AXILOG

CI-STRUCT-001

Certifiable Inference Data Structure Specification

Final v1.0 L2 [D1]

Last updated 20 January 2026
Changelog 1 entry

Contents

Traceability: All structures in this document are derived from CI-MATH-001 and implemented in header files under `include/`.

---

§1 Introduction

§1.1 Purpose

This document specifies all data structures for certifiable-inference. Every structure is derived from the mathematical definitions in CI-MATH-001 and serves as the authoritative reference for implementation.

§1.2 Scope

This specification covers:

- Fixed-point types and constants
- Matrix structures
- Convolution parameters
- Pooling parameters
- Activation parameters
- Hash table structures
- Fault flags
- Result codes

§1.3 Notation

Notation	Meaning
`uint8_t[32]`	32-byte array (SHA-256 hash)
`Q16.16`	Fixed-point: 16 integer bits, 16 fractional bits
`SHALL`	Mandatory requirement
`→`	Maps to / derived from

§1.4 Document Alignment

Topic	CI-MATH-001	CI-STRUCT-001	Implementation
Fixed-point	§2	§3	`fixed_point.h`
Matrix	§3	§4	`matrix.h`
Activation	§4	§5	`activations.h`
Convolution	§5	§6	`convolution.h`
Pooling	§6	§7	`pooling.h`
Hash table	§7	§8	`deterministic_hash.h`
Fault flags	§8	§9	`fixed_point.h`

---

§2 Type Definitions

§2.1 Standard Integer Types

All implementations SHALL use `<stdint.h>` for portable integer types:

```
#include <stdint.h>
#include <stdbool.h>

/* Required types */
int8_t      /* Signed 8-bit */
int16_t     /* Signed 16-bit */
int32_t     /* Signed 32-bit */
int64_t     /* Signed 64-bit */
uint8_t     /* Unsigned 8-bit */
uint16_t    /* Unsigned 16-bit */
uint32_t    /* Unsigned 32-bit */
uint64_t    /* Unsigned 64-bit */
```

§2.2 Boolean Type

```
#include <stdbool.h>

/* Boolean values */
true  = 1
false = 0
```

---

§3 Fixed-Point Types

Traceability: CI-MATH-001 §2

§3.1 Core Type Definition

```
/**
 * @brief Q16.16 fixed-point number.
 * @traceability CI-MATH-001 §2.1
 *
 * Format: Signed 32-bit integer
 * - Bits [31:16]: Integer part (signed)
 * - Bits [15:0]: Fractional part
 *
 * Range: [-32768.0, +32767.99998]
 * Precision: 1/65536 ≈ 0.0000153
 */
typedef int32_t fixed_t;
```

§3.2 Constants

Traceability: CI-MATH-001 §2.2

```
/** Number of fractional bits */
#define FIXED_SHIFT 16

/** 1.0 in Q16.16 */
#define FIXED_ONE (1 << FIXED_SHIFT)  /* 65536 = 0x00010000 */

/** 0.5 in Q16.16 (used for rounding) */
#define FIXED_HALF (1 << (FIXED_SHIFT - 1))  /* 32768 = 0x00008000 */

/** 0.0 in Q16.16 */
#define FIXED_ZERO (0)

/** Smallest positive value (1 LSB) */
#define FIXED_EPS (1)

/** Maximum representable value */
#define FIXED_MAX INT32_MAX  /* 0x7FFFFFFF */

/** Minimum representable value */
#define FIXED_MIN INT32_MIN  /* 0x80000000 */
```

§3.3 Value Table

Constant	Decimal Value	Q16.16 Hex	Q16.16 Decimal
FIXED_ONE	1.0	0x00010000	65536
FIXED_HALF	0.5	0x00008000	32768
FIXED_ZERO	0.0	0x00000000	0
FIXED_EPS	0.0000153	0x00000001	1
FIXED_MAX	32767.99998	0x7FFFFFFF	2147483647
FIXED_MIN	-32768.0	0x80000000	-2147483648

§3.4 Common Precomputed Values

```
/** π in Q16.16 (3.14159...) */
#define FIXED_PI 205887  /* 0x00032440 */

/** e in Q16.16 (2.71828...) */
#define FIXED_E 178145   /* 0x0002B7E1 */

/** 0.01 in Q16.16 (common Leaky ReLU alpha) */
#define FIXED_ALPHA_001 655  /* 0x0000028F */
```

