Below is a **User Manual** for `TritJS-CISA`, based on the provided `.cweb` document dated March 01, 2025. This manual is designed to guide users through installation, usage, and understanding of the ternary calculator's features, tailored for the Cybersecurity and Infrastructure Security Agency (CISA).

It covers compilation, command syntax, examples, and troubleshooting, reflecting the tool's capabilities in arithmetic, scientific functions, statistics, scripting, and state management—all within an ASCII-only CLI.

TritJS-CISA User Manual

Version: March 01, 2025

Overview

`TritJS-CISA` is an advanced ternary (base-3) scientific calculator developed for the Cybersecurity and Infrastructure Security Agency (CISA). It surpasses traditional calculators like the TI-82 by offering ternary arithmetic, AI-driven statistical analysis, basic scripting, and persistent state management, all optimized for cybersecurity tasks and educational purposes. Built in C as a `.cweb` literate program, it emulates ternary computation on binary hardware, providing a secure and interactive ASCII-only command-line interface (CLI).

Key Features

- **Ternary Computation**: Operates in base-3 (trits: 0, 1, 2) for arithmetic and scientific functions.
- **Memory-Mapped Files**: Uses `mmap` for efficient trit array handling, with memory usage visualization.
- **Security**: Logs errors to `/var/log/tritjs_cisa.log` and ensures memory safety.
- **Complex Arithmetic**: Supports `TritBigInt`, `TritFloat`, and `TritComplex` data types.
- **Scientific Functions**: Includes exponentiation, roots, logarithms, trigonometry, and factorials.
- **AI-Driven Stats**: Computes mean, mode, and median with optimized sorting (Quicksort or Mergesort).
- **Scripting**: Allows basic automation with `PROG` and `RUN` commands.

- **State Management**: Saves and loads state to `.trit` files (MIME type: `application/x-tritjs-cisa`).
- **Interactive CLI**: Features history, variables (A-Z), and a command-driven interface.

Installation

Prerequisites

- **Operating System**: Linux (or compatible UNIX-like system with `mmap` support).
- **Compiler**: GCC (GNU Compiler Collection).
- **Libraries**: Standard C library (`-lm` for math functions).
- **Tools**: `cweave` and `ctangle` for processing `.cweb` files (install via TeX Live or similar).

Compilation Steps

1. **Obtain the Source**:
   - Acquire the `tritjs_cisa.cweb` file (e.g., from a CISA-provided repository or this manual's source).
2. **Generate Documentation (Optional)**:`bash`
   `cweave tritjs_cisa.cweb`
   - Produces `tritjs_cisa.tex` for LaTeX processing into a readable document.
3. **Extract C Code**:`bash`
   `ctangle tritjs_cisa.cweb`
   - Outputs `tritjs_cisa.c`, the compilable C source.
4. **Compile the Program**:`bash`
   `gcc -o tritjs_cisa tritjs_cisa.c -lm`
   - Links the math library (`-lm`) and creates the executable `tritjs_cisa`.

5. **Run the Calculator**:`bash ./tritjs_cisa`
    - Launches the CLI with a welcome message: `=== TritJS-CISA Ternary Calculator ===`.

Notes

- Ensure write permissions for `/var/log/tritjs_cisa.log` (or `/tmp/tritjs_cisa.log` as a fallback) for audit logging.
- Compilation requires ~1MB of memory due to `MAX_MMAP_SIZE`.

Usage

Starting the Calculator

Upon running `./tritjs_cisa`, you'll see:

```
=== TritJS-CISA Ternary Calculator ===
Type 'help' for commands
>
```

Enter commands at the > prompt. All inputs are in ternary (digits 0, 1, 2), and negative numbers use a leading – (e.g., –12).

Command Syntax

Commands follow the format: `<operation> <arg1> [arg2]`. Arguments are optional unless specified. Examples:

- `add 12 2`
- `A=12`
- `save work.trit`

Command Categories

1. Arithmetic Operations

Perform calculations in ternary. Two arguments are required except for `fact`.

Command	Description	Example	Output
`add <a> <b>`	Adds `a` and `b`	`add 12 2`	`21`
`sub <a> <b>`	Subtracts `b` from `a`	`sub 21 12`	`2`
`mul <a> <b>`	Multiplies `a` by `b`	`mul 11 10`	`110`
`div <a> <b>`	Divides `a` by `b` (3 trits)	`div 21 2`	`10.1 r 1`
`pow <a> <b>`	Raises `a` to power `b`	`pow 2 2`	`11`
`fact <a>`	Computes factorial of `a`	`fact 2`	`2`

- **Notes**:
  - `div` outputs quotient and remainder (e.g., `10.1 r 1` means quotient 10.1, remainder 1).
  - `pow` and `fact` limit exponents/inputs to 1000 and 20, respectively, to prevent overflow.

2. Scientific Functions

Compute advanced mathematical functions. One argument is required except for `pi`.

Command	Description	Example	Output
`sqrt <a>`	Square root of `a` (3 trits)	`sqrt 12`	`11.1`
`log3 <a>`	Base-3 logarithm of `a`	`log3 100`	`2.002`
`sin <a>`	Sine of `a` (radians × π/10)	`sin 2`	`0.2`
`cos <a>`	Cosine of `a` (radians × π/10)	`cos 2`	`1.2`
`tan <a>`	Tangent of `a` (radians × π/10)	`tan 2`	`0.1`

`pi`	Ternary approximation of π	`pi`	`10010221`

- **Notes**:
  - Trigonometric functions approximate results with 3 fractional trits.
  - Negative inputs to `sqrt` yield complex results (e.g., `sqrt -12` outputs `0 11.1i`).

3. Statistics

Analyze history data with AI-optimized sorting.

Command	Description	Example	Output Example
`stats [quick	merge]`	Computes mean, mode, median	`stats`

- **Options**:
  - `quick`: Uses Quicksort for <10 trits.
  - `merge`: Uses Mergesort for ≥10 trits or balanced distributions.
  - Omit: Auto-selects based on history size and distribution.
- **Notes**: Requires history entries (e.g., prior results like `2`, `11`, `102`).

4. Scripting

Automate tasks with basic scripts.

Command	Description	Example
`PROG <name> {<cmds>}`	Defines a script	`PROG SUM {add A 1; A=A}`
`RUN <name>`	Executes a named script	`RUN SUM`

- **Syntax**:
  - Commands within `{}` are separated by `;`.
  - Supports `IF <cond> THEN <cmd>` and `FOR <var> <start> <end> <cmd>`.
- **Example**:

```
> A=1
```

- `> PROG LOOP {FOR I 1 2 {add A I; A=A}}`
- `Script 'LOOP' defined`
- `> RUN LOOP`
- `Script 'LOOP' executed`
- `> recall 0`
- `10`

5. Storage

Manage calculator state.

Command	Description	Example
`save <file>`	Saves state to `.trit` file	`save work.trit`
`load <file>`	Loads state from `.trit` file	`load work.trit`

- **Notes**:
  - Saves history, variables, and scripts.
  - Files use MIME type `application/x-tritjs-cisa`.

6. General Commands

Control the calculator environment.

Command	Description	Example
`help`	Displays this command list	`help`
`quit`	Exits the calculator	`quit`
`recall <n>`	Recalls nth last result (0=latest)	`recall 0`
`<var>=<value>`	Sets variable (A-Z)	`A=12`
`clear`	Resets history, variables, scripts	`clear`

- **Notes**:
  - History stores up to 10 entries.
  - Variables persist until `clear` or overwritten.

Examples

Basic Arithmetic

```
> add 12 2
21
> sub 21 12
2
> mul 11 2
22
> div 21 2
10.1 r 1
```

Scientific Calculations

```
> sqrt 12
11.1
> log3 100
2.002
> sin 2
0.2
> pi
10010221
```

Statistics

```
> add 1 1
2
> mul 2 1
2
> add 11 2
20
> stats
Mean: 1.33 | Mode: 2 | Median: 1.00 | Total Trits: 6 | Sort: merge
```

Scripting

```
> A=1
A stored
> PROG INCR {add A 1; A=A}
Script 'INCR' defined
> RUN INCR
Script 'INCR' executed
> recall 0
2
```

State Management

```
> A=12
A stored
> save calc.trit
State saved to calc.trit
> clear
```

```
History, variables, and scripts cleared
> load calc.trit
State loaded from calc.trit
> recall 0
12
```

Troubleshooting

Common Errors

- **"Invalid input (trits 0-2 only)"**:
  - Cause: Used digits outside 0, 1, 2 (e.g., `add 13 2`).
  - Fix: Ensure all inputs are ternary (e.g., `add 12 2`).
- **"Division by zero"**:
  - Cause: Attempted `div <a> 0`.
  - Fix: Use a non-zero divisor.
- **"Overflow detected"**:
  - Cause: Exceeded limits (e.g., `pow 2 1001`, `fact 21`).
  - Fix: Keep exponents ≤ 1000, factorials ≤ 20.
- **"Could not open file"**:
  - Cause: Invalid file path or permissions for `save`/`load`.
  - Fix: Check path (e.g., `./work.trit`) and permissions.

Logs

- Errors are logged to `/var/log/tritjs_cisa.log` (or `/tmp/tritjs_cisa.log` if inaccessible).

- Example entry:
  `[Sat Mar 01 12:00:00 2025] Error 3: Division by zero in tritjs_divide_big`
- Check logs for detailed diagnostics.

Technical Notes

Limits

- **Memory**: 1MB (`MAX_MMAP_SIZE`) for mapped arrays.
- **History**: 10 entries (`MAX_HISTORY`).
- **Scripts**: 10 scripts, 50 commands each (`MAX_SCRIPT_NAME`, `MAX_SCRIPT_CMDS`).
- **Precision**: 10 trits for fractional results.

Security

- Audit logging ensures traceability of errors.
- Memory safety is enforced via `tritbig_free` and `unmap_trits`.

MIME Type

- `.trit` files use `application/x-tritjs-cisa`, storing state in plain text.

Support

For issues or enhancements, contact CISA's technical support team or refer to the source documentation (`tritjs_cisa.tex`) generated via `cweave`.

This manual provides a comprehensive guide to using `TritJS-CISA` effectively for CISA's cybersecurity and educational needs. Experiment with commands and explore ternary computation!

@* TritJS-CISA: Enhanced Ternary Cybersecurity Tool for CISA.
This document defines \.{TritJS-CISA}, an advanced ternary (base-3) cybersecurity tool designed for the Cybersecurity and Infrastructure Security Agency (CISA) to protect critical infrastructure. It surpasses the TI-82 with ternary statistics optimized via AI-driven sorting, scripting, encrypted save/load, real-time networking, and certification testing in an ASCII-only CLI. Built in C as a \.{.cweb} literate program, it emulates ternary computation on binary hardware, enhancing security, scalability, and education. Enhancements include:

- **Scalable Memory**: Dynamic allocation up to 100MB, 10,000 history entries.
- **Security**: AES-256 encrypted storage and logs.
- **Real-Time**: Socket-based network input for live data.
- **Complex Arithmetic**: \.{TritFloat}, \.{TritBigInt}, \.{TritComplex}.
- **Scientific Functions**: Exponentiation, roots, logarithms, trigonometry, factorials.
- **CLI**: History, variables, stats, scripting, save/load, testing, and learning.
- **MIME Type**: \.{application/x-tritjs-cisa} for \.{.trit} files.

Version: March 01, 2025, with persistent storage and CISA enhancements.

@*1 Usage and Documentation.
Compile and run:
- \.{cweave tritjs_cisa.cweb} → \.{tritjs_cisa.tex}.
- \.{ctangle tritjs_cisa.cweb} → \.{tritjs_cisa.c}.
- \.{gcc -o tritjs_cisa tritjs_cisa.c -lm -lpthread -lcrypto}.
- \.{./tritjs_cisa}.

Commands: \.{<operation> <arg1> [arg2]}, e.g., \.{add 12 2}, \.{save work.trit secret}.
Operations:
- Arithmetic: \.{add/a}, \.{sub/s}, \.{mul/m}, \.{div/d}, \.{pow/p}, \.{fact/f}
- Scientific: \.{sqrt}, \.{log3}, \.{sin}, \.{cos}, \.{tan}, \.{pi}
- Stats: \.{stats [quick|merge]} (auto-selects if unspecified)
- Scripting: \.{PROG <name> { <cmds> }}, \.{RUN/r <name>}
- Storage: \.{save/sv <file> <key>}, \.{load/ld <file> <key>}, \.{export/json <file>}
- Networking: \.{NET <ip> <port>}
- Education: \.{LEARN <topic>}, \.{TEST <week>}, \.{CHECK <qnum> <ans>}, \.{SCORE}
- General: \.{help/h}, \.{quit/q}, \.{recall/rc <n>}, \.{clear/cl}, \.{<var>=<value>}

Inputs are base-3 (trits: 0, 1, 2). Save/load uses encrypted \.{.trit} files.

