

# Abstraction, Specialization and Intelligence

Ankush Yadav

Co-Founder - AuctaSapience

ankush@auctasapience.com

Ankit Yadav

Co-Founder - AuctaSapience

ankit@auctasapience.com

## Abstract

Artificial intelligence is rapidly transforming human civilization and reshaping our understanding of nature, the universe, and the mind. The objective of this paper is to develop an extremely capable AI agent that can perform all tasks currently carried out by humans. The agent is designed to be self-learning and continuously evolving through its own experiences, enabling it to handle highly complex tasks. This goal is achieved through a novel architecture inspired by abstract mathematics and natural systems.

## 1 Introduction

Let's get straight to our goal without wasting much time. Our goal is to make an AI agent that is as capable as a human. So, what does being as capable as a human mean? One possible interpretation is that it should be able to perform tasks a human can do—that is, we develop datasets and train our model to do those tasks, and over time it learns patterns and generalizes based on them. This is a good approach from a purely machine learning point of view, and most labs are pursuing this direction.

Another approach is to develop a unit that is as abstract as possible and can be specialized to perform any given task. I would call this the "cell approach", where we create cells and their connections, with each individual cell representing a different causal state. Each cell performs its own function and improves over time, and the network of cells combines to form a coherent agent. Mathematically, we can call these cells objects too. So, we want to make the most abstract object units, which will combine coherently to form an agent.

## 2 The Object Approach

### 2.1 What is an "Object"?

So, we should start with the definition of objects. Let's stop thinking about AI for the moment and

think in the most abstract terms. What is an object in its most abstract form mathematically? This question can be translated to an even better question: What is anything? or What is? These are the questions that have been asked by philosophers and mathematicians for centuries. While we might not answer this question exactly, let's try to answer it as far as we can.

What is Something? To answer this, we should ask ourselves what we would need to write or say to fully define anything. What is the minimum information you need about anything to describe it fully? One could say that, to fully describe something (let's call our something an "Object," as "something" is too ambiguous), we need to know how it behaves under different conditions or inputs.

So, one conclusion we might reach by thinking mathematically is that an "Anything" or an "Object" is a function—its behavior or output depends on its inputs. What it does in different circumstances depends on that object and its inputs. But is it only a function? That doesn't seem to be the case. An object also has a location with reference to others, since we need to reach the object to interact with it (i.e., to call that function). So, location and functionality seem to be the key information we need to fully define an object.

Another thing we should think about is that location is defined with reference to other objects—there can't be any center anywhere, as that causes the network to become inefficient due to the creation of a bottleneck at the center (in chemistry, you might have studied the rate-determining step). If we define objects in reference to fixed objects, then during search and other operations, the fixed points could become bottlenecks in that network, increasing latency.

We can think that we have found all the basic quantities needed to define an object, but it is not true. Whatever we are saying is also in language, so it is not exact and basically points to some object

or some combination of objects (an intelligent combination of objects, for example: "Imagine Salman Khan and Shahrukh Khan, Salman Khan's looks and Shahrukh Khan's charm"). What I mean is, language points to an object and then changes parts of it with other objects, if you can understand. So, our language depends on a reference—a common data, action, property, etc.—basically a constant in terms of syntactical definition across interacting cells or objects.

Yeah, I can understand that the last paragraph was not coherent and shifted dynamics too fast. Let's see that again. Basically, language is somewhat divided into two parts. One is semantics and the other is logic-based syntax (mostly—sometimes illogical too in natural language).

To understand it more clearly: semantics is basically a representation in our mind, or for an object, we can say it is in its content and location values—not in the interaction definition between objects or within itself.

We can say syntax is a logical or illogical definition (just satisfies the property of definition!) for external interactions with other objects or internal interactions within itself that are definable. More precisely syntax is a set of coded actions that determine how two objects can interact, what an object would do within itself on interaction, and how it would return the interaction.

So, to conclude what I started for confusion: we can combine both semantically and syntactically, but it is hard for us to notice semantics based on "object" or "unit"-specific definitions. This is in the sense that it is the maximum we can write syntactically in terms of definitions for the AI agent. Other things will need to be learned by training it using a generalized learning algorithm for both further syntactical definitions and semantic understanding from data or via a reasoning model capable of understanding and reasoning in this language.

## 2.2 Object Data Structure

Let's denote our object using the symbol "{}", and denote the syntactical components that define it using "[]" for data and "()=>{}" for actions/functions. For example, an object might be represented as: "{} [data], (Data)=>{ return Data } {}".

## 2.3 A Managed Object-Oriented Framework for C

To translate our abstract definition of an "Object" into a concrete and powerful system, a simple C 'struct' is insufficient. The philosophical requirements—encapsulating state (semantics), behavior (syntax), and relationships (location)—demand more than just a data container. They necessitate a complete **runtime environment** that manages the entire lifecycle of objects, provides a mechanism for polymorphic interaction, and ensures system-level robustness.

To this end, we developed the **AGENK Object Framework**, a managed object-oriented system for C. Instead of relying on language-native features, the framework provides these capabilities as a library and a set of conventions. The core design is built upon two key components: a high-level **Domain-Specific Language (DSL)** for defining objects, and a sophisticated **Runtime Architecture** that provides the services to manage them.