---

§4 Matrix Structures

Traceability: CI-MATH-001 §3

§4.1 Matrix Structure

```
/**
 * @brief Fixed-point matrix with caller-provided storage.
 * @traceability CI-MATH-001 §3.1
 *
 * Memory layout: Row-major order
 * Element access: data[i * cols + j] for M[i][j]
 *
 * IMPORTANT: This structure does NOT own the data buffer.
 * The caller MUST ensure the buffer outlives the matrix.
 */
typedef struct {
    fixed_t *data;      /**< Pointer to caller-owned buffer */
    int rows;           /**< Number of rows (height) */
    int cols;           /**< Number of columns (width) */
} fx_matrix_t;
```

§4.2 Memory Layout

```
Matrix M (3×4):
  Row 0: [M₀₀] [M₀₁] [M₀₂] [M₀₃]
  Row 1: [M₁₀] [M₁₁] [M₁₂] [M₁₃]
  Row 2: [M₂₀] [M₂₁] [M₂₂] [M₂₃]

Linear storage (row-major):
  data[0..3]  = Row 0
  data[4..7]  = Row 1
  data[8..11] = Row 2

Index formula:
  M[i][j] = data[i * cols + j]
```

§4.3 Size Requirements

Dimensions	Elements	Bytes (Q16.16)
1×1	1	4
3×3	9	36
28×28	784	3136
64×64	4096	16384
784×128	100352	401408

§4.4 Initialization

```
/**
 * @brief Initialize a matrix with caller-provided buffer.
 * @traceability CI-MATH-001 §3.2
 *
 * @param m Matrix to initialize (output)
 * @param buffer Pre-allocated buffer (at least rows×cols elements)
 * @param rows Number of rows
 * @param cols Number of columns
 *
 * @pre buffer ≠ NULL
 * @pre rows > 0 && cols > 0
 * @post m→data = buffer
 * @post m→rows = rows
 * @post m→cols = cols
 */
void fx_matrix_init(fx_matrix_t *m, fixed_t *buffer, int rows, int cols);
```

---

§5 Activation Structures

Traceability: CI-MATH-001 §4

§5.1 Activation Type Enumeration

```
/**
 * @brief Supported activation functions.
 * @traceability CI-MATH-001 §4
 */
typedef enum {
    CI_ACT_NONE = 0,        /**< No activation (linear) */
    CI_ACT_RELU,            /**< ReLU: max(0, x) */
    CI_ACT_LEAKY_RELU,      /**< Leaky ReLU: x if x>0 else α×x */
    CI_ACT_THRESHOLD        /**< Binary threshold: x > θ ? 1 : 0 */
} ci_activation_t;
```

§5.2 Activation Parameters

```
/**
 * @brief Parameters for activation functions.
 * @traceability CI-MATH-001 §4.2
 */
typedef struct {
    ci_activation_t type;   /**< Activation function type */
    fixed_t alpha;          /**< Leaky ReLU slope (default: 655 = 0.01) */
    fixed_t threshold;      /**< Threshold value for binary activation */
} ci_activation_params_t;
```

§5.3 Default Parameters

Activation	Default Parameters
NONE	—
RELU	—
LEAKY_RELU	alpha = 655 (0.01)
THRESHOLD	threshold = 0

---

§6 Convolution Structures

Traceability: CI-MATH-001 §5

§6.1 Convolution Parameters

```
/**
 * @brief Parameters for 2D convolution.
 * @traceability CI-MATH-001 §5.1
 *
 * Currently supports:
 * - Valid padding only (no padding)
 * - Stride 1 only
 * - Single channel only
 */
typedef struct {
    int kernel_h;           /**< Kernel height (typically 3, 5, 7) */
    int kernel_w;           /**< Kernel width (typically 3, 5, 7) */
    int stride_h;           /**< Vertical stride (currently must be 1) */
    int stride_w;           /**< Horizontal stride (currently must be 1) */
} ci_conv_params_t;
```