@*1 Implementation.
@c
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>
#include <time.h>
#include <pthread.h>
#include <openssl/aes.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define TRIT_MAX 3
#define MAX_MMAP_SIZE (100 * 1024 * 1024) /* 100MB */
#define MAX_DISPLAY_WIDTH 50
#define MAX_HISTORY 10000
#define MAX_VAR_NAME 2
#define MAX_SCRIPT_NAME 20
#define MAX_SCRIPT_CMDS 1000
#define MAX_FILENAME 256
#define MAX_QUESTIONS 40
#define QUESTIONS_PER_WEEK 5
#define AES_KEYLEN 32 /* AES-256 */
#define MAX_INPUT 256

typedef int Trit;
typedef struct {
 int sign;
 Trit* digits;
 int len;
 int is_mapped;
 int fd;
 char tmp_path[32];
} TritBigInt;
```

```
typedef struct {
 int sign;
 Trit* integer;
 Trit* fraction;
 int i_len, f_len;
 int i_mapped, f_mapped;
 int i_fd, f_fd;
 char i_tmp_path[32];
 char f_tmp_path[32];
} TritFloat;

typedef struct {
 TritFloat real;
 TritFloat imag;
} TritComplex;

typedef struct {
 TritFloat quotient;
 TritFloat remainder;
} TritDivResult;

typedef struct {
 char name[MAX_SCRIPT_NAME];
 char commands[MAX_SCRIPT_CMDS][MAX_INPUT];
 int cmd_count;
} Script;

/* Global state */
static long total_mapped_bytes = 0;
static int operation_steps = 0;
static char* history[MAX_HISTORY] = {0};
static int history_count = 0;
static TritBigInt* variables[26] = {0};
static Script scripts[100] = {0};
static int script_count = 0;
static int test_active = 0;
static int test_answers[MAX_QUESTIONS] = {0};
static int test_score = 0;
static int questions_answered = 0;
```

```
static pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

/* Function Prototypes */
TritError tritjs_add_big(TritBigInt* a, TritBigInt* b, TritBigInt** result);
TritError save_state_encrypted(const char* filename, const char* key);
TritError load_state_encrypted(const char* filename, const char* key);
TritError net_listen(const char* ip, int port);
TritError export_json(const char* filename);
void display_test_question(int qnum);
int check_test_answer(int qnum, int user_answer);
void learn_topic(const char* topic);

/* Quicksort */
void quicksort(int* arr, int low, int high) {
 if (low < high) {
 int pivot = arr[high];
 int i = low - 1;
 for (int j = low; j < high; j++) {
 if (arr[j] <= pivot) {
 i++;
 int temp = arr[i];
 arr[i] = arr[j];
 arr[j] = temp;
 }
 }
 int temp = arr[i + 1];
 arr[i + 1] = arr[high];
 arr[high] = temp;
 int pi = i + 1;
 quicksort(arr, low, pi - 1);
 quicksort(arr, pi + 1, high);
 }
}

/* Mergesort */
void merge(int* arr, int l, int m, int r) {
 int n1 = m - l + 1, n2 = r - m;
 int* L = malloc(n1 * sizeof(int));
 int* R = malloc(n2 * sizeof(int));
```

```
 if (!L || !R) {
 free(L); free(R);
 log_error(TRIT_ERR_MEM, "merge");
 return;
 }
 for (int i = 0; i < n1; i++) L[i] = arr[l + i];
 for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];
 int i = 0, j = 0, k = l;
 while (i < n1 && j < n2) {
 if (L[i] <= R[j]) arr[k++] = L[i++];
 else arr[k++] = R[j++];
 }
 while (i < n1) arr[k++] = L[i++];
 while (j < n2) arr[k++] = R[j++];
 free(L);
 free(R);
}

void mergesort(int* arr, int l, int r) {
 if (l < r) {
 int m = l + (r - l) / 2;
 mergesort(arr, l, m);
 mergesort(arr, m + 1, r);
 merge(arr, l, m, r);
 }
}

void display_memory_and_stats(const char* operation, const char* sort_method) {
 pthread_mutex_lock(&mutex);
 int bar_length = (int)((total_mapped_bytes * MAX_DISPLAY_WIDTH) / MAX_MMAP_SIZE);
 if (bar_length > MAX_DISPLAY_WIDTH) bar_length = MAX_DISPLAY_WIDTH;

 double mean = 0.0;
 int trit_counts[TRIT_MAX] = {0}, max_count = 0, mode = -1, total_trits = 0;
 int* all_trits = NULL;
 for (int i = 0; i < history_count; i++) {
 TritBigInt* bi;
 if (parse_trit_string(history[i], &bi) == TRIT_OK) {
 all_trits = realloc(all_trits, (total_trits + bi->len) * sizeof(int));
```

```
if (!all_trits) {
tritbig_free(bi);
log_error(TRIT_ERR_MEM, "display_memory_and_stats");
pthread_mutex_unlock(&mutex);
return;
}
for (int j = 0; j < bi->len; j++) {
mean += bi->digits[j];
trit_counts[bi->digits[j]]++;
all_trits[total_trits + j] = bi->digits[j];
}
total_trits += bi->len;
tritbig_free(bi);
}
}
if (total_trits > 0) mean /= total_trits;
for (int i = 0; i < TRIT_MAX; i++) {
if (trit_counts[i] > max_count) {
max_count = trit_counts[i];
mode = i;
}
}
double median = -1;
if (total_trits > 0) {
if (strcmp(sort_method, "quick") == 0) {
quicksort(all_trits, 0, total_trits - 1);
} else {
mergesort(all_trits, 0, total_trits - 1);
}
if (total_trits % 2 == 0) {
median = (all_trits[total_trits / 2 - 1] + all_trits[total_trits / 2]) / 2.0;
} else {
median = all_trits[total_trits / 2];
}
}
free(all_trits);

printf("\033[2K\033[1A");
printf("Mem: [");
```

```
 for (int i = 0; i < MAX_DISPLAY_WIDTH; i++) printf(i < bar_length ? "" : " ");
 printf("] %ld b | Steps: %d | Mean: %.2f | Mode: %d | Median: %.2f | Sort: %s | Op: %s\n",
 total_mapped_bytes, operation_steps, mean, mode >= 0 ? mode : -1, median >= 0 ? median : -1, sort_method, operation);
 fflush(stdout);
 pthread_mutex_unlock(&mutex);
}

void add_to_history(const char* result_str) {
 pthread_mutex_lock(&mutex);
 if (history_count < MAX_HISTORY) {
 history[history_count] = strdup(result_str);
 if (!history[history_count]) log_error(TRIT_ERR_MEM, "add_to_history");
 else history_count++;
 } else {
 free(history[0]);
 memmove(history, history + 1, (MAX_HISTORY - 1) * sizeof(char*));
 history[MAX_HISTORY - 1] = strdup(result_str);
 if (!history[MAX_HISTORY - 1]) log_error(TRIT_ERR_MEM, "add_to_history");
 }
 pthread_mutex_unlock(&mutex);
}

char* recall_history(int index) {
 pthread_mutex_lock(&mutex);
 char* result = (index < 0 || index >= history_count) ? NULL : strdup(history[history_count - 1 - index]);
 pthread_mutex_unlock(&mutex);
 return result;
}

void store_variable(const char* name, TritBigInt* value) {
 pthread_mutex_lock(&mutex);
 if (strlen(name) != 1 || name[0] < 'A' || name[0] > 'Z') {
 pthread_mutex_unlock(&mutex);
 return;
 }
 int idx = name[0] - 'A';
 if (variables[idx]) tritbig_free(variables[idx]);
 variables[idx] = value;
 pthread_mutex_unlock(&mutex);
```

```
}

TritBigInt* recall_variable(const char* name) {
 pthread_mutex_lock(&mutex);
 TritBigInt* result = (strlen(name) != 1 || name[0] < 'A' || name[0] > 'Z') ? NULL : variables[name[0] - 'A'];
 pthread_mutex_unlock(&mutex);
 return result;
}

void clear_history_and_vars() {
 pthread_mutex_lock(&mutex);
 for (int i = 0; i < history_count; i++) {
 free(history[i]);
 history[i] = NULL;
 }
 history_count = 0;
 for (int i = 0; i < 26; i++) {
 if (variables[i]) {
 tritbig_free(variables[i]);
 variables[i] = NULL;
 }
 }
 for (int i = 0; i < script_count; i++) {
 scripts[i].cmd_count = 0;
 }
 script_count = 0;
 test_active = 0;
 test_score = 0;
 questions_answered = 0;
 memset(test_answers, 0, sizeof(test_answers));
 pthread_mutex_unlock(&mutex);
}

TritError save_state_encrypted(const char* filename, const char* key) {
 FILE* f = fopen(filename, "wb");
 if (!f) {
 log_error(TRIT_ERR_INPUT, "save_state_encrypted: file open");
 return TRIT_ERR_INPUT;
 }
```

```
AES_KEY aes_key;
unsigned char iv[AES_BLOCK_SIZE] = "TritJS-CISA-IV";
if (AES_set_encrypt_key((unsigned char*)key, AES_KEYLEN * 8, &aes_key) < 0) {
fclose(f);
log_error(TRIT_ERR_INPUT, "save_state_encrypted: AES key");
return TRIT_ERR_INPUT;
}

char buffer[1024 * 1024]; /* 1MB buffer */
int len = snprintf(buffer, sizeof(buffer), "# TritJS-CISA Encrypted State\n# History\n");
for (int i = 0; i < history_count; i++) {
len += snprintf(buffer + len, sizeof(buffer) - len, "H: %s\n", history[i]);
}
len += snprintf(buffer + len, sizeof(buffer) - len, "# Variables\n");
for (int i = 0; i < 26; i++) {
if (variables[i]) {
char* str;
if (tritjs_to_string(variables[i], &str) == TRIT_OK) {
len += snprintf(buffer + len, sizeof(buffer) - len, "V: %c=%s\n", 'A' + i, str);
free(str);
}
}
}
len += snprintf(buffer + len, sizeof(buffer) - len, "# Scripts\n");
for (int i = 0; i < script_count; i++) {
len += snprintf(buffer + len, sizeof(buffer) - len, "S: %s\n", scripts[i].name);
for (int j = 0; j < scripts[i].cmd_count; j++) {
len += snprintf(buffer + len, sizeof(buffer) - len, "C: %s\n", scripts[i].commands[j]);
}
}
len += snprintf(buffer + len, sizeof(buffer) - len, "# Test\nT: %d %d %d\n", test_active, test_score, questions_answered);
for (int i = 0; i < MAX_QUESTIONS; i++) {
if (test_answers[i]) {
len += snprintf(buffer + len, sizeof(buffer) - len, "A: %d %d\n", i, test_answers[i]);
}
}

if (len >= sizeof(buffer)) {
```

```
 fclose(f);
 log_error(TRIT_ERR_OVERFLOW, "save_state_encrypted: buffer overflow");
 return TRIT_ERR_OVERFLOW;
 }

 unsigned char* encrypted = malloc(((len + AES_BLOCK_SIZE - 1) / AES_BLOCK_SIZE) * AES_BLOCK_SIZE);
 if (!encrypted) {
 fclose(f);
 log_error(TRIT_ERR_MEM, "save_state_encrypted: encrypt alloc");
 return TRIT_ERR_MEM;
 }
 int outlen = 0;
 AES_cbc_encrypt((unsigned char*)buffer, encrypted, len, &aes_key, iv, AES_ENCRYPT);
 fwrite(encrypted, 1, ((len + AES_BLOCK_SIZE - 1) / AES_BLOCK_SIZE) * AES_BLOCK_SIZE, f);
 free(encrypted);
 fclose(f);
 return TRIT_OK;
}

TritError load_state_encrypted(const char* filename, const char* key) {
 FILE* f = fopen(filename, "rb");
 if (!f) {
 log_error(TRIT_ERR_INPUT, "load_state_encrypted: file open");
 return TRIT_ERR_INPUT;
 }

 AES_KEY aes_key;
 unsigned char iv[AES_BLOCK_SIZE] = "TritJS-CISA-IV";
 if (AES_set_decrypt_key((unsigned char*)key, AES_KEYLEN * 8, &aes_key) < 0) {
 fclose(f);
 log_error(TRIT_ERR_INPUT, "load_state_encrypted: AES key");
 return TRIT_ERR_INPUT;
 }

 fseek(f, 0, SEEK_END);
 long fsize = ftell(f);
 if (fsize > MAX_MMAP_SIZE) {
 fclose(f);
 log_error(TRIT_ERR_OVERFLOW, "load_state_encrypted: file too large");
```

```
return TRIT_ERR_OVERFLOW;
}
fseek(f, 0, SEEK_SET);
unsigned char* encrypted = malloc(fsize);
if (!encrypted) {
fclose(f);
log_error(TRIT_ERR_MEM, "load_state_encrypted: decrypt alloc");
return TRIT_ERR_MEM;
}
fread(encrypted, 1, fsize, f);
fclose(f);

char* buffer = malloc(fsize);
if (!buffer) {
free(encrypted);
log_error(TRIT_ERR_MEM, "load_state_encrypted: buffer alloc");
return TRIT_ERR_MEM;
}
AES_cbc_encrypt(encrypted, (unsigned char*)buffer, fsize, &aes_key, iv, AES_DECRYPT);
free(encrypted);

clear_history_and_vars();
char* line = strtok(buffer, "\n");
Script* current_script = NULL;
while (line) {
if (line[0] == '#') {
line = strtok(NULL, "\n");
continue;
}
if (strncmp(line, "H: ", 3) == 0 && history_count < MAX_HISTORY) {
history[history_count++] = strdup(line + 3);
} else if (strncmp(line, "V: ", 3) == 0) {
char var_name[2] = {line[3], '\0'};
char* value = line + 5;
TritBigInt* bi;
if (parse_trit_string(value, &bi) == TRIT_OK) store_variable(var_name, bi);
} else if (strncmp(line, "S: ", 3) == 0 && script_count < 100) {
current_script = &scripts[script_count++];
strncpy(current_script->name, line + 3, MAX_SCRIPT_NAME - 1);
```

```
 current_script->name[MAX_SCRIPT_NAME - 1] = '\0';
 current_script->cmd_count = 0;
 } else if (strncmp(line, "C: ", 3) == 0 && current_script && current_script->cmd_count < MAX_SCRIPT_CMDS) {
 strncpy(current_script->commands[current_script->cmd_count++], line + 3, MAX_INPUT - 1);
 current_script->commands[current_script->cmd_count - 1][MAX_INPUT - 1] = '\0';
 } else if (strncmp(line, "T: ", 3) == 0) {
 sscanf(line + 3, "%d %d %d", &test_active, &test_score, &questions_answered);
 } else if (strncmp(line, "A: ", 3) == 0) {
 int qnum, ans;
 sscanf(line + 3, "%d %d", &qnum, &ans);
 if (qnum >= 0 && qnum < MAX_QUESTIONS) test_answers[qnum] = ans;
 }
 line = strtok(NULL, "\n");
 }
 free(buffer);
 return TRIT_OK;
}