### 2.3.1 The Declarative Object Language (DSL)

Since C lacks a native 'class' keyword, we created a declarative language using the C preprocessor. This DSL allows a developer to define object classes in a way that is clear, structured, and reminiscent of modern object-oriented languages, while the macros themselves expand into the necessary C boilerplate code. This approach provides a high level of abstraction without sacrificing the performance of C. The core of the DSL is exposed through the framework's public API, 'object.h'.

```

1 /**
2 * @file object.h
3 * @brief Public API for the
4 * AuctaSapience Declarative Object
5 * Runtime.
6 */
7 #ifndef AGENK_OBJECT_H
8 #define AGENK_OBJECT_H
9
10 /**
11 * @file object.h
12 * @brief Public API and Domain-Specific
13 * Language for the AuctaSapience
14 * Object Framework.
15 *
16 * @version 1.0.0
17 * @author Ankush Yadav, Ankit Yadav -
18 * AuctaSapience
19 *
20 * @section description_main Main
21 * Description
22 * This header file is the central
23 * nervous system of the AGENK Object
24 * Framework.

```

- \* It provides a complete, self-contained system for creating and managing objects in
- \* a pure C99 environment. It achieves object-oriented capabilities like inheritance
- \* (via composition and VTables), polymorphism, and managed lifecycles without
- \* relying on C++ or any non-standard compiler extensions.
- \*
- \* The file is divided into three major sections:
  - \* 1. **\*\*Public API:\*\*** A set of functions (`CLASS\_INIT`, `CREATE\_GROUP`) and macros (`NEW\_ROOT`, `CALL`, `GET`, `DESTROY`) that form the public-facing interface
  - \* for interacting with the framework.
  - \* 2. **\*\*Domain-Specific Language (DSL) :\*\*** A collection of powerful preprocessor macros (`OBJECT\_START`, `METHOD`, `CONSTRUCTOR`, etc.) that allow developers to declare new "classes" in a structured, readable way. This DSL abstracts away the complex C boilerplate required for manual VTable management and data structure setup.
  - \* 3. **\*\*Internal Helper Prototypes:\*\*** Forward declarations for the private backend functions that power the framework (e.g., registry and database interactions).
- \*
- \* @section design\_philosophy Design Philosophy
  - \* - **\*\*Abstraction & Specialization:\*\*** The framework is built on the idea that a generic `Object` (the abstraction) can be specialized into concrete types (like `Task` or `FileParser`). The DSL is the mechanism for this specialization.
  - \* - **\*\*Polymorphism in C:\*\*** The core of the object-oriented design is the Virtual Table (VTable). Every object contains a pointer to a table of its methods.
  - \* This allows generic code to operate on an `Object\*` and call its methods
  - \* without knowing the object's concrete type at compile time. This is achieved
  - \* safely through carefully designed macros and struct inheritance.
  - **\*\*Strict C99 Compliance:\*\*** The framework avoids all non-standard compiler

```

44 * extensions (like `typeof`) to
45 * ensure maximum portability. It can
46 * be compiled
47 * with any standard C99 compiler,
48 * from GCC to TinyCC.
49 * - **Managed Lifecycle:** Memory
50 * management in large C projects is
51 * notoriously
52 * difficult. The framework tackles
53 * this head-on with a group-based
54 * lifecycle
55 * system. Objects are created
56 * within a "group," and a single call
57 * to
58 * `DESTROY_BY_GROUP` can reliably
59 * clean up all memory associated with
60 * a specific
61 * task, preventing leaks.
62 * - **Safety and Robustness:** Macros
63 * are designed to be "NULL-safe,"
64 * meaning they
65 * will not crash if passed a `NULL`
66 * object pointer. This reduces the
67 * amount of
68 * defensive boilerplate code
69 * developers need to write.
70 */
71
72 #include <stdio.h>
73 #include <stdlib.h>
74 #include <string.h>
75 #include <stdbool.h>
76 #include <assert.h>
77 #include <stdint.h>
78
79 // Forward declare the generic Object
80 // handle.
81 typedef struct AgenkObject Object;
82
83 // --- Core VTable Definition ---
84 /**
85 * @struct _BaseVTable
86 * @brief The foundational structure for
87 * all virtual tables.
88 * @details Every VTable for every class
89 * created with this framework *must*
90 * begin
91 * with this exact structure. This is
92 * the cornerstone of polymorphism in
93 * this system.
94 * It allows any `Object*` to be safely
95 * cast to a handle with a `_BaseVTable*`,
96 * guaranteeing that we can always call
97 * these fundamental methods (like `destroy` or
98 * `get_id`) regardless of the object's
99 * specific type.
100 */
101 typedef struct _BaseVTable {
102     void (*destroy)(Object*);
103     unsigned long (*get_id)(const void*)
104     ;
105     unsigned long (*get_group_id)(const
106         void* );
107     unsigned long (*get_parent_id)(const
108         void* );
109     void (* private destroy)(void*); //