§6.2 Output Dimension Calculation

Traceability: CI-MATH-001 §5.1

```
/**
 * Calculate convolution output dimensions.
 *
 * For valid padding:
 *   out_h = in_h - kernel_h + 1
 *   out_w = in_w - kernel_w + 1
 */
#define CI_CONV_OUT_H(in_h, k_h) ((in_h) - (k_h) + 1)
#define CI_CONV_OUT_W(in_w, k_w) ((in_w) - (k_w) + 1)
```

§6.3 Common Kernel Sizes

Kernel	Input 28×28	Output Size	Operations
3×3	28×28	26×26	6084
5×5	28×28	24×24	14400
7×7	28×28	22×22	23716

§7 Pooling Structures

Traceability: CI-MATH-001 §6

§7.1 Pooling Type Enumeration

```
/**
 * @brief Supported pooling operations.
 * @traceability CI-MATH-001 §6
 */
typedef enum {
    CI_POOL_MAX = 0,        /**< Maximum value in window */
    CI_POOL_AVG             /**< Average value in window (future) */
} ci_pool_type_t;
```

§7.2 Pooling Parameters

```
/**
 * @brief Parameters for pooling operations.
 * @traceability CI-MATH-001 §6.1
 *
 * Currently supports:
 * - 2×2 window with stride 2 only
 * - Max pooling only
 */
typedef struct {
    ci_pool_type_t type;    /**< Pooling type */
    int window_h;           /**< Window height (must be 2) */
    int window_w;           /**< Window width (must be 2) */
    int stride_h;           /**< Vertical stride (must equal window_h) */
    int stride_w;           /**< Horizontal stride (must equal window_w) */
} ci_pool_params_t;
```

§7.3 Output Dimension Calculation

Traceability: CI-MATH-001 §6.1

```
/**
 * Calculate pooling output dimensions.
 *
 * For non-overlapping pooling (stride = window):
 *   out_h = in_h / window_h
 *   out_w = in_w / window_w
 */
#define CI_POOL_OUT_H(in_h, win_h) ((in_h) / (win_h))
#define CI_POOL_OUT_W(in_w, win_w) ((in_w) / (win_w))
```

---

§8 Hash Table Structures

Traceability: CI-MATH-001 §7

§8.1 Result Codes

```
/**
 * @brief Hash table operation results.
 * @traceability CI-MATH-001 §7.5
 */
typedef enum {
    D_TABLE_OK = 0,         /**< Operation succeeded */
    D_TABLE_KEY_EXISTS,     /**< Insert failed: key already exists */
    D_TABLE_NOT_FOUND,      /**< Get failed: key not found */
    D_TABLE_FULL,           /**< Insert failed: table at capacity */
    D_TABLE_ERROR           /**< Generic error */
} d_table_res_t;
```

§8.2 Entry Structure

```
/**
 * @brief Single hash table entry.
 * @traceability CI-MATH-001 §7.5
 */
typedef struct {
    char key[D_TABLE_MAX_KEY_LEN];  /**< Null-terminated key string */
    int32_t value;                   /**< Associated value */
    bool occupied;                   /**< True if slot contains valid data */
} d_table_entry_t;

/** Maximum key length including null terminator */
#define D_TABLE_MAX_KEY_LEN 64
```

§8.3 Table Structure

```
/**
 * @brief Deterministic hash table with insertion-order iteration.
 * @traceability CI-MATH-001 §7.1, §7.4
 *
 * Memory layout:
 * - entries[]: Hash table slots (linear probing)
 * - insertion_order[]: Indices for deterministic iteration
 *
 * The table uses caller-provided buffer for both arrays.
 */
typedef struct {
    d_table_entry_t *entries;       /**< Array of hash slots */
    uint32_t *insertion_order;      /**< Order of insertions for iteration */
    uint32_t capacity;              /**< Maximum entries */
    uint32_t count;                 /**< Current entry count */
} d_table_t;
```

§8.4 Memory Requirements

```
/**
 * Calculate buffer size for hash table.
 *
 * @param capacity Maximum number of entries
 * @return Required buffer size in bytes
 */
#define D_TABLE_BUFFER_SIZE(capacity) \
    ((capacity) * sizeof(d_table_entry_t) + (capacity) * sizeof(uint32_t))
```