Script* find_script(const char* name) {
 for (int i = 0; i < script_count; i++) {
 if (strcmp(scripts[i].name, name) == 0) return &scripts[i];
 }
 return NULL;
}

@*2 Error Handling.
@d TritError int
@d TRIT_OK 0
@d TRIT_ERR_MEM 1
@d TRIT_ERR_INPUT 2
@d TRIT_ERR_DIV_ZERO 3
@d TRIT_ERR_OVERFLOW 4
@d TRIT_ERR_UNDEFINED 5
@d TRIT_ERR_NEGATIVE 6
@d TRIT_ERR_PRECISION 7
@d TRIT_ERR_MMAP 8
@d TRIT_ERR_SCRIPT 9
@d TRIT_ERR_NETWORK 10
```

```
FILE* audit_log = NULL;

void init_audit_log() {
 audit_log = fopen("/var/log/tritjs_cisa.log", "a");
 if (!audit_log) {
 perror("Audit log initialization failed");
 audit_log = fopen("/tmp/tritjs_cisa.log", "a"); /* Fallback */
 }
 if (audit_log) chmod("/var/log/tritjs_cisa.log", 0600); /* Secure permissions */
}

void log_error(TritError err, const char* context) {
 pthread_mutex_lock(&mutex);
 if (audit_log) {
 time_t now;
 time(&now);
 fprintf(audit_log, "[%s] Error %d: %s in %s\n", ctime(&now), err, trit_error_str(err), context);
 fflush(audit_log);
 }
 pthread_mutex_unlock(&mutex);
}

const char* trit_error_str(TritError err) {
 switch (err) {
 case TRIT_OK: return "No error";
 case TRIT_ERR_MEM: return "Memory allocation failed";
 case TRIT_ERR_INPUT: return "Invalid input (trits 0-2 only)";
 case TRIT_ERR_DIV_ZERO: return "Division by zero";
 case TRIT_ERR_OVERFLOW: return "Overflow detected";
 case TRIT_ERR_UNDEFINED: return "Operation undefined";
 case TRIT_ERR_NEGATIVE: return "Negative input";
 case TRIT_ERR_PRECISION: return "Precision limit exceeded";
 case TRIT_ERR_MMAP: return "Memory mapping failed";
 case TRIT_ERR_SCRIPT: return "Scripting error";
 case TRIT_ERR_NETWORK: return "Network error";
 default: return "Unknown error";
 }
}
```

```
TritError map_trits(Trit** digits, int len, int* is_mapped, int* fd, char* tmp_path) {
 if (len * sizeof(Trit) > MAX_MMAP_SIZE) {
 log_error(TRIT_ERR_OVERFLOW, "map_trits");
 return TRIT_ERR_OVERFLOW;
 }
 strcpy(tmp_path, "/tmp/tritjs_cisa_XXXXXX");
 *fd = mkstemp(tmp_path);
 if (*fd < 0) {
 log_error(TRIT_ERR_MMAP, "map_trits: mkstemp");
 return TRIT_ERR_MMAP;
 }
 if (ftruncate(*fd, len * sizeof(Trit)) < 0) {
 close(*fd);
 unlink(tmp_path);
 log_error(TRIT_ERR_MMAP, "map_trits: ftruncate");
 return TRIT_ERR_MMAP;
 }
 *digits = mmap(NULL, len * sizeof(Trit), PROT_READ | PROT_WRITE, MAP_SHARED, *fd, 0);
 if (*digits == MAP_FAILED) {
 close(*fd);
 unlink(tmp_path);
 log_error(TRIT_ERR_MMAP, "map_trits: mmap");
 return TRIT_ERR_MMAP;
 }
 *is_mapped = 1;
 pthread_mutex_lock(&mutex);
 total_mapped_bytes += len * sizeof(Trit);
 operation_steps++;
 display_memory_and_stats("Mapping", "merge");
 pthread_mutex_unlock(&mutex);
 unlink(tmp_path);
 return TRIT_OK;
}

void unmap_trits(Trit* digits, int len, int is_mapped, int fd) {
 if (is_mapped && digits != MAP_FAILED) {
 pthread_mutex_lock(&mutex);
 total_mapped_bytes -= len * sizeof(Trit);
 operation_steps++;
```

```
 display_memory_and_stats("Unmapping", "merge");
 pthread_mutex_unlock(&mutex);
 munmap(digits, len * sizeof(Trit));
 if (fd >= 0) close(fd);
 } else if (!is_mapped) {
 free(digits);
 }
}

TritError tritbig_from_trits(Trit* trits, int len, int sign, TritBigInt** bi) {
 if (!trits || len <= 0) {
 log_error(TRIT_ERR_INPUT, "tritbig_from_trits");
 return TRIT_ERR_INPUT;
 }
 *bi = calloc(1, sizeof(TritBigInt));
 if (!*bi) {
 log_error(TRIT_ERR_MEM, "tritbig_from_trits");
 return TRIT_ERR_MEM;
 }
 TritError err = map_trits(&(*bi)->digits, len, &(*bi)->is_mapped, &(*bi)->fd, (*bi)->tmp_path);
 if (err != TRIT_OK) {
 free(*bi);
 return err;
 }
 memcpy((*bi)->digits, trits, len * sizeof(Trit));
 (*bi)->len = len;
 (*bi)->sign = sign;
 return TRIT_OK;
}

void tritbig_free(TritBigInt* bi) {
 if (bi) {
 unmap_trits(bi->digits, bi->len, bi->is_mapped, bi->fd);
 free(bi);
 }
}

TritError tritfloat_from_bigint(TritBigInt* bi, TritFloat* tf) {
 if (!bi || bi->len <= 0) {
```

```
 log_error(TRIT_ERR_INPUT, "tritfloat_from_bigint");
 return TRIT_ERR_INPUT;
 }
 tf->sign = bi->sign;
 tf->i_len = bi->len;
 tf->f_len = 0;
 TritError err = map_trits(&tf->integer, bi->len, &tf->i_mapped, &tf->i_fd, tf->i_tmp_path);
 if (err != TRIT_OK) return err;
 memcpy(tf->integer, bi->digits, bi->len * sizeof(Trit));
 tf->fraction = NULL;
 tf->f_mapped = 0;
 return TRIT_OK;
}

void tritfloat_free(TritFloat tf) {
 unmap_trits(tf.integer, tf.i_len, tf.i_mapped, tf.i_fd);
 if (tf.f_len > 0) unmap_trits(tf.fraction, tf.f_len, tf.f_mapped, tf.f_fd);
}

TritError tritcomplex_from_float(TritFloat real, TritFloat imag, TritComplex* tc) {
 tc->real = real;
 tc->imag = imag;
 return TRIT_OK;
}

void tritcomplex_free(TritComplex tc) {
 tritfloat_free(tc.real);
 tritfloat_free(tc.imag);
}

@*2 Arithmetic Operations.
@c
TritError tritjs_add_big(TritBigInt* a, TritBigInt* b, TritBigInt** result) {
 if (!a || !b || a->len > 10000 || b->len > 10000) {
 log_error(TRIT_ERR_INPUT, "tritjs_add_big");
 return TRIT_ERR_INPUT;
 }
 int max_len = (a->len > b->len) ? a->len : b->len;
 Trit* temp = calloc(max_len + 1, sizeof(Trit));
```

```
 if (!temp) {
 log_error(TRIT_ERR_MEM, "tritjs_add_big");
 return TRIT_ERR_MEM;
 }
 int carry = 0;

 if (a->sign == b->sign) {
 for (int i = max_len - 1, pos = 0; i >= 0; i--, pos++) {
 Trit a_trit = (i < a->len) ? a->digits[i] : 0;
 Trit b_trit = (i < b->len) ? b->digits[i] : 0;
 int sum = a_trit + b_trit + carry;
 temp[max_len - pos] = sum % TRIT_MAX;
 carry = sum / TRIT_MAX;
 operation_steps++;
 display_memory_and_stats("add", "merge");
 }
 if (carry) temp[0] = carry;
 int result_len = carry ? max_len + 1 : max_len;
 if (!carry) memmove(temp, temp + 1, max_len * sizeof(Trit));
 TritError err = tritbig_from_trits(temp, result_len, a->sign, result);
 free(temp);
 return err;
 } else {
 TritBigInt* b_neg;
 TritError err = tritbig_from_trits(b->digits, b->len, !b->sign, &b_neg);
 if (err != TRIT_OK) {
 free(temp);
 return err;
 }
 err = tritjs_add_big(a, b_neg, result);
 tritbig_free(b_neg);
 free(temp);
 return err;
 }
}

TritError tritjs_subtract_big(TritBigInt* a, TritBigInt* b, TritBigInt** result) {
 if (!a || !b) {
 log_error(TRIT_ERR_INPUT, "tritjs_subtract_big");
```

```
 return TRIT_ERR_INPUT;
 }
 TritBigInt* b_neg;
 TritError err = tritbig_from_trits(b->digits, b->len, !b->sign, &b_neg);
 if (err != TRIT_OK) return err;
 err = tritjs_add_big(a, b_neg, result);
 tritbig_free(b_neg);
 return err;
}

TritError tritjs_multiply_big(TritBigInt* a, TritBigInt* b, TritBigInt** result) {
 if (!a || !b) {
 log_error(TRIT_ERR_INPUT, "tritjs_multiply_big");
 return TRIT_ERR_INPUT;
 }
 int max_len = a->len + b->len;
 Trit* temp = calloc(max_len, sizeof(Trit));
 if (!temp) {
 log_error(TRIT_ERR_MEM, "tritjs_multiply_big");
 return TRIT_ERR_MEM;
 }
 for (int i = a->len - 1; i >= 0; i--) {
 int carry = 0;
 for (int j = b->len - 1; j >= 0; j--) {
 int pos = i + j + 1;
 int prod = a->digits[i] * b->digits[j] + temp[pos] + carry;
 temp[pos] = prod % TRIT_MAX;
 carry = prod / TRIT_MAX;
 operation_steps++;
 display_memory_and_stats("mul", "merge");
 }
 if (carry) temp[i] += carry;
 }
 int start = 0;
 while (start < max_len - 1 && temp[start] == 0) start++;
 int sign = (a->sign == b->sign) ? 0 : 1;
 TritError err = tritbig_from_trits(temp + start, max_len - start, sign, result);
 free(temp);
 return err;
```

```
}

void cleanup_div(TritBigInt* temp_rem, TritFloat* dividend, TritFloat* divisor, TritDivResult* result) {
 if (temp_rem) tritbig_free(temp_rem);
 if (dividend->integer) tritfloat_free(*dividend);
 if (divisor->integer) tritfloat_free(*divisor);
 if (result->quotient.integer) tritfloat_free(result->quotient);
 if (result->remainder.integer) tritfloat_free(result->remainder);
}

TritError tritjs_divide_big(TritBigInt* a, TritBigInt* b, TritDivResult* result, int precision) {
 TritError err;
 if (!a || !b) {
 log_error(TRIT_ERR_INPUT, "tritjs_divide_big");
 return TRIT_ERR_INPUT;
 }
 if (precision <= 0 || precision > 10) {
 log_error(TRIT_ERR_PRECISION, "tritjs_divide_big");
 return TRIT_ERR_PRECISION;
 }

 int b_is_zero = 1;
 for (int i = 0; i < b->len; i++) {
 if (b->digits[i] != 0) { b_is_zero = 0; break; }
 }
 if (b_is_zero) {
 log_error(TRIT_ERR_DIV_ZERO, "tritjs_divide_big");
 return TRIT_ERR_DIV_ZERO;
 }

 TritFloat dividend = {0}, divisor = {0};
 if ((err = tritfloat_from_bigint(a, &dividend)) != TRIT_OK) return err;
 if ((err = tritfloat_from_bigint(b, &divisor)) != TRIT_OK) {
 tritfloat_free(dividend);
 return err;
 }

 result->quotient.i_len = a->len;
 result->quotient.f_len = precision;
```

```
 result->remainder.i_len = b->len;
 result->quotient.sign = (a->sign == b->sign) ? 0 : 1;
 result->remainder.sign = a->sign;

 if ((err = map_trits(&result->quotient.integer, a->len, &result->quotient.i_mapped, &result->quotient.i_fd, result-
>quotient.i_tmp_path)) != TRIT_OK) goto cleanup;
 if ((err = map_trits(&result->quotient.fraction, precision, &result->quotient.f_mapped, &result->quotient.f_fd, result-
>quotient.f_tmp_path)) != TRIT_OK) goto cleanup;
 if ((err = map_trits(&result->remainder.integer, b->len, &result->remainder.i_mapped, &result->remainder.i_fd, result-
>remainder.i_tmp_path)) != TRIT_OK) goto cleanup;

 TritBigInt* temp_rem = NULL;
 if ((err = tritbig_from_trits(a->digits, a->len, a->sign, &temp_rem)) != TRIT_OK) goto cleanup;

 for (int i = 0; i < a->len; i++) {
 int digit = 0;
 for (int q = 2; q >= 0; q--) {
 TritBigInt* multiple = NULL;
 Trit trits[] = {(Trit)q};
 if ((err = tritbig_from_trits(trits, 1, 0, &multiple)) != TRIT_OK) goto cleanup_inner;
 TritBigInt* prod = NULL;
 if ((err = tritjs_multiply_big(b, multiple, &prod)) != TRIT_OK) {
 tritbig_free(multiple);
 goto cleanup_inner;
 }
 TritBigInt* sub = NULL;
 if ((err = tritjs_subtract_big(temp_rem, prod, &sub)) == TRIT_OK) {
 digit = q;
 tritbig_free(temp_rem);
 temp_rem = sub;
 tritbig_free(multiple);
 tritbig_free(prod);
 break;
 }
 tritbig_free(multiple);
 tritbig_free(prod);
 operation_steps++;
 display_memory_and_stats("div", "merge");
 }
```