```

```

Internal hook for the class's
own destructor logic.
} _BaseVTable;
// --- The Generic Object Handle ---
/** @struct AgenkObject
 * @brief The public-facing, generic
 * handle for any object in the
 * framework.
 * @details This is the "void pointer"
 * of the object world. It's an opaque
 * handle
 * that contains the two essential
 * pieces of information for
 * polymorphism:
 * 1. `vtable`: A pointer to the
 * object's method table. Crucially,
 * it's typed
 * as a `_BaseVTable*` to allow
 * safe access to the common methods.
 * 2. `data`: A void pointer to the
 * object's unique, private data
 * members.
 */
struct AgenkObject {
    const _BaseVTable* vtable;
    void* data;
};

/* ===== */
/* == PUBLIC FRAMEWORK API == */
/* ===== */

/** @brief Initializes the entire object
 * framework runtime. Must be called
 * once at application start. */
void CLASS_INIT(void);
/** @brief Shuts down the framework and
 * cleans up all global resources. Must
 * be called once at application end.
 */
void CLASS_SHUTDOWN(void);
/** @brief Creates a new lifecycle group
 * for objects, returning a unique ID
 * for that group. */
unsigned long CREATE_GROUP(const char*
    description);
/** @brief Destroys all objects
 * associated with a specific group ID,
 * preventing memory leaks. */
void DESTROY_BY_GROUP(unsigned long
    group_id);

/* ===== */
/* == PUBLIC OBJECT LIFECYCLE API == */
/* ===== */

/** @brief Creates a new "root" object (one
 * without a parent) in a specified
 * group. */
#define NEW_ROOT(ClassName, group_id,
    ...) \
    ClassName##_create(group_id, 0, ##_
        VA_ARGS__)
/** @brief Creates a "child" object,
 * automatically inheriting its parent'
 * s group and setting its parent ID.
 */

/*
#define NEW_CHILD(ClassName, parent_obj,
    ...) \
    ClassName##_create( \
        parent_obj ? GET(parent_obj,
            group_id, unsigned long) : \
            0, \
        parent_obj ? GET(parent_obj, id
            , unsigned long) : 0), \
        ##_VA_ARGS_
)

/** @brief Destroys a single object
 * instance. Safe to call on NULL. */
#define DESTROY(obj) \
    do { \
        if (obj) { \
            ((const Object*)obj)->vtable
                ->destroy((Object*)obj); \
        } \
    } while (0)
/** @brief (Internal) Destroys an object
 * using its unique ID. Not intended
 * for public use. */
void _class_destroy_by_id(unsigned long
    object_id);

/* ===== */
/* = PUBLIC OBJECT INTERACTION API = */
/* ===== */

/** @brief (C99 Compliant) Calls a
 * method on an object that does not
 * return a value. */
#define CALL(ClassName, obj, method,
    ...) \
    do { \
        if (obj) { \
            ((const ClassName##_VTable*)
                ((const Object*)obj)->
                    vtable)->method((void*)
                        obj, ##_VA_ARGS_); \
        } \
    } while (0)
/** @brief (C99 Compliant) Calls a
 * method on an object that returns a
 * value. */
#define CALL_R(ClassName, obj, method,
    ret_type, ...) \
    (obj ? \
        ((const ClassName##_VTable*)(((
            const Object*)obj)->vtable)
            ->method((void*)obj, ##_VA_
                ARGS_) \
            : (ret_type)0)

/** @brief Gets the value of a built-in
 * property (id, group_id, parent_id). */
#define GET(obj, property, type) \
    (obj && ((const Object*)obj)->vtable
        ? \
            ((const Object*)obj)->vtable->
                get_##property((const void*)
                    obj) \
            : (type)0)

