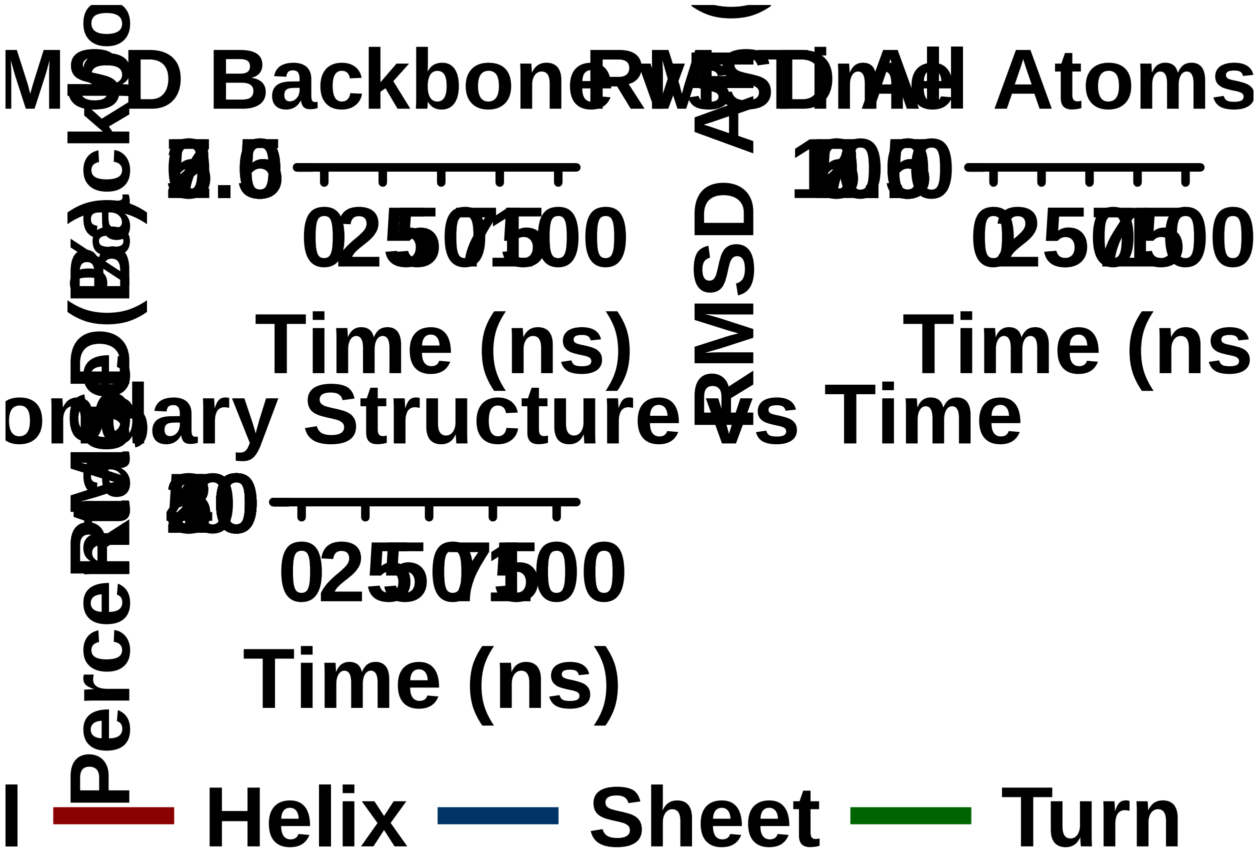
```r
# Save each plot as high-quality TIFF files in the new folder
cat("Saving main time evolution plots as TIFF files...\n")
```

```
## Saving main time evolution plots as TIFF files...
```

```r
ggsave(file.path(time_output_dir, "RMSD_Ca_vs_Time.tiff"), p1, width = 10, height = 8, dpi = 30
ggsave(file.path(time_output_dir, "RadGyration_vs_Time.tiff"), p2, width = 10, height = 8, dpi
ggsave(file.path(time_output_dir, "TotalEnergy_vs_Time.tiff"), p3, width = 10, height = 8, dpi
ggsave(file.path(time_output_dir, "NonBondedEnergy_vs_Time.tiff"), p4, width = 10, height = 8,
ggsave(file.path(time_output_dir, "HydrogenBonds_vs_Time.tiff"), p5, width = 10, height = 8, dp
ggsave(file.path(time_output_dir, "BoxDimensions_vs_Time.tiff"), p6, width = 10, height = 8, dp

# CREATE ADDITIONAL ANALYSIS PLOTS
# ===========================================================================

cat("Creating additional time evolution plots...\n")
```

```
## Creating additional time evolution plots...
```

```r
# 7. RMSD of Backbone atoms vs Time
p7 <- create_plot(data, "RMSDBb", "RMSD Backbone vs Time", "RMSD Backbone (Å)", deep_colors[["d

# 8. RMSD of All atoms vs Time
p8 <- create_plot(data, "RMSDAll", "RMSD All Atoms vs Time", "RMSD All (Å)", deep_colors[["viol

# 9. Secondary Structure vs Time
# Shows changes in protein structure (helix, sheet, turn, coil)
p9 <- create_multi_line_plot(data,
                             y_cols = c("Helix", "Sheet", "Turn", "Coil"),
                             colors = c(deep_colors[["red"]], deep_colors[["blue"]], deep_colors
                             labels = c("Helix", "Sheet", "Turn", "Coil"),
                             title = "Secondary Structure vs Time",
                             y_lab = "Percentage (%)")

# DISPLAY AND SAVE ADDITIONAL PLOTS
# ===========================================================================

# Arrange the additional plots
grid.arrange(p7, p8, p9, ncol = 2)
```




```r
# Save additional plots as TIFF files in the new folder
cat("Saving additional time evolution plots as TIFF files...\n")
```

```
## Saving additional time evolution plots as TIFF files...
```

```r
ggsave(file.path(time_output_dir, "RMSD_Backbone_vs_Time.tiff"), p7, width = 10, height = 8, dp
ggsave(file.path(time_output_dir, "RMSD_All_vs_Time.tiff"), p8, width = 10, height = 8, dpi = 3
ggsave(file.path(time_output_dir, "Secondary_Structure_vs_Time.tiff"), p9, width = 10, height =

cat("  PART 5 (Time Evolution) Analysis complete!\n\n")
```

```
##   PART 5 (Time Evolution) Analysis complete!
```

```r
# ===================================================================
# === FINAL WRAP-UP =================================================
# ===================================================================

cat("\n\n#####################################################\n")
```