```
result->quotient.integer[i] = digit;
}

for (int i = 0; i < precision; i++) {
TritBigInt* three = NULL;
if ((err = tritbig_from_trits((Trit[]){1}, 1, 0, &three)) != TRIT_OK) goto cleanup_inner;
TritBigInt* temp_mul = NULL;
if ((err = tritjs_multiply_big(temp_rem, three, &temp_mul)) != TRIT_OK) {
tritbig_free(three);
goto cleanup_inner;
}
tritbig_free(temp_rem);
temp_rem = temp_mul;
int digit = 0;
for (int q = 2; q >= 0; q--) {
TritBigInt* multiple = NULL;
Trit trits[] = {(Trit)q};
if ((err = tritbig_from_trits(trits, 1, 0, &multiple)) != TRIT_OK) goto cleanup_inner;
TritBigInt* prod = NULL;
if ((err = tritjs_multiply_big(b, multiple, &prod)) != TRIT_OK) {
tritbig_free(multiple);
goto cleanup_inner;
}
TritBigInt* sub = NULL;
if ((err = tritjs_subtract_big(temp_rem, prod, &sub)) == TRIT_OK) {
digit = q;
tritbig_free(temp_rem);
temp_rem = sub;
tritbig_free(multiple);
tritbig_free(prod);
break;
}
tritbig_free(multiple);
tritbig_free(prod);
operation_steps++;
display_memory_and_stats("div", "merge");
}
result->quotient.fraction[i] = digit;
}
```

```
 int start = 0;
 while (start < result->quotient.i_len - 1 && result->quotient.integer[start] == 0) start++;
 if (start > 0) {
 memmove(result->quotient.integer, result->quotient.integer + start, (result->quotient.i_len - start) * sizeof(Trit));
 result->quotient.i_len -= start;
 }

cleanup_inner:
 tritbig_free(temp_rem);
cleanup:
 if (err != TRIT_OK) {
 tritfloat_free(result->quotient);
 tritfloat_free(result->remainder);
 }
 tritfloat_free(dividend);
 tritfloat_free(divisor);
 return err;
}

TritError tritjs_power_big(TritBigInt* base, TritBigInt* exp, TritBigInt** result) {
 if (!base || !exp) {
 log_error(TRIT_ERR_INPUT, "tritjs_power_big");
 return TRIT_ERR_INPUT;
 }
 if (exp->sign) {
 log_error(TRIT_ERR_NEGATIVE, "tritjs_power_big");
 return TRIT_ERR_NEGATIVE;
 }
 TritError err;
 Trit trits[] = {1};
 if ((err = tritbig_from_trits(trits, 1, 0, result)) != TRIT_OK) return err;
 unsigned long exp_val = 0;
 for (int i = 0; i < exp->len; i++) exp_val = exp_val * TRIT_MAX + exp->digits[i];
 if (exp_val > 1000) {
 tritbig_free(*result);
 log_error(TRIT_ERR_OVERFLOW, "tritjs_power_big");
 return TRIT_ERR_OVERFLOW;
 }
```

```
 int sign = (base->sign && (exp_val % 2)) ? 1 : 0;
 for (unsigned long i = 0; i < exp_val; i++) {
 TritBigInt* temp;
 if ((err = tritjs_multiply_big(*result, base, &temp)) != TRIT_OK) {
 tritbig_free(*result);
 return err;
 }
 tritbig_free(*result);
 *result = temp;
 operation_steps++;
 display_memory_and_stats("pow", "merge");
 }
 (*result)->sign = sign;
 return TRIT_OK;
}

TritError tritjs_factorial_big(TritBigInt* a, TritBigInt** result) {
 if (!a) {
 log_error(TRIT_ERR_INPUT, "tritjs_factorial_big");
 return TRIT_ERR_INPUT;
 }
 if (a->sign) {
 log_error(TRIT_ERR_NEGATIVE, "tritjs_factorial_big");
 return TRIT_ERR_NEGATIVE;
 }
 unsigned long a_val = 0;
 for (int i = 0; i < a->len; i++) a_val = a_val * TRIT_MAX + a->digits[i];
 if (a_val > 20) {
 log_error(TRIT_ERR_OVERFLOW, "tritjs_factorial_big");
 return TRIT_ERR_OVERFLOW;
 }
 TritError err;
 Trit trits[] = {1};
 if ((err = tritbig_from_trits(trits, 1, 0, result)) != TRIT_OK) return err;
 for (unsigned long i = 1; i <= a_val; i++) {
 TritBigInt* i_bi;
 Trit i_trits[2];
 i_trits[0] = i / TRIT_MAX; i_trits[1] = i % TRIT_MAX;
 int len = (i >= TRIT_MAX) ? 2 : 1;
```

```
 if ((err = tritbig_from_trits(i_trits + (2 - len), len, 0, &i_bi)) != TRIT_OK) {
 tritbig_free(*result);
 return err;
 }
 TritBigInt* temp;
 if ((err = tritjs_multiply_big(*result, i_bi, &temp)) != TRIT_OK) {
 tritbig_free(i_bi);
 tritbig_free(*result);
 return err;
 }
 tritbig_free(*result);
 tritbig_free(i_bi);
 *result = temp;
 operation_steps++;
 display_memory_and_stats("fact", "merge");
 }
 return TRIT_OK;
}

@*2 Scientific Operations.
@c
TritError tritjs_sqrt_complex(TritBigInt* a, int precision, TritComplex* result) {
 if (!a || precision <= 0 || precision > 10 || a->len > 10000) {
 log_error(TRIT_ERR_PRECISION, "tritjs_sqrt_complex");
 return TRIT_ERR_PRECISION;
 }
 unsigned long a_val = 0;
 for (int i = 0; i < a->len; i++) {
 if (a_val > ULONG_MAX / TRIT_MAX) {
 log_error(TRIT_ERR_OVERFLOW, "tritjs_sqrt_complex");
 return TRIT_ERR_OVERFLOW;
 }
 a_val = a_val * TRIT_MAX + a->digits[i];
 }
 double val = (double)a_val * (a->sign ? -1 : 1);
 TritError err;
 if (val >= 0) {
 double sqrt_val = sqrt(val);
 unsigned long int_part = (unsigned long)sqrt_val;
```

```
 double frac_part = sqrt_val - int_part;
 Trit* int_trits = calloc((a->len + 1) / 2, sizeof(Trit));
 Trit* frac_trits = calloc(precision, sizeof(Trit));
 if (!int_trits || !frac_trits) {
 free(int_trits); free(frac_trits);
 log_error(TRIT_ERR_MEM, "tritjs_sqrt_complex");
 return TRIT_ERR_MEM;
 }
 for (int i = (a->len + 1) / 2 - 1; i >= 0; i--) {
 int_trits[i] = int_part % TRIT_MAX;
 int_part /= TRIT_MAX;
 operation_steps++;
 display_memory_and_stats("sqrt", "merge");
 }
 for (int i = precision - 1; i >= 0; i--) {
 frac_part *= TRIT_MAX;
 frac_trits[i] = (unsigned long)frac_part;
 frac_part -= (unsigned long)frac_part;
 operation_steps++;
 display_memory_and_stats("sqrt", "merge");
 }
 TritBigInt* real_int;
 if ((err = tritbig_from_trits(int_trits, (a->len + 1) / 2, 0, &real_int)) != TRIT_OK) goto sqrt_cleanup;
 if ((err = tritfloat_from_bigint(real_int, &result->real)) != TRIT_OK) goto sqrt_cleanup;
 tritbig_free(real_int);
 if ((err = map_trits(&result->real.fraction, precision, &result->real.f_mapped, &result->real.f_fd, result->real.f_tmp_path)) != TRIT_OK)
goto sqrt_cleanup;
 memcpy(result->real.fraction, frac_trits, precision * sizeof(Trit));
 result->real.f_len = precision;
 result->imag.integer = calloc(1, sizeof(Trit));
 if (!result->imag.integer) { err = TRIT_ERR_MEM; goto sqrt_cleanup; }
 result->imag.i_len = 1;
 result->imag.i_mapped = 0;
 result->imag.sign = 0;
 result->imag.fraction = NULL;
 result->imag.f_len = 0;
 sqrt_cleanup:
 free(int_trits); free(frac_trits);
 if (err != TRIT_OK) tritcomplex_free(*result);
```

```
 return err;
 } else {
 double sqrt_val = sqrt(-val);
 unsigned long int_part = (unsigned long)sqrt_val;
 double frac_part = sqrt_val - int_part;
 result->real.integer = calloc(1, sizeof(Trit));
 if (!result->real.integer) {
 log_error(TRIT_ERR_MEM, "tritjs_sqrt_complex");
 return TRIT_ERR_MEM;
 }
 result->real.i_len = 1;
 result->real.i_mapped = 0;
 result->real.sign = 0;
 if ((err = map_trits(&result->real.fraction, precision, &result->real.f_mapped, &result->real.f_fd, result->real.f_tmp_path)) != TRIT_OK)
{
 free(result->real.integer);
 return err;
 }
 memset(result->real.fraction, 0, precision * sizeof(Trit));
 result->real.f_len = precision;
 Trit* imag_int = calloc((a->len + 1) / 2, sizeof(Trit));
 Trit* imag_frac = calloc(precision, sizeof(Trit));
 if (!imag_int || !imag_frac) {
 free(imag_int); free(imag_frac);
 tritfloat_free(result->real);
 log_error(TRIT_ERR_MEM, "tritjs_sqrt_complex");
 return TRIT_ERR_MEM;
 }
 for (int i = (a->len + 1) / 2 - 1; i >= 0; i--) {
 imag_int[i] = int_part % TRIT_MAX;
 int_part /= TRIT_MAX;
 operation_steps++;
 display_memory_and_stats("sqrt", "merge");
 }
 for (int i = precision - 1; i >= 0; i--) {
 frac_part *= TRIT_MAX;
 imag_frac[i] = (unsigned long)frac_part;
 frac_part -= (unsigned long)frac_part;
 operation_steps++;
```

```
 display_memory_and_stats("sqrt", "merge");
 }
 TritBigInt* imag_bi;
 if ((err = tritbig_from_trits(imag_int, (a->len + 1) / 2, 0, &imag_bi)) != TRIT_OK) {
 free(imag_int); free(imag_frac);
 tritfloat_free(result->real);
 return err;
 }
 if ((err = tritfloat_from_bigint(imag_bi, &result->imag)) != TRIT_OK) {
 tritbig_free(imag_bi);
 free(imag_int); free(imag_frac);
 return err;
 }
 tritbig_free(imag_bi);
 if ((err = map_trits(&result->imag.fraction, precision, &result->imag.f_mapped, &result->imag.f_fd, result->imag.f_tmp_path)) !=
TRIT_OK) {
 free(imag_int); free(imag_frac);
 tritfloat_free(result->real);
 return err;
 }
 memcpy(result->imag.fraction, imag_frac, precision * sizeof(Trit));
 result->imag.f_len = precision;
 free(imag_int); free(imag_frac);
 return TRIT_OK;
 }
}

TritError tritjs_log3_complex(TritBigInt* a, int precision, TritComplex* result) {
 if (!a || precision <= 0 || precision > 10) {
 log_error(TRIT_ERR_PRECISION, "tritjs_log3_complex");
 return TRIT_ERR_PRECISION;
 }
 unsigned long a_val = 0;
 for (int i = 0; i < a->len; i++) a_val = a_val * TRIT_MAX + a->digits[i];
 double real = (double)a_val * (a->sign ? -1 : 1);
 double imag = 0;
 double mag = sqrt(real * real + imag * imag);
 double arg = atan2(imag, real);
 double ln3 = log(3.0);
```

```
 double real_val = log(mag) / ln3;
 double imag_val = arg / ln3;

 TritError err;
 Trit* real_int = calloc(a->len, sizeof(Trit));
 Trit* real_frac = calloc(precision, sizeof(Trit));
 Trit* imag_int = calloc(a->len, sizeof(Trit));
 Trit* imag_frac = calloc(precision, sizeof(Trit));
 if (!real_int || !real_frac || !imag_int || !imag_frac) {
 free(real_int); free(real_frac); free(imag_int); free(imag_frac);
 log_error(TRIT_ERR_MEM, "tritjs_log3_complex");
 return TRIT_ERR_MEM;
 }

 unsigned long r_int_part = (unsigned long)fabs(real_val);
 double r_frac_part = fabs(real_val) - r_int_part;
 for (int i = a->len - 1; i >= 0; i--) {
 real_int[i] = r_int_part % TRIT_MAX;
 r_int_part /= TRIT_MAX;
 operation_steps++;
 display_memory_and_stats("log3", "merge");
 }
 for (int i = precision - 1; i >= 0; i--) {
 r_frac_part *= TRIT_MAX;
 real_frac[i] = (unsigned long)r_frac_part;
 r_frac_part -= (unsigned long)r_frac_part;
 operation_steps++;
 display_memory_and_stats("log3", "merge");
 }
 TritBigInt* real_bi;
 if ((err = tritbig_from_trits(real_int, a->len, real_val < 0 ? 1 : 0, &real_bi)) != TRIT_OK) goto log_cleanup;
 if ((err = tritfloat_from_bigint(real_bi, &result->real)) != TRIT_OK) goto log_cleanup;
 tritbig_free(real_bi);
 if ((err = map_trits(&result->real.fraction, precision, &result->real.f_mapped, &result->real.f_fd, result->real.f_tmp_path)) != TRIT_OK)
goto log_cleanup;
 memcpy(result->real.fraction, real_frac, precision * sizeof(Trit));
 result->real.f_len = precision;

 unsigned long i_int_part = (unsigned long)fabs(imag_val);
```

```
 double i_frac_part = fabs(imag_val) - i_int_part;
 for (int i = a->len - 1; i >= 0; i--) {
 imag_int[i] = i_int_part % TRIT_MAX;
 i_int_part /= TRIT_MAX;
 operation_steps++;
 display_memory_and_stats("log3", "merge");
 }
 for (int i = precision -  1; i >= 0; i--) {
 i_frac_part *= TRIT_MAX;
 imag_frac[i] = (unsigned long)i_frac_part;
 i_frac_part -= (unsigned long)i_frac_part;
 operation_steps++;
 display_memory_and_stats("log3", "merge");
 }
 TritBigInt* imag_bi;
 if ((err = tritbig_from_trits(imag_int, a->len, imag_val < 0 ? 1 : 0, &imag_bi)) != TRIT_OK) goto log_cleanup;
 if ((err = tritfloat_from_bigint(imag_bi, &result->imag)) != TRIT_OK) goto log_cleanup;
 tritbig_free(imag_bi);
 if ((err = map_trits(&result->imag.fraction, precision, &result->imag.f_mapped, &result->imag.f_fd, result->imag.f_tmp_path)) !=
TRIT_OK) goto log_cleanup;
 memcpy(result->imag.fraction, imag_frac, precision * sizeof(Trit));
 result->imag.f_len = precision;

log_cleanup:
 free(real_int); free(real_frac); free(imag_int); free(imag_frac);
 if (err != TRIT_OK) tritcomplex_free(*result);
 return err;
}