```

```

166 /* ===== */
167 /* === DSL FOR OBJECT CREATION === */
168 /* ===== */
169
170 /** @brief Begins the definition of a
171 new class. */
172 #define OBJECT_START(ClassName) \
173     typedef struct ClassName##_Data \
174         ClassName##_Data; \
175     typedef struct ClassName##_VTable \
176         ClassName##_VTable; \
177     typedef struct ClassName { \
178         const ClassName##_VTable* vtable \
179             ; \
180         ClassName##_Data* data; \
181     } ClassName; \
182     struct ClassName##_Data { \
183         unsigned long id; \
184         unsigned long group_id; \
185         unsigned long parent_id; \
186
187     /** @brief Separates the data member
188 section from the method definition
189 section within a class definition.
190 */
191 #define VTABLE(ClassName) \
192     }; \
193     struct ClassName##_VTable { \
194         _BaseVTable base; \
195
196     /** @brief Declares a new method (
197 function pointer) within the class's
198 VTable. */
199 #define METHOD(ret_type, name, ...) \
200     ret_type (*name)(void* self, ## \
201         _VA_ARGS__);
202
203     /** @brief Concludes the definition of a
204 class's data and methods. */
205 #define OBJECT_END(ClassName, ...) \
206     }; \
207     Object* ClassName##_create(unsigned \
208         long group_id, unsigned long \
209         parent_id, ##_VA_ARGS__);
210
211     /** @brief Begins the implementation
212 block for a class's constructor
213 logic. */
214 #define CONSTRUCTOR(ClassName, ...) \
215     static void _##ClassName##_private_ \
216         destroy(void* self_void); \
217     static void _##ClassName##_destroy( \
218         Object* self_void); \
219     static unsigned long _##ClassName##_ \
220         get_id(const void* s); \
221     static unsigned long _##ClassName##_ \
222         get_group_id(const void* s); \
223     static unsigned long _##ClassName##_ \
224         get_parent_id(const void* s); \
225     Object* ClassName##_create(unsigned \
226         long group_id, unsigned long \
227         parent_id, ##_VA_ARGS_) { \
228         static unsigned long s_type_id = \
229             0; \
230         if (s_type_id == 0) { s_type_id = \
231             _CLASS_HASH(#ClassName); } \
232         ClassName* self_obj = (ClassName*) \
233             malloc(sizeof(ClassName));
234
235         if (!self_obj) { return NULL; } \
236         self_obj->data = (ClassName##_ \
237             Data*)calloc(1, sizeof( \
238                 ClassName##_Data)); \
239         if (!self_obj->data) { free(self_ \
240             _obj); return NULL; } \
241         extern const ClassName##_VTable \
242             VTABLE_##ClassName; \
243         self_obj->vtable = (const \
244             ClassName##_VTable*)&VTABLE_ \
245                 ##ClassName; \
246         self_obj->data->group_id = group_ \
247             id; \
248         self_obj->data->parent_id = \
249             parent_id; \
250         self_obj->data->id = class_ \
251             registry_register((Object*) \
252                 self_obj); \
253         if (self_obj->data->id == 0) { \
254             free(self_obj->data); free( \
255                 self_obj); return NULL; } \
256         _class_db_log_creation(self_obj \
257             ->data->id, group_id, parent_ \
258             id, s_type_id); \
259         { ClassName##_Data* self = self_ \
260             obj->data;
261
262         /** @brief Begins the implementation
263 block for a class's destructor logic
264 . */
265 #define DESTRUCTOR(ClassName) \
266     } \
267     return (Object*)self_obj; \
268 } \
269     static void _##ClassName##_destroy( \
270         Object* self_void) { \
271         ClassName* self = (ClassName*) \
272             self_void; \
273         if (!self) return; \
274         if (self->vtable->base._private_ \
275             destroy) { \
276             self->vtable->base._private_ \
277                 destroy(self); \
278         } \
279         _class_db_log_destruction(self-> \
280             data->id); \
281         _class_registry_unregister(self \
282             ->data->id); \
283         free(self->data); \
284         free(self); \
285     } \
286     static unsigned long _##ClassName##_ \
287         get_id(const void* s) { return \
288         ((const ClassName*)s)->data->id; \
289     } \
290     static unsigned long _##ClassName##_ \
291         get_group_id(const void* s) { \
292         return ((const ClassName*)s)-> \
293             data->group_id; } \
294     static unsigned long _##ClassName##_ \
295         get_parent_id(const void* s) { \
296         return ((const ClassName*)s)-> \
297             data->parent_id; } \
298     static void _##ClassName##_private_ \
299         destroy(void* self_void) { \
300         ClassName* self_obj = (ClassName* \
301             *)self_void; \
302         ClassName##_Data* self = self_ \
303             obj->data;

```

```

244     obj->data;
245 
246 #define METHOD_IMPL(ClassName, ret_type,
247   name, ...) \
248   static ret_type _##ClassName## ##
249   name(void* self_void, ##_VA_
250   ARGS__); \
251   static ret_type _##ClassName## ##
252   name(void* self_void, ##_VA_
253   ARGS_) { \
254     ClassName* self_obj = (ClassName
255       *)self_void; \
256     ClassName##_Data* self = self_
257     obj->data;
258 
259   /** @brief Concludes a method
260    implementation block. */
261 #define METHOD_IMPL_END }
262 
263 /**
264  * @brief Begins the initialization of
265  * the global static VTable for a class
266  * .
267  */
268 #define VTABLE_IMPL(ClassName, ...) \
269   const ClassName##_VTable VTABLE_##
270   ClassName = { \
271     .base = { \
272       .destroy = _##ClassName##_
273       destroy, \
274       .get_id = _##ClassName##_get
275       _id, \
276       .get_group_id = _##ClassName
277       ##_get_group_id, \
278       .get_parent_id = _##
279       ClassName##_get_parent_
280       id, \
281       ._private_destroy = _##
282       ClassName##_private_
283       destroy \
284     }, \
285     ##_VA_ARGS_ \
286   };
287 
288 /* ===== */
289 /* ===== INTERNAL HELPERS ===== */
290 /* ===== */
291 
292 // Prototypes for the backend functions
293 // hidden from the end-user.
294 unsigned long _class_registry_register(
295   Object* obj);
296 void _class_registry_unregister(unsigned
297   long id);
298 bool _class_db_log_creation(unsigned
299   long object_id, unsigned long group_
300   id, unsigned long parent_id,
301   unsigned long class_hash);
302 void _class_db_log_destruction(unsigned
303   long object_id);
304 unsigned long _class_db_create_group(
305   const char* description);
306 void _class_db_find_by_group(unsigned
307   long group_id, unsigned long** out_
308   ids, size_t* out_count);
309 void _class_db_destroy_group(unsigned
310   long group_id);
311 unsigned long _CLASS_HASH(const char*
312   str);
313 
```

```

281 // Assertion macro for debugging builds.
282 // Does nothing in release builds.
283 #ifndef NDEBUG
284   #define _CLASS_ASSERT_TYPE(obj,
285     member) assert(obj && ((const
286     Object*)obj)->vtable && "Object"
287     is NULL or vtable is NULL.)
288 #else
289   #define _CLASS_ASSERT_TYPE(obj,
290     member) ((void)0)
291 #endif
292 
```

Listing 1: The framework's public API and DSL definition (object.h).