Capacity	Entry Size	Order Size	Total Bytes
8	576	32	608
16	1152	64	1216
32	2304	128	2432
64	4608	256	4864
128	9216	512	9728

§8.5 Callback Type

```
/**
 * @brief Callback function for hash table iteration.
 * @traceability CI-MATH-001 §7.4
 *
 * @param key Entry key (null-terminated string)
 * @param value Entry value
 */
typedef void (*d_table_iter_fn)(const char *key, int32_t value);
```

---

§9 Fault Flags

Traceability: CI-MATH-001 §8

§9.1 Fault Flag Structure

```
/**
 * @brief Sticky fault flags for arithmetic operations.
 * @traceability CI-MATH-001 §8.1
 *
 * Flags are STICKY: once set, they remain set until explicitly cleared.
 * This enables fault detection across a sequence of operations.
 */
typedef struct {
    uint32_t overflow    : 1;   /**< Result saturated high (> INT32_MAX) */
    uint32_t underflow   : 1;   /**< Result saturated low (< INT32_MIN) */
    uint32_t div_zero    : 1;   /**< Division by zero attempted */
    uint32_t domain      : 1;   /**< Input outside valid domain */
    uint32_t precision   : 1;   /**< Significant precision loss */
    uint32_t _reserved   : 27;  /**< Reserved for future use */
} ci_fault_flags_t;
```

§9.2 Fault Flag Semantics

Flag	Set When	Value Returned
`overflow`	Result > INT32_MAX	INT32_MAX
`underflow`	Result < INT32_MIN	INT32_MIN
`div_zero`	Divisor == 0	0
`domain`	Invalid input (e.g., negative to sqrt)	0
`precision`	Significant bits lost in conversion	Rounded value

§9.3 Helper Functions

```
/**
 * @brief Check if any fault occurred.
 * @traceability CI-MATH-001 §8.3
 */
static inline bool ci_has_fault(const ci_fault_flags_t *f) {
    return f->overflow || f->underflow || f->div_zero ||
           f->domain || f->precision;
}

/**
 * @brief Clear all fault flags.
 * @traceability CI-MATH-001 §8.3
 */
static inline void ci_clear_faults(ci_fault_flags_t *f) {
    f->overflow = 0;
    f->underflow = 0;
    f->div_zero = 0;
    f->domain = 0;
    f->precision = 0;
}
```

---

§10 Layer Structures (Future)

§10.1 Dense Layer

```
/**
 * @brief Dense (fully connected) layer parameters.
 * @traceability CI-MATH-001 §4.4
 *
 * Computation: Y = activation(X × W + b)
 */
typedef struct {
    fx_matrix_t weights;            /**< Weight matrix (in_features ×
out_features) */
    fx_matrix_t bias;               /**< Bias vector (1 × out_features) */
    ci_activation_params_t act;     /**< Activation function */
} ci_dense_layer_t;
```

§10.2 Convolutional Layer

```
/**
 * @brief Convolutional layer parameters.
 * @traceability CI-MATH-001 §5
 *
 * Single-channel 2D convolution with optional activation.
 */
typedef struct {
    fx_matrix_t kernel;             /**< Convolution kernel */
    fixed_t bias;                   /**< Scalar bias */
    ci_conv_params_t conv;          /**< Convolution parameters */
    ci_activation_params_t act;     /**< Activation function */
} ci_conv_layer_t;
```

§10.3 Pooling Layer

```
/**
 * @brief Pooling layer parameters.
 * @traceability CI-MATH-001 §6
 */
typedef struct {
    ci_pool_params_t pool;          /**< Pooling parameters */
} ci_pool_layer_t;
```

---

§11 Memory Layout Requirements

§11.1 Alignment

All structures SHALL be naturally aligned:

- 1-byte fields: no alignment requirement
- 2-byte fields: 2-byte aligned
- 4-byte fields: 4-byte aligned
- 8-byte fields: 8-byte aligned
- Pointers: platform word-aligned (4 or 8 bytes)

§11.2 Padding

Structures MAY contain compiler-inserted padding for alignment. Do NOT hash structures directly; serialize field-by-field for deterministic hashing.