TritError tritjs_trig_complex(TritBigInt* a, int precision, TritComplex* result, double (*trig_func)(double)) {
 if (!a || precision <= 0 || precision > 10) {
 log_error(TRIT_ERR_PRECISION, "tritjs_trig_complex");
 return TRIT_ERR_PRECISION;
 }
 unsigned long a_val = 0;
 for (int i = 0; i < a->len; i++) a_val = a_val * TRIT_MAX + a->digits[i];
 double pi_approx = 3.1415926535;
 double angle = (double)a_val * pi_approx / 10.0 * (a->sign ? -1 : 1);
 double trig_val = trig_func(angle);
```

```
 int sign = trig_val < 0 ? 1 : 0;
 double abs_val = fabs(trig_val);
 unsigned long int_part = (unsigned long)abs_val;
 double frac_part = abs_val - int_part;

 TritError err;
 Trit* int_trits = calloc(1, sizeof(Trit));
 Trit* frac_trits = calloc(precision, sizeof(Trit));
 if (!int_trits || !frac_trits) {
 free(int_trits); free(frac_trits);
 log_error(TRIT_ERR_MEM, "tritjs_trig_complex");
 return TRIT_ERR_MEM;
 }
 int_trits[0] = int_part % TRIT_MAX;
 for (int i = precision - 1; i >= 0; i--) {
 frac_part *= TRIT_MAX;
 frac_trits[i] = (unsigned long)frac_part;
 frac_part -= (unsigned long)frac_part;
 operation_steps++;
 display_memory_and_stats("trig", "merge");
 }
 TritBigInt* real_bi;
 if ((err = tritbig_from_trits(int_trits, 1, sign, &real_bi)) != TRIT_OK) goto trig_cleanup;
 if ((err = tritfloat_from_bigint(real_bi, &result->real)) != TRIT_OK) goto trig_cleanup;
 tritbig_free(real_bi);
 if ((err = map_trits(&result->real.fraction, precision, &result->real.f_mapped, &result->real.f_fd, result->real.f_tmp_path)) != TRIT_OK)
goto trig_cleanup;
 memcpy(result->real.fraction, frac_trits, precision * sizeof(Trit));
 result->real.f_len = precision;
 result->imag.integer = calloc(1, sizeof(Trit));
 if (!result->imag.integer) { err = TRIT_ERR_MEM; goto trig_cleanup; }
 result->imag.i_len = 1;
 result->imag.i_mapped = 0;
 result->imag.sign = 0;
 result->imag.fraction = NULL;
 result->imag.f_len = 0;

trig_cleanup:
 free(int_trits); free(frac_trits);
```

```
 if (err != TRIT_OK) tritcomplex_free(*result);
 return err;
}

TritError tritjs_sin_complex(TritBigInt* a, int precision, TritComplex* result) {
 return tritjs_trig_complex(a, precision, result, sin);
}

TritError tritjs_cos_complex(TritBigInt* a, int precision, TritComplex* result) {
 return tritjs_trig_complex(a, precision, result, cos);
}

TritError tritjs_tan_complex(TritBigInt* a, int precision, TritComplex* result) {
 if (!a || precision <= 0 || precision > 10) {
 log_error(TRIT_ERR_PRECISION, "tritjs_tan_complex");
 return TRIT_ERR_PRECISION;
 }
 unsigned long a_val = 0;
 for (int i = 0; i < a->len; i++) a_val = a_val * TRIT_MAX + a->digits[i];
 double pi_approx = 3.1415926535;
 double angle = (double)a_val * pi_approx / 10.0 * (a->sign ? -1 : 1);
 double tan_val = tan(angle);
 if (fabs(tan_val) > 1000.0) {
 log_error(TRIT_ERR_UNDEFINED, "tritjs_tan_complex");
 return TRIT_ERR_UNDEFINED;
 }
 return tritjs_trig_complex(a, precision, result, tan);
}

TritError tritjs_pi(int* len, Trit** pi) {
 Trit pi_val[] = {1, 0, 0, 1, 0, 2, 2, 1};
 *len = 8;
 *pi = malloc(*len * sizeof(Trit));
 if (!*pi) {
 log_error(TRIT_ERR_MEM, "tritjs_pi");
 return TRIT_ERR_MEM;
 }
 memcpy(*pi, pi_val, *len * sizeof(Trit));
 operation_steps++;
```

```
 display_memory_and_stats("pi", "merge");
 return TRIT_OK;
}

@*2 Utility Functions.
@c
TritError tritjs_to_string(TritBigInt* bi, char** str) {
 if (!bi || bi->len <= 0) {
 log_error(TRIT_ERR_INPUT, "tritjs_to_string");
 return TRIT_ERR_INPUT;
 }
 *str = malloc(bi->len + 1 + (bi->sign ? 1 : 0));
 if (!*str) {
 log_error(TRIT_ERR_MEM, "tritjs_to_string");
 return TRIT_ERR_MEM;
 }
 char* p = *str;
 if (bi->sign) *p++ = '-';
 for (int i = 0; i < bi->len; i++) *p++ = '0' + bi->digits[i];
 *p = '\0';
 return TRIT_OK;
}

TritError tritfloat_to_string(TritFloat tf, char** str) {
 if (!tf.integer || tf.i_len <= 0) {
 log_error(TRIT_ERR_INPUT, "tritfloat_to_string");
 return TRIT_ERR_INPUT;
 }
 int total_len = tf.i_len + (tf.f_len > 0 ? tf.f_len + 1 : 0) + (tf.sign ? 1 : 0);
 *str = malloc(total_len + 1);
 if (!*str) {
 log_error(TRIT_ERR_MEM, "tritfloat_to_string");
 return TRIT_ERR_MEM;
 }
 char* p = *str;
 if (tf.sign) *p++ = '-';
 for (int i = 0; i < tf.i_len; i++) *p++ = '0' + tf.integer[i];
 if (tf.f_len > 0) {
 *p++ = '.';
```

```
 for (int i = 0; i < tf.f_len; i++) *p++ = '0' + tf.fraction[i];
 }
 *p = '\0';
 return TRIT_OK;
}

TritError tritcomplex_to_string(TritComplex tc, char** str) {
 char* real_str, *imag_str;
 TritError err;
 if ((err = tritfloat_to_string(tc.real, &real_str)) != TRIT_OK) return err;
 if ((err = tritfloat_to_string(tc.imag, &imag_str)) != TRIT_OK) {
 free(real_str);
 return err;
 }
 int imag_zero = (tc.imag.i_len == 1 && tc.imag.integer[0] == 0 && tc.imag.f_len == 0);
 if (imag_zero) {
 *str = real_str;
 free(imag_str);
 return TRIT_OK;
 }
 *str = malloc(strlen(real_str) + strlen(imag_str) + 4);
 if (!*str) {
 free(real_str);
 free(imag_str);
 log_error(TRIT_ERR_MEM, "tritcomplex_to_string");
 return TRIT_ERR_MEM;
 }
 sprintf(*str, "%s %si", real_str, imag_str);
 free(real_str);
 free(imag_str);
 return TRIT_OK;
}

TritError net_listen(const char* ip, int port) {
 int sock = socket(AF_INET, SOCK_STREAM, 0);
 if (sock < 0) {
 log_error(TRIT_ERR_NETWORK, "net_listen: socket creation");
 return TRIT_ERR_NETWORK;
 }
```

```
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_port = htons(port);
if (inet_pton(AF_INET, ip, &addr.sin_addr) <= 0) {
close(sock);
log_error(TRIT_ERR_NETWORK, "net_listen: invalid IP");
return TRIT_ERR_NETWORK;
}

if (bind(sock, (struct sockaddr*)&addr, sizeof(addr)) < 0) {
close(sock);
log_error(TRIT_ERR_NETWORK, "net_listen: bind");
return TRIT_ERR_NETWORK;
}
if (listen(sock, 5) < 0) {
close(sock);
log_error(TRIT_ERR_NETWORK, "net_listen: listen");
return TRIT_ERR_NETWORK;
}

printf("Listening on %s:%d\n", ip, port);
char buffer[MAX_INPUT];
while (1) {
int client = accept(sock, NULL, NULL);
if (client < 0) {
log_error(TRIT_ERR_NETWORK, "net_listen: accept");
continue;
}
int len = recv(client, buffer, sizeof(buffer) - 1, 0);
if (len > 0) {
buffer[len] = '\0';
add_to_history(buffer);
printf("Received: %s\n", buffer);
}
close(client);
}
close(sock);
return TRIT_OK;
```

```
}

TritError export_json(const char* filename) {
 FILE* f = fopen(filename, "w");
 if (!f) {
 log_error(TRIT_ERR_INPUT, "export_json: file open");
 return TRIT_ERR_INPUT;
 }

 fprintf(f, "{\n \"history\": [");
 for (int i = 0; i < history_count; i++) {
 fprintf(f, "%s\"%s\"", i == 0 ? "" : ", ", history[i]);
 }
 fprintf(f, "],\n \"variables\": {\n");
 int first_var = 1;
 for (int i = 0; i < 26; i++) {
 if (variables[i]) {
 char* str;
 if (tritjs_to_string(variables[i], &str) == TRIT_OK) {
 fprintf(f, "%s \"%c\": \"%s\"", first_var ? "" : ",\n", 'A' + i, str);
 free(str);
 first_var = 0;
 }
 }
 }
 fprintf(f, "\n },\n \"test\": {\n \"active\": %d,\n \"score\": %d,\n \"answered\": %d,\n \"answers\": {", test_active, test_score,
questions_answered);
 int first_ans = 1;
 for (int i = 0; i < MAX_QUESTIONS; i++) {
 if (test_answers[i]) {
 fprintf(f, "%s \"%d\": %d", first_ans ? "" : ",", i, test_answers[i]);
 first_ans = 0;
 }
 }
 fprintf(f, "\n }\n }\n}\n");
 fclose(f);
 return TRIT_OK;
}
```

```
@*2 CLI for CISA with AI-Driven Stats, Scripting, and Save/Load.
@c
TritError parse_trit_string(const char* str, TritBigInt** bi) {
 if (!str || !strlen(str) || strlen(str) > MAX_INPUT) {
 log_error(TRIT_ERR_INPUT, "parse_trit_string");
 return TRIT_ERR_INPUT;
 }
 int sign = (str[0] == '-') ? 1 : 0;
 const char* mag = sign ? str + 1 : str;
 int len = strlen(mag);
 Trit* trits = malloc(len * sizeof(Trit));
 if (!trits) {
 log_error(TRIT_ERR_MEM, "parse_trit_string");
 return TRIT_ERR_MEM;
 }
 for (int i = 0; i < len; i++) {
 if (mag[i] < '0' || mag[i] > '2') {
 free(trits);
 log_error(TRIT_ERR_INPUT, "parse_trit_string: invalid trit");
 return TRIT_ERR_INPUT;
 }
 trits[i] = mag[i] - '0';
 }
 TritError err = tritbig_from_trits(trits, len, sign, bi);
 free(trits);
 return err;
}

static const char* test_questions[MAX_QUESTIONS][6] = {
 {"What are the digits in ternary?", "0,1,2,3", "0,1,2", "0,1", "1,2,3", "2"},
 {"Decimal value of 120 ternary?", "3", "12", "15", "18", "3"},
 {"Application of TritJS-CISA?", "Weather", "Cybersecurity", "Games", "Finance", "2"},
 {"101 ternary to decimal?", "10", "11", "12", "13", "1"},
 {"1101 binary to ternary?", "21", "111", "102", "120", "3"},
 {"What is 1 + 2 in ternary?", "3", "10", "12", "11", "2"},
 {"What happens when you add 2 + 2 in ternary?", "Remains 2", "Becomes 4", "Carries over to 11", "Becomes 10", "3"},
 {"Perform 12 + 21 in ternary:", "100", "110", "101", "33", "1"},
 {"What is the sum of 102 + 11 in ternary?", "120", "110", "111", "200", "1"},
 {"In TritJS-CISA, what does operation_steps++ track?", "Memory", "Additions", "Carries", "Steps", "4"},
```