**The In-Memory Registry: A High-Concurrency Object Cache** At the core of the runtime is a high-performance, in-memory object registry. Its function is to act as a volatile, high-speed cache that maintains the crucial mapping from a persistent, unique ‘ObjectID’ to a live, in-memory ‘Object\*’ pointer. This registry is the engine that powers all real-time object interactions; without it, every ‘CALL’ or ‘GET’ operation would require a prohibitively slow disk lookup. The entire design of this component is therefore relentlessly optimized for concurrent performance and thread safety.

**Architectural Insight — Why a Separate Registry?:** A key architectural decision was to separate this in-memory registry from the persistent database. The database is the “source of truth” about what objects **should** exist, while the in-memory registry is the “working set” of objects that are **currently live** in a specific process. This separation is fundamental. The database provides durability and a global view, while the in-memory hash map provides the microsecond-level lookup speeds required for high-performance computation.

**The Algorithm — Fine-Grained Locking:** To support extreme concurrency, a simple global lock protecting the entire hash map was deemed insufficient. Such a design would serialize all object creations and destructions, creating a major bottleneck in a multi-threaded agent. Instead, we implemented a **fine-grained, per-bucket locking** strategy. The hash map is an array of buckets, and we maintain a parallel array of lightweight mutexes, one for each bucket. When a thread needs to modify the linked list within a bucket (to add or remove an object), it locks **only that specific bucket’s mutex**. This is a powerful concurrency

pattern because it allows threads operating on different buckets—which is the vast majority of cases with a good hash function—to execute in parallel without any contention. The performance scales almost linearly with the number of CPU cores, a critical feature for a modern AI agent.

**The Algorithm — Lock-Free Reads:** For even greater performance, read operations ('find\_by\_id') are designed to be **lock-free**. On all modern architectures, a pointer read/write is an atomic operation. This allows threads to traverse the linked lists in the hash map without acquiring any locks. While this introduces the theoretical possibility of a "stale read" (reading a pointer just as another thread is freeing it), the consequence is a harmless lookup miss, which is an acceptable and standard trade-off for the immense performance gain in read-heavy workloads. Writes, however, remain fully protected by the per-bucket locks to guarantee data integrity.

**The Algorithm — Dedicated Counter Lock:** A subtle but important detail is the handling of the global 'ObjectID' counter. If this counter were protected by the same locks as the hash map buckets, it would re-introduce a point of global contention. To prevent this, the counter is protected by its own dedicated, high-speed mutex. This allows one thread to acquire a new unique ID while other threads are simultaneously inserting previously-acquired objects into the main hash table, further maximizing parallelism.

**Implementation Insights — Portability:** Since the C99 standard lacks a native threading model, the locking primitives are implemented via a thin abstraction layer. Using preprocessor directives, the framework selects the optimal native implementation at compile time: 'pthreads' on POSIX-compliant systems (Linux, macOS) and 'CRITICAL\_SECTION' on Windows. This provides the highest performance on each platform without sacrificing the portability of the core algorithm's logic. This registry, therefore, represents a synthesis of high-performance concurrent algorithms and pragmatic, portable engineering, creating a foundation that is both extremely fast and robust enough to manage the complex, dynamic state of an intelligent agent.

```

1 /**
2  * @file object_registry.c
3  * @brief Portable, thread-safe, high-
4  *        performance implementation of the
5  *        Object Registry.
6 */

```

```

5  * @version 1.0.0
6  * @author Ankush Yadav, Ankit Yadav,
7  *        AuctaSapience
8  *
9  * @section description_main Main
10 * Description
11 * This file implements the central in-
12 * memory object registry. Its sole
13 * purpose is
14 * to maintain a high-speed mapping
15 * between a unique `ObjectID` (an
16 * unsigned long)
17 * and a live `Object*` pointer in
18 * memory. This provides O(1) average
19 * time
20 * complexity for all critical
21 * operations: registering a new
22 * object, unregistering
23 * a destroyed object, and finding an
24 * object by its ID.
25 *
26 * @section design_philosophy Design
27 * Philosophy
28 * - **Performance:** Speed is
29 * paramount. The registry uses a
30 * classic chained
31 * hash table for O(1) average
32 * lookups. Reads (`_class_registry_
33 * find_by_id`)
34 * are designed to be lock-free,
35 * providing maximum concurrency for
36 * the most
37 * common operation.
38 * - **Thread Safety & Concurrency:** The
39 * registry is designed to be used
40 * heavily by multiple threads. It
41 * employs a fine-grained, per-bucket
42 * locking
43 * strategy. Instead of one global
44 * lock creating a bottleneck, we have
45 * an
46 * array of locks. A thread only
47 * locks the specific hash table
48 * bucket it
49 * needs to modify. This allows
50 * threads operating on different
51 * buckets to
52 * execute in parallel, dramatically
53 * increasing throughput in high-
54 * contention
55 * scenarios. This is a standard
56 * industry technique for concurrent
57 * hash maps.
58 * - **Portability:** C99 has no
59 * native thread support. This file
60 * implements a
61 * thin "Threading Abstraction Layer"
62 * (TAL) over platform-specific
63 * mutexes
64 * (pthreads for POSIX, Critical
65 * Sections for Windows). This makes
66 * the core
67 * logic OS-agnostic and achieves
68 * maximum portability without
69 * sacrificing
70 * thread safety.
71 */