§11.3 Static Allocation

All structures are designed for static allocation. No `malloc` / `free` in hot paths.

§11.4 Buffer Ownership

Structures containing pointers (e.g., `fx_matrix_t`) do NOT own their buffers. The caller is responsible for:

1. Allocating buffers of sufficient size
2. Ensuring buffers outlive structures
3. Deallocating buffers when no longer needed

---

§12 Serialization Rules

Traceability: CI-MATH-001 Appendix B

§12.1 Integer Encoding

All integers are serialized as **little-endian**:

Type	Bytes	Encoding
`int8_t`	1	Direct (signed)
`uint8_t`	1	Direct
`int16_t`	2	Little-endian
`uint16_t`	2	Little-endian
`int32_t` / `fixed_t`	4	Little-endian
`uint32_t`	4	Little-endian
`int64_t`	8	Little-endian
`uint64_t`	8	Little-endian

§12.2 Matrix Serialization

```
serialize_matrix(m):
    rows[4]         /* int32_t, little-endian */
    cols[4]         /* int32_t, little-endian */
    data[rows×cols×4]  /* Row-major, each element 4 bytes LE */
```

§12.3 String Serialization

Strings are serialized as:

```
length[4]       /* uint32_t, NOT including null terminator */
chars[length]   /* UTF-8 bytes, NO null terminator */
```

§12.4 Boolean Serialization

```
bool → uint8_t: false = 0x00, true = 0x01
```

---

§13 Implementation Checklist

§13.1 Required Types (`fixed_point.h`)

- ☑ fixed_t
- ☑ FIXED_SHIFT
- ☑ FIXED_ONE
- ☑ FIXED_HALF
- ☑ FIXED_ZERO
- ☑ FIXED_EPS
- ☑ FIXED_MAX
- ☑ FIXED_MIN

§13.2 Required Types (`matrix.h`)

- ☑ fx_matrix_t
- ☑ fx_matrix_init()
- ☑ fx_matrix_mul()
- ☑ fx_matrix_add()

§13.3 Required Types (`activations.h`)

- ☑ ci_activation_t
- ☑ ci_activation_params_t
- ☑ fx_relu()
- ☑ fx_leaky_relu()
- ☑ fx_add_bias()

§13.4 Required Types (`convolution.h`)

- ☑ ci_conv_params_t
- ☑ fx_conv2d()
- ☑ CI_CONV_OUT_H()
- ☑ CI_CONV_OUT_W()

§13.5 Required Types (`pooling.h`)

- ☑ ci_pool_type_t
- ☑ ci_pool_params_t
- ☑ fx_maxpool_2x2()
- ☑ CI_POOL_OUT_H()
- ☑ CI_POOL_OUT_W()

§13.6 Required Types (`deterministic_hash.h`)

- ☑ d_table_res_t
- ☑ d_table_entry_t
- ☑ d_table_t
- ☑ d_table_iter_fn
- ☑ d_table_init()
- ☑ d_table_insert()
- ☑ d_table_get()
- ☑ d_table_iterate()

Appendix A: Size Summary

Structure	Size (bytes)	Notes
`fixed_t`	4	int32_t alias
`fx_matrix_t`	12-16	Pointer + 2 ints
`ci_activation_params_t`	12	enum + 2 fixed_t
`ci_conv_params_t`	16	4 ints
`ci_pool_params_t`	20	enum + 4 ints
`d_table_entry_t`	72	64-char key + value + bool + padding
`d_table_t`	24-32	2 pointers + 2 uint32_t
`ci_fault_flags_t`	4	Bitfield

Appendix B: Memory Budget Example

B.1 MNIST Inference (784→64→10)

Component	Elements	Bytes
Input (28×28)	784	3,136
W1 (784×64)	50,176	200,704
b1 (1×64)	64	256
Hidden (1×64)	64	256
W2 (64×10)	640	2,560
b2 (1×10)	10	40
Output (1×10)	10	40
Total	**51,748**	**206,992**

All buffers can be statically allocated. Zero runtime heap usage.

Revision History

Version	Date	Author	Changes
1.0.0	2026-01-20	William Murray	Initial release

Document Status: Final

Implementation: `include/*.h` files

Traceability: All structures trace to CI-MATH-001 sections as noted.