```
 {"What is 2 - 1 in ternary?", "0", "1", "2", "10", "2"},
 {"What happens when subtracting 1 - 2?", "Borrow", "Negative", "Always 1", "No borrow", "1"},
 {"Compute 21 - 12 in ternary:", "2", "10", "12", "1", "2"},
 {"What is 100 - 11 in ternary?", "22", "12", "21", "11", "3"},
 {"How does tritjs_subtract_big handle borrowing?", "Adds negative", "Direct subtract", "Decimal", "Ignores", "1"},
 {"What is 2 × 2 in ternary?", "4", "11", "10", "20", "2"},
 {"What is 12 × 2 in ternary?", "22", "111", "101", "24", "3"},
 {"Compute 11 × 10 in ternary:", "110", "100", "120", "210", "1"},
 {"What does tritjs_multiply_big do when product exceeds 2?", "Discards", "Carries", "Decimal", "Error", "2"},
 {"Multiply 102 × 2 in ternary:", "211", "201", "210", "220", "1"},
 {"What is 10 ÷ 2 in ternary?", "1", "2", "11", "10", "2"},
 {"How are remainders handled in division?", "Ignored", "Binary", "Fractions", "None", "3"},
 {"Divide 21 ÷ 2 in ternary:", "10.1", "11", "10", "12", "1"},
 {"Quotient of 102 ÷ 11 in ternary?", "2", "10", "12", "20", "2"},
 {"Perform 210 ÷ 12 in ternary:", "12", "11", "20", "10", "1"},
 {"What is 2² in ternary?", "10", "11", "12", "20", "2"},
 {"What is log₃(9) in ternary?", "10", "2", "11", "12", "1"},
 {"Compute 3! in ternary:", "20", "12", "11", "10", "1"},
 {"What is √12 in ternary (approx 1 trit)?", "10", "11", "20", "12", "2"},
 {"What limits factorial in TritJS-CISA?", "Memory", "Input > 20", "Precision", "Negatives", "2"},
 {"Solve X + 12 = 21 in ternary:", "1", "10", "11", "20", "2"},
 {"Solve Y × 2 = 11 in ternary:", "2", "10", "1", "12", "2"},
 {"Solve X - 10 = 2 in ternary:", "11", "12", "20", "10", "2"},
 {"Solve X + Y = 12 and X × Y = 21:", "X=10,Y=2", "X=11,Y=1", "X=12,Y=0", "No solution", "2"},
 {"How does TritJS-CISA handle variables?", "Decimal", "store_variable", "Binary", "Ignores", "2"},
 {"Advantage of ternary computing?", "Less energy", "Binary gates", "Quantum", "No use", "1"},
 {"Field benefiting from TritJS-CISA?", "Cybersecurity", "AI", "Compression", "All", "4"},
 {"How does ternary improve cybersecurity?", "Faster", "Compact", "Obfuscation", "Simpler", "3"},
 {"Ternary sum 210 + 122?", "1002", "1102", "1021", "1111", "2"},
 {"What does save_state enable?", "Real-time", "Persistence", "Binary", "Sorting", "2"}
};

void display_test_question(int qnum) {
 if (qnum < 0 || qnum >= MAX_QUESTIONS) {
 printf("Error: Invalid question number\n");
 return;
 }
 printf("\nQuestion %d: %s\n", qnum + 1, test_questions[qnum][0]);
 printf("a) %s\nb) %s\nc) %s\nd) %s\nEnter answer (1-4): ",
```

```
 test_questions[qnum][1], test_questions[qnum][2],
 test_questions[qnum][3], test_questions[qnum][4]);
}

int check_test_answer(int qnum, int user_answer) {
 if (qnum < 0 || qnum >= MAX_QUESTIONS || user_answer < 1 || user_answer > 4) return 0;
 int correct_answer = atoi(test_questions[qnum][5]);
 return (user_answer == correct_answer) ? 1 : 0;
}

void learn_topic(const char* topic) {
    if (strcmp(topic, "ADD") == 0) {
        printf("Lesson: Ternary Addition\n");
        printf("In ternary (base-3), digits are 0, 1, 2. Example: 12 + 2\n");
        printf("Step 1: Align right:  12\n");
        printf("                    +  2\n");
        printf("Step 2: Add columns:\n");
        printf("  Units: 2 + 2 = 11 (3¹×1 + 3⁰×1 = 4 decimal, carry 1)\n");
        printf("  Threes: 1 + 0 + carry 1 = 2\n");
        printf("Result: 21 (3¹×2 + 3⁰×1 = 7 decimal)\n");
        printf("Try: 'add 12 2' in the CLI.\n");
    } else if (strcmp(topic, "SUB") == 0) {
        printf("Lesson: Ternary Subtraction\n");
        printf("Example: 21 - 12\n");
        printf("Step 1: Align right:  21\n");
        printf("                    - 12\n");
        printf("Step 2: Subtract with borrowing:\n");
        printf("  Units: 1 - 2 = borrow 1 from 2, becomes 11 - 2 = 2\n");
        printf("  Threes: 1 - 1 = 0\n");
        printf("Result: 2 (3⁰×2 = 2 decimal)\n");
        printf("Try: 'sub 21 12' in the CLI.\n");
    } else if (strcmp(topic, "MUL") == 0) {
        printf("Lesson: Ternary Multiplication\n");
        printf("Example: 11 × 10\n");
        printf("Step 1: Multiply each digit:\n");
        printf("  11 × 0 = 00\n");
        printf("  11 × 1 = 11 (shifted left)\n");
        printf("Step 2: Add:  00\n");
        printf("            + 11\n");
```

```
        printf("Result: 110 (3²×1 + 3¹×1 = 12 decimal)\n");
        printf("Try: 'mul 11 10' in the CLI.\n");
    } else {
        printf("Error: Unknown topic. Try 'LEARN ADD', 'LEARN SUB', or 'LEARN MUL'.\n");
    }
}

TritError execute_command(const char* input, int is_script) {
    char op[10], arg1[MAX_INPUT], arg2[MAX_INPUT] = "";
    int parsed = sscanf(input, "%9s %255s %255s", op, arg1, arg2);
    if (parsed < 1 || (parsed >= 2 && strlen(arg1) >= MAX_INPUT) || (arg2[0] && strlen(arg2) >= MAX_INPUT)) {
        if (!is_script) printf("Error: Invalid format or input too long\n");
        log_error(TRIT_ERR_INPUT, "execute_command: parsing");
        return TRIT_ERR_INPUT;
    }

    TritBigInt* a = NULL;
    TritBigInt* b = NULL;
    TritError err;

    /* Variable assignment */
    if (parsed == 2 && strchr(arg1, '=') && !arg2[0]) {
        char var_name[2] = {arg1[0], '\0'};
        char* value = strchr(arg1, '=') + 1;
        if ((err = parse_trit_string(value, &a)) != TRIT_OK) {
            if (!is_script) printf("Error: %s\n", trit_error_str(err));
            return err;
        }
        store_variable(var_name, a);
        if (!is_script) printf("%s stored\n", var_name);
        return TRIT_OK;
    }

    /* Variable recall or parsing arguments */
    if (parsed >= 2) {
        if (arg1[0] >= 'A' && arg1[0] <= 'Z' && arg1[1] == '\0') {
            a = recall_variable(arg1);
            if (!a) {
                if (!is_script) printf("Error: Variable %s not set\n", arg1);
```

```
            log_error(TRIT_ERR_INPUT, "execute_command: variable not set");
            return TRIT_ERR_INPUT;
        }
    } else if ((err = parse_trit_string(arg1, &a)) != TRIT_OK) {
        if (!is_script) printf("Error: %s\n", trit_error_str(err));
        return err;
    }
    if (strlen(arg2) > 0) {
        if (arg2[0] >= 'A' && arg2[0] <= 'Z' && arg2[1] == '\0') {
            b = recall_variable(arg2);
            if (!b) {
                if (!is_script) printf("Error: Variable %s not set\n", arg2);
                tritbig_free(a);
                log_error(TRIT_ERR_INPUT, "execute_command: variable not set");
                return TRIT_ERR_INPUT;
            }
        } else if ((err = parse_trit_string(arg2, &b)) != TRIT_OK) {
            if (!is_script) printf("Error: %s\n", trit_error_str(err));
            tritbig_free(a);
            return err;
        }
    }
}

/* Arithmetic Commands */
if ((strcmp(op, "add") == 0 || strcmp(op, "a") == 0) && b) {
    TritBigInt* result;
    if ((err = tritjs_add_big(a, b, &result)) != TRIT_OK) {
        if (!is_script) printf("Error: %s\n", trit_error_str(err));
    } else {
        char* str;
        if ((err = tritjs_to_string(result, &str)) == TRIT_OK) {
            if (!is_script) {
                printf("%s\n", str);
                add_to_history(str);
            }
            free(str);
        }
        tritbig_free(result);
```

```
    }
} else if ((strcmp(op, "sub") == 0 || strcmp(op, "s") == 0) && b) {
    TritBigInt* result;
    if ((err = tritjs_subtract_big(a, b, &result)) != TRIT_OK) {
        if (!is_script) printf("Error: %s\n", trit_error_str(err));
    } else {
        char* str;
        if ((err = tritjs_to_string(result, &str)) == TRIT_OK) {
            if (!is_script) {
                printf("%s\n", str);
                add_to_history(str);
            }
            free(str);
        }
        tritbig_free(result);
    }
} else if ((strcmp(op, "mul") == 0 || strcmp(op, "m") == 0) && b) {
    TritBigInt* result;
    if ((err = tritjs_multiply_big(a, b, &result)) != TRIT_OK) {
        if (!is_script) printf("Error: %s\n", trit_error_str(err));
    } else {
        char* str;
        if ((err = tritjs_to_string(result, &str)) == TRIT_OK) {
            if (!is_script) {
                printf("%s\n", str);
                add_to_history(str);
            }
            free(str);
        }
        tritbig_free(result);
    }
} else if ((strcmp(op, "div") == 0 || strcmp(op, "d") == 0) && b) {
    TritDivResult result = {{0}, {0}};
    if ((err = tritjs_divide_big(a, b, &result, 3)) != TRIT_OK) {
        if (!is_script) printf("Error: %s\n", trit_error_str(err));
    } else {
        char* q_str, *r_str;
        if ((err = tritfloat_to_string(result.quotient, &q_str)) == TRIT_OK &&
            (err = tritfloat_to_string(result.remainder, &r_str)) == TRIT_OK) {
```

```
            char full_result[512];
            snprintf(full_result, sizeof(full_result), "%s r %s", q_str, r_str);
            if (!is_script) {
                printf("%s\n", full_result);
                add_to_history(full_result);
            }
            free(q_str);
            free(r_str);
        }
        tritfloat_free(result.quotient);
        tritfloat_free(result.remainder);
    }
} else if ((strcmp(op, "pow") == 0 || strcmp(op, "p") == 0) && b) {
    TritBigInt* result;
    if ((err = tritjs_power_big(a, b, &result)) != TRIT_OK) {
        if (!is_script) printf("Error: %s\n", trit_error_str(err));
    } else {
        char* str;
        if ((err = tritjs_to_string(result, &str)) == TRIT_OK) {
            if (!is_script) {
                printf("%s\n", str);
                add_to_history(str);
            }
            free(str);
        }
        tritbig_free(result);
    }
} else if ((strcmp(op, "fact") == 0 || strcmp(op, "f") == 0) && parsed == 2) {
    TritBigInt* result;
    if ((err = tritjs_factorial_big(a, &result)) != TRIT_OK) {
        if (!is_script) printf("Error: %s\n", trit_error_str(err));
    } else {
        char* str;
        if ((err = tritjs_to_string(result, &str)) == TRIT_OK) {
            if (!is_script) {
                printf("%s\n", str);
                add_to_history(str);
            }
            free(str);
```

```
        }
        tritbig_free(result);
    }
}
/* Scientific Commands */
else if (strcmp(op, "sqrt") == 0 && parsed == 2) {
    TritComplex result;
    if ((err = tritjs_sqrt_complex(a, 3, &result)) != TRIT_OK) {
        if (!is_script) printf("Error: %s\n", trit_error_str(err));
    } else {
        char* str;
        if ((err = tritcomplex_to_string(result, &str)) == TRIT_OK) {
            if (!is_script) {
                printf("%s\n", str);
                add_to_history(str);
            }
            free(str);
        }
        tritcomplex_free(result);
    }
} else if (strcmp(op, "log3") == 0 && parsed == 2) {
    TritComplex result;
    if ((err = tritjs_log3_complex(a, 3, &result)) != TRIT_OK) {
        if (!is_script) printf("Error: %s\n", trit_error_str(err));
    } else {
        char* str;
        if ((err = tritcomplex_to_string(result, &str)) == TRIT_OK) {
            if (!is_script) {
                printf("%s\n", str);
                add_to_history(str);
            }
            free(str);
        }
        tritcomplex_free(result);
    }
} else if (strcmp(op, "sin") == 0 && parsed == 2) {
    TritComplex result;
    if ((err = tritjs_sin_complex(a, 3, &result)) != TRIT_OK) {
        if (!is_script) printf("Error: %s\n", trit_error_str(err));
```

```
    } else {
        char* str;
        if ((err = tritcomplex_to_string(result, &str)) == TRIT_OK) {
            if (!is_script) {
                printf("%s\n", str);
                add_to_history(str);
            }
            free(str);
        }
        tritcomplex_free(result);
    }
} else if (strcmp(op, "cos") == 0 && parsed == 2) {
    TritComplex result;
    if ((err = tritjs_cos_complex(a, 3, &result)) != TRIT_OK) {
        if (!is_script) printf("Error: %s\n", trit_error_str(err));
    } else {
        char* str;
        if ((err = tritcomplex_to_string(result, &str)) == TRIT_OK) {
            if (!is_script) {
                printf("%s\n", str);
                add_to_history(str);
            }
            free(str);
        }
        tritcomplex_free(result);
    }
} else if (strcmp(op, "tan") == 0 && parsed == 2) {
    TritComplex result;
    if ((err = tritjs_tan_complex(a, 3, &result)) != TRIT_OK) {
        if (!is_script) printf("Error: %s\n", trit_error_str(err));
    } else {
        char* str;
        if ((err = tritcomplex_to_string(result, &str)) == TRIT_OK) {
            if (!is_script) {
                printf("%s\n", str);
                add_to_history(str);
            }
            free(str);
        }
```