```

```

34 #include "object.h"
35 #include <stdlib.h>

```



```

132     fprintf(stderr, "CLASS_FATAL:\n"
133             "Failed to allocate memory\n"
134             "for object registry. Cannot\n"
135             "continue.\n");
136         exit(EXIT_FAILURE); // A failure
137         here is unrecoverable.
138     }
139
140     // Initialize every single mutex in
141     // the lock array for fine-grained
142     // locking.
143     for (size_t i = 0; i < g_registry.
144         capacity; ++i) {
145         LOCK_INIT(&g_registry.locks[i]);
146     }
147     LOCK_INIT(&g_id_lock);
148
149     g_is_initialized = true;
150     UNLOCK(&g_init_lock);
151
152 /**
153 * @brief Shuts down and deallocates the
154 * global object registry.
155 * @details Performs a final leak check
156 * by reporting if the object count is
157 * non-zero.
158 */
159 void _class_registry_shutdown_internal(
160     void) {
161     LOCK(&g_init_lock);
162     if (!g_is_initialized) {
163         UNLOCK(&g_init_lock);
164         return;
165     }
166
167     for (size_t i = 0; i < g_registry.
168         capacity; ++i) {
169         LOCK_DESTROY(&g_registry.locks[i]);
170     }
171     LOCK_DESTROY(&g_id_lock);
172
173     if (g_registry.count > 0) {
174         fprintf(stderr, "CLASS_WARNING:\n"
175                 "%zu object(s) were not\n"
176                 "destroyed. Memory leak\n"
177                 "detected.\n", g_registry.
178                 count);
179     }
180
181     // Since this is final shutdown, we
182     // don't need to lock each bucket
183     // to free nodes.
184     for (size_t i = 0; i < g_registry.
185         capacity; ++i) {
186         RegistryNode* current = g_
187             registry.table[i];
188         while(current) {
189             RegistryNode* next = current
190                 ->next;
191             free(current);
192             current = next;
193         }
194     }
195     free(g_registry.table);
196     free(g_registry.locks);
197
198     // Reset state to prevent use-after-
199
200     // free if framework is re-
201     // initialized.
202     g_registry.table = NULL;
203     g_registry.locks = NULL;
204     g_registry.capacity = 0;
205     g_registry.count = 0;
206     g_is_initialized = false;
207
208     UNLOCK(&g_init_lock);
209     LOCK_DESTROY(&g_init_lock);
210
211 /**
212 * ===== CORE REGISTRY OPERATIONS =====
213 */
214
215 /**
216 * @brief Registers a newly created
217 * object, assigning it a unique ID.
218 * @details This is the single source of
219 * truth for new ObjectIDs. It gets
220 * an ID,
221 * allocates a registry node, and
222 * inserts it into the correct hash
223 * table bucket
224 * under a fine-grained lock. This
225 * function is thread-safe.
226 * @param obj A pointer to the live
227 * object to register.
228 * @return The new unique ID for the
229 * object, or 0 on allocation failure.
230 */
231 unsigned long _class_registry_register(
232     Object* obj) {
233     if (!g_is_initialized) {
234         fprintf(stderr, "CLASS_FATAL:\n"
235                 "Attempted to register an\n"
236                 "object before CLASS_INIT()\n"
237                 "was called.\n");
238         return 0;
239     }
240
241     // Step 1: Get a unique ID. This is
242     // a very fast, locked operation.
243     LOCK(&g_id_lock);
244     const unsigned long id = g_next_
245         object_id++;
246     UNLOCK(&g_id_lock);
247
248     // Step 2: Prepare the new node.
249     // This happens lock-free.
250     RegistryNode* new_node = (
251         RegistryNode*)malloc(sizeof(
252             RegistryNode));
253     if (!new_node) {
254         fprintf(stderr, "CLASS_FATAL:\n"
255                 "Failed to allocate memory\n"
256                 "for registry node.\n");
257         // We should "give back" the ID
258         // here, but in a fatal error
259         // case, it's less critical.
260         return 0;
261     }
262     new_node->id = id;
263     new_node->object_ptr = obj;
264
265     // Step 3: Insert the node into the
266     // hash table with a fine-grained
267     // lock.