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

```r
cat("            COMPREHENSIVE MD ANALYSIS COMPLETE!  \n")
```

```
##              COMPREHENSIVE MD ANALYSIS COMPLETE!
```

```r
cat("#####################################################\n")
```

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

```r
cat("All 5 analysis parts completed successfully:\n")
```

```
## All 5 analysis parts completed successfully:
```

```r
cat("1.  RMSF Analysis - Protein flexibility\n")
```

```
## 1.  RMSF Analysis - Protein flexibility
```

```r
cat("2.  Secondary Structure Analysis - Structural changes\n")
```

```
## 2.  Secondary Structure Analysis - Structural changes
```

```r
cat("3.  Residue Contacts Analysis - Stability assessment\n")
```

```
## 3.  Residue Contacts Analysis - Stability assessment
```

```r
cat("4.  DCCM/PCA Analysis - Collective motions\n")
```

```
## 4.  DCCM/PCA Analysis - Collective motions
```

```r
cat("5.  Time Evolution Analysis - Simulation dynamics\n\n")
```

```
## 5.  Time Evolution Analysis - Simulation dynamics
```

```r
cat(" OUTPUT FOLDERS CREATED:\n")
```

```
##  OUTPUT FOLDERS CREATED:
```

```r
cat("- Working Directory:", getwd(), "\n")
```

```
## - Working Directory: C:/Users/SHAHIN/Desktop/SAMIP
```

```r
cat("- Secondary Structure Plots:", ss_output_dir, "\n")
```

```
## - Secondary Structure Plots: ss_plots
```

```r
cat("- Contact Stability Plots:", contact_output_dir, "\n")
```

```
## - Contact Stability Plots: protein_stability_plots
```

```r
cat("- DCCM/PCA Plots:", dccm_output_dir, "\n")
```

```
## - DCCM/PCA Plots: Heatmap
```

```r
cat("- Time Evolution Plots:", time_output_dir, "\n\n")
```

```
## - Time Evolution Plots: Time_Evolution_Plots
```

```r
cat(" TOTAL PLOTS GENERATED:\n")
```

```
##  TOTAL PLOTS GENERATED:
```

```r
cat("- RMSF Analysis: 2 comprehensive plots + data file\n")
```

```
## - RMSF Analysis: 2 comprehensive plots + data file
cat("- Secondary Structure: 5 detailed plots\n")

## - Secondary Structure: 5 detailed plots
cat("- Residue Contacts: 4-6 stability plots\n")

## - Residue Contacts: 4-6 stability plots
cat("- DCCM/PCA: 8 correlation and PCA plots\n")

## - DCCM/PCA: 8 correlation and PCA plots
cat("- Time Evolution: 9 time-dependent plots\n")

## - Time Evolution: 9 time-dependent plots
cat("Total: ~30+ publication-ready plots\n\n")

## Total: ~30+ publication-ready plots
cat("  VISUALIZATION FEATURES:\n")

##   VISUALIZATION FEATURES:
cat("- All axis labels and numbers: Bold, Black, 16pt\n")

## - All axis labels and numbers: Bold, Black, 16pt
cat("- Consistent figure sizes across all plots\n")

## - Consistent figure sizes across all plots
cat("- Professional color schemes\n")

## - Professional color schemes
cat("- High-resolution TIFF format (300 DPI)\n")

## - High-resolution TIFF format (300 DPI)
cat("- NO GRID BOX in background - Clean publication-ready style\n")

## - NO GRID BOX in background - Clean publication-ready style
cat("- Black x,y axes only\n")

## - Black x,y axes only
cat("- Publication-ready quality\n\n")

## - Publication-ready quality
cat("  ANALYSIS COMPLETE - Ready for scientific publication!\n")

##   ANALYSIS COMPLETE - Ready for scientific publication!
```

```r
cat("#####################################################\n")
```

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