```
            tritcomplex_free(result);
        }
    } else if (strcmp(op, "pi") == 0 && parsed == 1) {
        int len;
        Trit* pi;
        if ((err = tritjs_pi(&len, &pi)) != TRIT_OK) {
            if (!is_script) printf("Error: %s\n", trit_error_str(err));
        } else {
            char* str = malloc(len + 1);
            if (str) {
                for (int i = 0; i < len; i++) str[i] = '0' + pi[i];
                str[len] = '\0';
                if (!is_script) {
                    printf("%s\n", str);
                    add_to_history(str);
                }
                free(str);
            }
            free(pi);
        }
    }
    /* Recall Command */
    else if ((strcmp(op, "recall") == 0 || strcmp(op, "rc") == 0) && parsed == 2) {
        int index = atoi(arg1);
        char* recalled = recall_history(index);
        if (recalled) {
            if (!is_script) printf("%s\n", recalled);
            free(recalled);
        } else {
            if (!is_script) printf("Error: Invalid history index\n");
            err = TRIT_ERR_INPUT;
        }
    }
    else if (strcmp(op, "LEARN") == 0 && parsed == 2) {
        learn_topic(arg1);
        err = TRIT_OK;
    }
    else if (strcmp(op, "TEST") == 0 && parsed == 2) {
        int week = atoi(arg1);
```

```
    if (week < 1 || week > 8) {
        if (!is_script) printf("Error: Invalid week (1-8)\n");
        log_error(TRIT_ERR_INPUT, "execute_command: TEST invalid week");
        err = TRIT_ERR_INPUT;
    } else {
        test_active = week;
        int start_q = (week - 1) * QUESTIONS_PER_WEEK;
        if (!is_script) {
            printf("Starting Week %d Test. Use 'CHECK <qnum> <answer>' to submit answers.\n", week);
            for (int i = 0; i < QUESTIONS_PER_WEEK; i++) {
                display_test_question(start_q + i);
                printf("(Previously answered: %d)\n", test_answers[start_q + i]);
            }
        }
        err = TRIT_OK;
    }
}
else if (strcmp(op, "CHECK") == 0 && parsed == 3) {
    if (!test_active) {
        if (!is_script) printf("Error: No test active. Use 'TEST <week>' first.\n");
        err = TRIT_ERR_INPUT;
    } else {
        int qnum = atoi(arg1);
        int answer = atoi(arg2);
        if (qnum < 1 || qnum > QUESTIONS_PER_WEEK || answer < 1 || answer > 4) {
            if (!is_script) printf("Error: Invalid format. Use 'CHECK <qnum> <answer>' (qnum 1-5, answer 1-4)\n");
            log_error(TRIT_ERR_INPUT, "execute_command: CHECK format");
            err = TRIT_ERR_INPUT;
        } else {
            int global_qnum = (test_active - 1) * QUESTIONS_PER_WEEK + (qnum - 1);
            if (test_answers[global_qnum] == 0) questions_answered++;
            test_answers[global_qnum] = answer;
            int correct = check_test_answer(global_qnum, answer);
            if (correct && test_answers[global_qnum] == answer && questions_answered <= QUESTIONS_PER_WEEK) test_score++;
            if (!is_script) printf("Answer recorded. Correct: %s\n", correct ? "Yes" : "No");
            err = TRIT_OK;
        }
    }
}
```

```
else if (strcmp(op, "SCORE") == 0 && parsed == 1) {
    if (!test_active) {
        if (!is_script) printf("Error: No test active. Start with 'TEST <week>'\n");
        err = TRIT_ERR_INPUT;
    } else {
        if (!is_script) {
            printf("Week %d Test Score: %d/%d (%.2f%%)\n", test_active, test_score, QUESTIONS_PER_WEEK,
                   (float)test_score / QUESTIONS_PER_WEEK * 100);
            if (test_score == QUESTIONS_PER_WEEK) printf("Congratulations! Certified for Week %d!\n", test_active);
        }
        err = TRIT_OK;
    }
}
else if (strcmp(op, "PROG") == 0 && parsed >= 2) {
    char script_name[MAX_SCRIPT_NAME];
    char* brace_start = strchr(input, '{');
    char* brace_end = strrchr(input, '}');
    if (!brace_start || !brace_end || script_count >= 100) {
        if (!is_script) printf("Error: Invalid script syntax or too many scripts (max 100)\n");
        log_error(TRIT_ERR_SCRIPT, "execute_command: PROG syntax or limit");
        err = TRIT_ERR_SCRIPT;
    } else {
        sscanf(input + 5, "%19s", script_name);
        Script* script = &scripts[script_count];
        strncpy(script->name, script_name, MAX_SCRIPT_NAME - 1);
        script->name[MAX_SCRIPT_NAME - 1] = '\0';
        script->cmd_count = 0;

        char* cmd_start = brace_start + 1;
        while (cmd_start < brace_end && script->cmd_count < MAX_SCRIPT_CMDS) {
            char* cmd_end = strchr(cmd_start, ';');
            if (!cmd_end || cmd_end > brace_end) cmd_end = brace_end;
            int len = cmd_end - cmd_start;
            while (*cmd_start == ' ' && len > 0) { cmd_start++; len--; }
            if (len > 0) {
                if (len >= MAX_INPUT) len = MAX_INPUT - 1;
                strncpy(script->commands[script->cmd_count], cmd_start, len);
                script->commands[script->cmd_count][len] = '\0';
                script->cmd_count++;
```

```
            }
            cmd_start = cmd_end + 1;
        }
        script_count++;
        if (!is_script) printf("Script '%s' defined\n", script_name);
        err = TRIT_OK;
    }
}
else if ((strcmp(op, "RUN") == 0 || strcmp(op, "r") == 0) && parsed == 2) {
    Script* script = find_script(arg1);
    if (!script) {
        if (!is_script) printf("Error: Script '%s' not found\n", arg1);
        log_error(TRIT_ERR_SCRIPT, "execute_command: RUN script not found");
        err = TRIT_ERR_SCRIPT;
    } else {
        if ((err = run_script(script)) == TRIT_OK) {
            if (!is_script) printf("Script '%s' executed\n", arg1);
        } else {
            if (!is_script) printf("Error: Script execution failed\n");
        }
    }
}
else if (strcmp(op, "NET") == 0 && parsed == 3) {
    char ip[16];
    int port = atoi(arg2);
    if (sscanf(input + 4, "%15s %d", ip, &port) != 2 || port < 1 || port > 65535) {
        if (!is_script) printf("Error: Usage: NET <ip> <port> (e.g., NET 127.0.0.1 8080)\n");
        log_error(TRIT_ERR_INPUT, "execute_command: NET format");
        err = TRIT_ERR_INPUT;
    } else {
        pthread_t net_thread;
        struct NetArgs { char ip[16]; int port; } args;
        strncpy(args.ip, ip, 16);
        args.ip[15] = '\0';
        args.port = port;
        if (pthread_create(&net_thread, NULL, (void*(*)(void*))net_listen, &args) != 0) {
            if (!is_script) printf("Error: Failed to start network thread\n");
            log_error(TRIT_ERR_NETWORK, "execute_command: NET thread");
            err = TRIT_ERR_NETWORK;
```

```
            } else {
                pthread_detach(net_thread); /* Run in background */
                if (!is_script) printf("Network thread started\n");
                err = TRIT_OK;
            }
        }
    }
    else if ((strcmp(op, "export") == 0 || strcmp(op, "json") == 0) && parsed == 2) {
        if ((err = export_json(arg1)) == TRIT_OK) {
            if (!is_script) printf("Exported to %s in JSON format\n", arg1);
        }
    }
    else {
        if (!is_script) printf("Error: Unknown command or invalid arguments\n");
        err = TRIT_ERR_INPUT;
    }

    if (a && !(arg1[0] >= 'A' && arg1[0] <= 'Z' && arg1[1] == '\0')) tritbig_free(a);
    if (b && !(arg2[0] >= 'A' && arg2[0] <= 'Z' && arg2[1] == '\0')) tritbig_free(b);
    return err;
}

TritError run_script(Script* script) {
    for (int i = 0; i < script->cmd_count; i++) {
        char* cmd = script->commands[i];
        if (strncmp(cmd, "IF ", 3) == 0) {
            char cond[MAX_INPUT], then_cmd[MAX_INPUT];
            if (sscanf(cmd, "IF %255s THEN %255[^\n]", cond, then_cmd) != 2) {
                printf("Script Error: Invalid IF syntax\n");
                log_error(TRIT_ERR_SCRIPT, "run_script: IF syntax");
                return TRIT_ERR_SCRIPT;
            }
            TritBigInt* cond_val;
            if (parse_trit_string(cond, &cond_val) != TRIT_OK) {
                printf("Script Error: Invalid condition\n");
                log_error(TRIT_ERR_SCRIPT, "run_script: IF condition");
                return TRIT_ERR_SCRIPT;
            }
            int val = 0;
```

```
    for (int j = 0; j < cond_val->len; j++) val = val * TRIT_MAX + cond_val->digits[j];
    tritbig_free(cond_val);
    if (val != 0) {
        if (execute_command(then_cmd, 1) != TRIT_OK) return TRIT_ERR_SCRIPT;
    }
} else if (strncmp(cmd, "FOR ", 4) == 0) {
    char var[2], start_str[MAX_INPUT], end_str[MAX_INPUT], loop_cmd[MAX_INPUT];
    if (sscanf(cmd, "FOR %1s %255s %255s %255[^\n]", var, start_str, end_str, loop_cmd) != 4) {
        printf("Script Error: Invalid FOR syntax\n");
        log_error(TRIT_ERR_SCRIPT, "run_script: FOR syntax");
        return TRIT_ERR_SCRIPT;
    }
    TritBigInt *start, *end;
    if (parse_trit_string(start_str, &start) != TRIT_OK || parse_trit_string(end_str, &end) != TRIT_OK) {
        printf("Script Error: Invalid FOR range\n");
        log_error(TRIT_ERR_SCRIPT, "run_script: FOR range");
        return TRIT_ERR_SCRIPT;
    }
    int start_val = 0, end_val = 0;
    for (int j = 0; j < start->len; j++) start_val = start_val * TRIT_MAX + start->digits[j];
    for (int j = 0; j < end->len; j++) end_val = end_val * TRIT_MAX + end->digits[j];
    for (int k = start_val; k <= end_val; k++) {
        char val_str[10];
        snprintf(val_str, sizeof(val_str), "%d", k);
        TritBigInt* i_bi;
        if (parse_trit_string(val_str, &i_bi) != TRIT_OK) {
            tritbig_free(start);
            tritbig_free(end);
            return TRIT_ERR_SCRIPT;
        }
        store_variable(var, i_bi);
        if (execute_command(loop_cmd, 1) != TRIT_OK) {
            tritbig_free(start);
            tritbig_free(end);
            return TRIT_ERR_SCRIPT;
        }
    }
    tritbig_free(start);
    tritbig_free(end);
```

```
        } else {
            if (execute_command(cmd, 1) != TRIT_OK) return TRIT_ERR_SCRIPT;
        }
    }
    return TRIT_OK;
}

void print_help() {
    printf("\n=== TritJS-CISA Commands ===\n");
    printf("Arithmetic:\n");
    printf("  add/a <a> <b>  - Add two ternary numbers\n");
    printf("  sub/s <a> <b>  - Subtract b from a\n");
    printf("  mul/m <a> <b>  - Multiply a and b\n");
    printf("  div/d <a> <b>  - Divide a by b\n");
    printf("  pow/p <a> <b>  - Raise a to power b\n");
    printf("  fact/f <a>     - Factorial of a\n");
    printf("Scientific:\n");
    printf("  sqrt <a>       - Square root of a\n");
    printf("  log3 <a>       - Base-3 logarithm of a\n");
    printf("  sin <a>        - Sine of a\n");
    printf("  cos <a>        - Cosine of a\n");
    printf("  tan <a>        - Tangent of a\n");
    printf("  pi             - Pi in base-3\n");
    printf("Stats:\n");
    printf("  stats [quick|merge] - Show mean, mode, median (auto-selects if omitted)\n");
    printf("Memory:\n");
    printf("  <A-Z>=<val>    - Store value in variable (e.g., A=12)\n");
    printf("  recall/rc <n>  - Recall nth last result (0 = latest)\n");
    printf("  clear/cl       - Clear history, variables, and scripts\n");
    printf("Storage:\n");
    printf("  save/sv <file> <key> - Save encrypted state to file (.trit)\n");
    printf("  load/ld <file> <key> - Load encrypted state from file (.trit)\n");
    printf("  export/json <file>   - Export state to JSON\n");
    printf("Networking:\n");
    printf("  NET <ip> <port>      - Listen for ternary data (e.g., NET 127.0.0.1 8080)\n");
    printf("Scripting:\n");
    printf("  PROG <name> {<cmds>} - Define script (e.g., PROG LOOP {add A 1; A=A})\n");
    printf("  RUN/r <name>         - Run named script\n");
    printf("Education:\n");
```

```
    printf("  LEARN <topic>        - Learn a topic (e.g., ADD, SUB, MUL)\n");
    printf("  TEST <week>          - Start test for week (1-8)\n");
    printf("  CHECK <qnum> <ans>   - Submit test answer (qnum 1-5, ans 1-4)\n");
    printf("  SCORE                - Show test score\n");
    printf("General:\n");
    printf("  help/h               - Show this help\n");
    printf("  quit/q               - Exit\n");
    printf("====================\n");
}

void run_calculator() {
    init_audit_log();
    char input[MAX_INPUT];
    printf("=== TritJS-CISA Ternary Cybersecurity Tool ===\n");
    printf("Type 'help' for commands\n");
    while (1) {
        pthread_mutex_lock(&mutex);
        total_mapped_bytes = 0;
        operation_steps = 0;
        pthread_mutex_unlock(&mutex);
        printf("> ");
        if (!fgets(input, MAX_INPUT, stdin)) break;
        input[strcspn(input, "\n")] = 0;
        if (audit_log) fprintf(audit_log, "[%ld] Command: %s\n", time(NULL), input);

        if (strcmp(input, "quit") == 0 || strcmp(input, "q") == 0) break;
        if (strcmp(input, "help") == 0 || strcmp(input, "h") == 0) {
            print_help();
            continue;
        }
        if (strcmp(input, "clear") == 0 || strcmp(input, "cl") == 0) {
            clear_history_and_vars();
            printf("History, variables, scripts, and test state cleared\n");
            continue;
        }
        if (strncmp(input, "stats", 5) == 0) {
            char sort_method[10] = "auto";
            sscanf(input, "%*s %9s", sort_method);
            if (strcmp(sort_method, "quick") != 0 && strcmp(sort_method, "merge") != 0 && strcmp(sort_method, "auto") != 0) {
```