```

```

225 const unsigned long index = _pyc_
226     hash_id(id) % g_registry.
227     capacity;
228 LOCK(&g_registry.locks[index]); // Lock only the specific bucket we
229     need.
230 new_node->next = g_registry.table[
231     index];
232 g_registry.table[index] = new_node;
233 UNLOCK(&g_registry.locks[index]); // Unlock immediately.
234
235 // [FIX] Protect the global counter
236     with a global lock to prevent
237     race conditions.
238 // This ensures two threads
239     operating on different buckets
240     don't clobber the count.
241 LOCK(&g_id_lock);
242 g_registry.count++;
243 UNLOCK(&g_id_lock);
244
245 return id;
246 }
247
248 /**
249 * @brief Unregisters an object from the
250     registry, given its ID.
251 * @details This function is thread-safe
252     .
253 * @param id The unique ID of the object
254     to remove. Called by the
255     destructor.
256 */
257 void _class_registry_unregister(unsigned
258     long id) {
259     if (!g_is_initialized || id == 0)
260         return;
261
262     const unsigned long index = _pyc_
263         hash_id(id) % g_registry.
264         capacity;
265
266     LOCK(&g_registry.locks[index]);
267
268     RegistryNode* current = g_registry.
269         table[index];
270     RegistryNode* prev = NULL;
271     while (current != NULL) {
272         if (current->id == id) {
273             if (prev == NULL) { // Node
274                 is at the head of the
275                 chain.
276                 g_registry.table[index]
277                     = current->next;
278             } else {
279                 prev->next = current->
280                     next;
281             }
282             free(current);
283             // [FIX] Decrement the
284             // counter under a global
285             // lock to prevent a race
286             // condition.
287             LOCK(&g_id_lock);
288             g_registry.count--;
289             UNLOCK(&g_id_lock);
290             break;
291         }
292         prev = current;
293     }
294
295     current = current->next;
296 }
297
298 /**
299 * @brief Finds an object pointer in the
300     registry by its unique ID.
301 * @details This operation is designed
302     to be **lock-free** for maximum
303     read
304 * performance. This is safe because
305     pointer reads are atomic on modern
306     CPUs,
307 * and we assume that read-after-write
308     consistency is not strictly
309     required
310 * for most lookup operations. The worst
311     case is a stale read, which is an
312     acceptable performance trade-off in
313     this context.
314 * @param id The ID of the object to
315     find.
316 * @return A pointer to the live Object,
317     or NULL if not found.
318 */
319 void* _class_registry_find_by_id(
320     unsigned long id){
321     if (!g_is_initialized || id == 0)
322         return NULL;
323
324     const unsigned long index = _pyc_
325         hash_id(id) % g_registry.
326         capacity;
327
328     // No lock is taken for reads,
329     // maximizing concurrency.
330     RegistryNode* current = g_registry.
331         table[index];
332     while (current != NULL) {
333         if (current->id == id) {
334             return current->object_ptr;
335         }
336         current = current->next;
337     }
338     return NULL;
339 }
```

Listing 2: The thread-safe, in-memory object registry (`src/objects/core/object_registry.c`).

**The Persistence Layer** To ensure robustness and provide deep introspection capabilities, the framework's architecture is built upon a persistent, transactional log of object metadata. This is a foundational design choice that moves beyond simple in-memory management to create a resilient and auditable system. At first glance, integrating a database might seem like a performance bottleneck; however, by carefully selecting the technology (LMDB) and strictly defining its role, we gain immense benefits in stability and diagnostics with negligible overhead.

The persistence layer's primary role is to act as

the agent's "black box flight recorder." It **does not store the live C objects themselves**—a crucial distinction, as memory pointers are volatile and meaningless outside of a running process. Instead, it transactionally records the **story** of each object's life: its creation and its eventual destruction. This metadata, stored in a high-performance LMDB (Lightning Memory-Mapped Database) key-value store on disk, provides a durable and consistent source of truth about the system's state over time.

**The Rationale for LMDB:** Our choice of LMDB was deliberate and critical for meeting the project's performance and portability goals. Unlike traditional databases, LMDB is an embedded, memory-mapped key-value store written in highly portable C. Its key advantages for this framework are:

- **Extreme Read Performance:** Because it is memory-mapped, reading data does not involve extra copies between kernel and user space. This makes introspection operations, like querying all objects in a group, blisteringly fast.
- **Transactional Integrity (ACID):** LMDB guarantees that all write operations are atomic, consistent, isolated, and durable. This is non-negotiable for system stability. An operation either completes fully or it has no effect at all, even in the event of an application crash. This prevents database corruption.
- **Portability and Simplicity:** As a small set of '.c' and '.h' files, LMDB can be easily "vendored"—included directly in our project's source code. This eliminates external dependencies and ensures the framework can be compiled easily with any standard C99 compiler, like TinyCC, on any platform.
- **Concurrency:** LMDB is designed for high concurrency, allowing many threads to read from the database simultaneously without locks, while safely serializing the single writer, which perfectly matches our agent's expected workload.