```
            printf("Error: Sort method must be 'quick', 'merge', or omitted (auto)\n");
            continue;
        }
        display_memory_and_stats("stats", sort_method);
        continue;
    }
    if (strncmp(input, "save ", 5) == 0 || strncmp(input, "sv ", 3) == 0) {
        char filename[MAX_FILENAME], key[AES_KEYLEN + 1];
        if (sscanf(input + (input[1] == 'v' ? 3 : 5), "%255s %32s", filename, key) != 2) {
            printf("Error: Usage: save <file> <key> (key max 32 chars)\n");
            continue;
        }
        if (save_state_encrypted(filename, key) == TRIT_OK) printf("State saved to %s\n", filename);
        continue;
    }
    if (strncmp(input, "load ", 5) == 0 || strncmp(input, "ld ", 3) == 0) {
        char filename[MAX_FILENAME], key[AES_KEYLEN + 1];
        if (sscanf(input + (input[1] == 'd' ? 3 : 5), "%255s %32s", filename, key) != 2) {
            printf("Error: Usage: load <file> <key> (key max 32 chars)\n");
            continue;
        }
        if (load_state_encrypted(filename, key) == TRIT_OK) printf("State loaded from %s\n", filename);
        continue;
    }
    if (strncmp(input, "export ", 7) == 0 || strncmp(input, "json ", 5) == 0) {
        char filename[MAX_FILENAME];
        if (sscanf(input + (input[1] == 'x' ? 7 : 5), "%255s", filename) != 1) {
            printf("Error: Usage: export <file>\n");
            continue;
        }
        if (export_json(filename) == TRIT_OK) printf("State exported to %s\n", filename);
        continue;
    }
    if (strncmp(input, "NET ", 4) == 0) {
        char ip[16];
        int port;
        if (sscanf(input + 4, "%15s %d", ip, &port) != 2 || port < 1 || port > 65535) {
            printf("Error: Usage: NET <ip> <port> (e.g., NET 127.0.0.1 8080)\n");
            continue;
```

```
    }
    pthread_t net_thread;
    struct NetArgs { char ip[16]; int port; } args;
    strncpy(args.ip, ip, 16);
    args.ip[15] = '\0';
    args.port = port;
    if (pthread_create(&net_thread, NULL, (void*(*)(void*))net_listen, &args) != 0) {
        printf("Error: Failed to start network thread\n");
        log_error(TRIT_ERR_NETWORK, "run_calculator: NET thread");
    } else {
        pthread_detach(net_thread);
        printf("Network listener started on %s:%d\n", ip, port);
    }
    continue;
}
if (strncmp(input, "LEARN ", 6) == 0) {
    char topic[MAX_INPUT];
    if (sscanf(input + 6, "%255s", topic) != 1) {
        printf("Error: Usage: LEARN <topic> (e.g., LEARN ADD)\n");
        continue;
    }
    learn_topic(topic);
    continue;
}
if (strncmp(input, "TEST ", 5) == 0) {
    int week;
    if (sscanf(input + 5, "%d", &week) != 1) {
        printf("Error: Usage: TEST <week> (1-8)\n");
        continue;
    }
    execute_command(input, 0);
    continue;
}
if (strncmp(input, "CHECK ", 6) == 0) {
    execute_command(input, 0);
    continue;
}
if (strcmp(input, "SCORE") == 0) {
    execute_command(input, 0);
```

```
            continue;
        }
        if (strncmp(input, "PROG ", 5) == 0) {
            execute_command(input, 0);
            continue;
        }
        if (strncmp(input, "RUN ", 4) == 0 || strncmp(input, "r ", 2) == 0) {
            execute_command(input, 0);
            continue;
        }
        execute_command(input, 0);
    }
    clear_history_and_vars();
    if (audit_log) fclose(audit_log);
}

@*2 Main Function.
@c
int main() {
    if (pthread_mutex_init(&mutex, NULL) != 0) {
        fprintf(stderr, "Error: Failed to initialize mutex\n");
        return 1;
    }
    run_calculator();
    pthread_mutex_destroy(&mutex);
    return 0;
}
```

TritJS-CISA Training Guide: Mastering Ternary Logic for Cybersecurity

Version: March 01, 2025

Introduction

Welcome to the `TritJS-CISA` Training Guide, developed for the Cybersecurity and Infrastructure Security Agency (CISA). As cyber threats evolve, understanding alternative computational frameworks like ternary (base-3) logic becomes critical. Unlike binary systems (base-2), ternary uses three states (0, 1, 2), offering unique advantages in obfuscation, efficiency, and resilience. `TritJS-CISA` is a powerful tool designed to perform ternary arithmetic, scientific calculations, statistical analysis, and scripting—all within an interactive CLI.

This guide aims to equip CISA personnel with hands-on skills in ternary logic, enabling you to:

- Analyze and manipulate data in a non-binary format.
- Leverage `TritJS-CISA` for cybersecurity tasks like anomaly detection and data encoding.
- Prepare for emerging threats that exploit unconventional computational methods.

No prior ternary knowledge is required—just a basic understanding of cybersecurity concepts and CLI usage.

Learning Objectives

By the end of this training, you will:

1. **Understand Ternary Basics**: Grasp the fundamentals of base-3 arithmetic and its differences from binary.
2. **Perform Ternary Operations**: Use `TritJS-CISA` for arithmetic, scientific, and statistical calculations.
3. **Automate Tasks**: Write and execute scripts to streamline repetitive operations.
4. **Apply to Cybersecurity**: Encode data, analyze patterns, and manage state for practical scenarios.
5. **Troubleshoot Issues**: Interpret errors and logs to resolve common problems.

Training Lessons

Lesson 1: Introduction to Ternary Logic

Duration: 15 minutes
Objective: Learn the basics of ternary numbers and their representation in `TritJS-CISA`.
Concepts:

- Ternary uses digits 0, 1, 2 (trits) instead of 0, 1 (bits).
- Conversion: Decimal 5 = Binary `101` = Ternary `12` ($1\times3^1 + 2\times3^0$).
- Negative numbers use a leading `-` (e.g., `-12`).

Steps:

1. Start `TritJS-CISA`:`bash`
   `./tritjs_cisa`
   Output: `=== TritJS-CISA Ternary Calculator ===`.
2. View commands:
   `> help`
   - Note arithmetic commands: `add`, `sub`, `mul`, etc.
3. Try a simple addition:
   `> add 1 1`
4. `2`
   - Explanation: 1 + 1 = 2 (no carry in ternary).

Key Takeaway: Ternary logic is intuitive once you adjust to the three-digit system.

Lesson 2: Basic Arithmetic Operations

Duration: 20 minutes

Objective: Perform core ternary calculations using `TritJS-CISA`.
Concepts:

- Arithmetic operations mirror decimal but use base-3 rules (e.g., 2 + 2 = 11 due to carry).

Steps:

1. **Addition**:
   `> add 12 2`
2. `21`
   - Breakdown: Units (2 + 2 = 11, carry 1), Threes (1 + 1 = 2), Result = 21.

3. **Subtraction**:
   `> sub 21 12`
4. `2`
   - Breakdown: Units (1 - 2 = borrow 1, 11 - 2 = 2), Threes (1 - 1 = 0).
5. **Multiplication**:
   `> mul 11 2`
6. `22`
   - Breakdown: 11 × 2 = 20 + 2 = 22.
7. **Division**:
   `> div 21 2`
8. `10.1 r 1`
   - Explanation: Quotient 10.1 (3 trits precision), remainder 1.
9. **Exponentiation**:
   `> pow 2 2`
10. `11`
    - Explanation: $2^2 = 4$ decimal = 11 ternary.

Exercise:

- Compute: `add 20 11`, `sub 100 21`, `mul 12 2`.
- Answers: `101`, `12`, `101`.

Key Takeaway: Ternary arithmetic follows familiar patterns with base-3 adjustments.

Lesson 3: Scientific Functions

Duration: 20 minutes
Objective: Explore advanced ternary calculations for cybersecurity applications.
Concepts:

- Scientific functions approximate results in ternary, useful for encoding or modeling.

Steps:

1. **Square Root**:
   `> sqrt 12`
2. `11.1`
   - Approximation of √5 ≈ 11.1 ternary (3 trits).
3. **Logarithm (Base-3)**:
   `> log3 100`
4. `2.002`
   - $\log_3(9) \approx 2$, with fractional precision.
5. **Trigonometry**:
   `> sin 2`
6. `0.2`
   - Input scaled as radians × π/10, output approximated.
7. **Pi**:
   `> pi`
8. `10010221`
   - Ternary π (8 trits).

Exercise:

- Calculate: `sqrt 21`, `log3 12`, `cos 1`.
- Answers: ~`12.1, ~1.112, ~1.1`.

Key Takeaway: Scientific functions extend ternary utility beyond basic math.

Lesson 4: Statistical Analysis

Duration: 15 minutes
Objective: Use `stats` to analyze ternary data patterns.
Concepts:

- `stats` computes mean, mode, median from history, auto-selecting Quicksort or Mergesort.

Steps:

1. Build history:
   ```
   > add 1 1
   ```
2. `2`
3. `> mul 2 1`
4. `2`
5. `> add 11 2`
6. `20`
7. Run statistics:
   ```
   > stats
   ```
8. `Mean: 1.33 | Mode: 2 | Median: 1.00 | Total Trits: 6 | Sort: merge`
   - Mean: Average trit value.
   - Mode: Most frequent (2 appears twice).
   - Median: Middle value when sorted.
9. Specify sort method:
   ```
   > stats quick
   ```
10. `Mean: 1.33 | Mode: 2 | Median: 1.00 | Total Trits: 6 | Sort: quick`

Exercise:

- Add `10`, `11`, `12` to history, then run `stats`.
- Answer: Mean ≈ 1.33, Mode = -1 (no repeat), Median ≈ 11.

Key Takeaway: Stats reveal data trends, critical for anomaly detection.

Lesson 5: Scripting and Automation

Duration: 25 minutes

Objective: Automate tasks with `PROG` and `RUN`.

Concepts:

- Scripts use `IF` and `FOR` for conditional and iterative logic.

Steps:

1. Define a simple script:
   ```
   > A=1
   ```
2. `A stored`
3. `> PROG INCR {add A 1; A=A}`
4. `Script 'INCR' defined`
5. `> RUN INCR`
6. `Script 'INCR' executed`
7. `> recall 0`
8. `2`
9. Use a loop:
   ```
   > A=0
   ```
10. `A stored`
11. `> PROG SUM {FOR I 1 2 {add A I; A=A}}`
12. `Script 'SUM' defined`
13. `> RUN SUM`
14. `Script 'SUM' executed`
15. `> recall 0`
16. `10`
    - Explanation: Adds 1, then 2 to A (0 + 1 + 2 = 10 ternary).
17. Conditional script:
    ```
    > A=2
    ```
18. `A stored`
19. `> PROG CHECK {IF A THEN add A 1; A=A}`
20. `Script 'CHECK' defined`
21. `> RUN CHECK`
22. `Script 'CHECK' executed`
23. `> recall 0`
24. `10`

Exercise:

- Write a script `DOUBLE` to multiply A by 2, run it with A=11.
- Answer: `22`.

Key Takeaway: Scripting automates repetitive cybersecurity tasks.

Lesson 6: State Management and Application

Duration: 20 minutes
Objective: Save and load states for continuity and analysis.
Concepts:

- `save`/`load` preserves history, variables, and scripts.

Steps:

1. Set up state:
   ```
   > A=12
   ```
2. `A stored`
3. `> add 11 2`
4. `20`
5. Save state:
   ```
   > save training.trit
   ```
6. `State saved to training.trit`
7. Clear and restore:
   ```
   > clear
   ```
8. `History, variables, and scripts cleared`
9. `> load training.trit`
10. `State loaded from training.trit`
11. `> recall 0`
12. `20`

Exercise:

- Save a state with A=21 and history `11`, `22`, then load after `clear`.

- Verify: `recall 0` returns 22.

Key Takeaway: State management ensures operational continuity.

Practical Exercise: Cybersecurity Scenario

Scenario: You're monitoring a ternary-encoded sensor (values 0-2). An anomaly (e.g., sudden spike) may indicate a cyber attack.

Task:

1. Record readings: 1, 2, 1.
2. Analyze with `stats`.
3. Script a check for values > 1.
4. Save the state.

Solution:

```
> add 1 0
1
> add 2 0
2
> add 1 0
1
> stats
Mean: 1.33 | Mode: 1 | Median: 1.00 | Total Trits: 3 | Sort: quick
> A=1
A stored
> PROG ALERT {IF A THEN add A 1; A=A}
Script 'ALERT' defined
> RUN ALERT
Script 'ALERT' executed
> recall 0
```

```
2
> save sensor.trit
State saved to sensor.trit
```
Analysis: If A exceeds 1 (e.g., becomes 2), it triggers an alert-worthy increment.

Conclusion

Congratulations! You've mastered `TritJS-CISA`'s ternary logic and CLI features:

- Performed arithmetic and scientific calculations.
- Analyzed data with statistics.
- Automated tasks with scripts.
- Managed state for continuity.

Next Steps:

- Apply these skills to encode IoT data or detect anomalies (see CISA Use Cases).
- Explore logs (`/var/log/tritjs_cisa.log`) for error diagnostics.
- Experiment with larger scripts or complex calculations.

For advanced training or support, consult CISA's technical team or the `tritjs_cisa.tex` documentation.

This guide provides a structured, hands-on approach to learning `TritJS-CISA`, directly supporting CISA's educational goals in cybersecurity resilience. Let me know if you'd like additional modules or adjustments!