**The Data Model and the Three IDs:** The database records a small, efficient metadata struct for every object, indexed by its unique 'ObjectID'. This struct contains the three critical identifiers that define an object's context:

- The 'ObjectID': The object's absolute, unique identity. It serves as the primary key in the database.
- The 'GroupID': The "task" or "session" identifier. This is our primary tool for lifecycle management. The ability to query the database for all objects where 'group\_id == X' is what makes the powerful 'DESTROY\_BY\_GROUP' function possible.
- The 'ParentID': The 'ObjectID' of the object that created this one. This creates a causal chain, allowing a developer or a diagnostic tool to reconstruct the entire execution tree of a task ("this 'FileParser' object created three 'Line' objects, which in turn created...") from the database log alone.

**The Critical Lesson: Transactional Integrity in Practice:** The decision to enforce strict transactional writes was a critical lesson learned during development. An early, non-transactional prototype revealed a fatal flaw: if the application crashed **after** an object was created in memory but **before** its metadata could be written to disk, a "ghost object" was created. The in-memory registry would leak, but the database—our source of truth for cleanup—would be unaware of the object's existence, rendering the leak undetectable by 'DESTROY\_BY\_GROUP'. By wrapping every creation and destruction log in an LMDB transaction ('mdb\_txn\_begin'...'mdb\_txn\_commit'), we guarantee this cannot happen. If a crash occurs mid-operation, the transaction is automatically aborted, and the database remains in a perfectly consistent state.

This persistence layer, therefore, is not merely a logging feature. It is the architectural cornerstone that enables robust group-based lifecycle management, provides a complete audit trail for debugging complex agent behavior, and opens the door for future capabilities like crash recovery and inter-process introspection.

```

1 /**
2 * @file object_db.c
3 * @brief Private database layer for
4 * object metadata persistence using
5 * LMDB.
6 *
7 * @version 1.0.0
8 * @author Ankush Yadav, Ankit Yadav,
9 * AuctaSapience
10 */

```

```

9 #include "object.h"
10 #include <lmdb.h>
11 #include <stdlib.h>
12 #include <stdio.h>
13 #include <time.h>
14
15 #ifdef _WIN32
16 #include <windows.h>
17 #include <shlobj.h>
18 #else
19 #include <sys/stat.h>
20 #include <sys/types.h>
21 #include <unistd.h>
22 #include <pwd.h>
23 #endif
24
25 /* ===== */
26 /* === INTERNAL DATA STRUCTURES === */
27 /* ===== */
28
29 typedef struct {
30     unsigned long object_id;
31     unsigned long group_id;
32     unsigned long parent_id;
33     unsigned long class_type_hash;
34     time_t creation_timestamp;
35 } ObjectMetadata;
36
37 typedef struct {
38     unsigned long group_id;
39     char description[256];
40     time_t creation_timestamp;
41 } GroupMetadata;
42
43 /* ===== */
44 /* === STATIC GLOBALS & HELPERS === */
45 /* ===== */
46
47 static struct {
48     MDB_env *env;
49     MDB_dbi objects_db;
50     MDB_dbi groups_db;
51     MDB_dbi counters_db;
52 } g_db;
53
54 /**
55 * @brief A suite of robust error
56 * handling macros for LMDB operations
57 *
58 * @details This variadic macro handles
59 * both void and non-void functions.
60 * - MDB_CHECK(op, txn) -> used
61 * in a void function, expands to `return;`
62 * - MDB_CHECK(op, txn, retval)
63 * -> used in a non-void function,
64 * expands to `return retval;`
65 */
66 #define MDB_CHECK(op, txn, ...) \
67 do { \
68     int rc = (op); \
69     if (rc != MDB_SUCCESS) { \
70         fprintf(stderr, "LMDB_ERROR: \
71             %s:%d %s - %s\n", \
72             FILE_, LINE_, #op, \
73             mdb_strerror(rc)); \
74         if (txn) mdb_txn_abort(txn); \
75         return __VA_ARGS__; \
76     } \
77 } while (0)
78
79 // --- Static helper function
80 // declarations ---
81 static bool _get_app_data_path(char*
82     buffer, size_t len);
83 static void _ensure_dir_exists(const
84     char* path);
85 static unsigned long _db_get_next_id(MDB
86     _txn* txn, const char* counter_key);
87
88 /* ===== */
89 /* == INTERNAL LIFECYCLE FUNCTIONS == */
90 /* ===== */
91
92 void _class_db_init_internal(void) {
93     char db_path[1024];
94     if (!_get_app_data_path(db_path,
95         sizeof(db_path))) {
96         fprintf(stderr, "CLASS_DB_FATAL:
97             Could not determine
98             application data directory.\n");
99         exit(EXIT_FAILURE);
100    }
101    _ensure_dir_exists(db_path);
102
103    MDB_env *env;
104    MDB_CHECK(mdb_env_create(&env), NULL
105        );
106    MDB_CHECK(mdb_env_set_mapsize(env,
107        1024L * 1024L * 1024L), NULL);
108    MDB_CHECK(mdb_env_set_max dbs(env, 4
109        , NULL);
110
111    int rc = mdb_env_open(env, db_path,
112        MDB_WRITEMAP | MDB_NOSYNC, 0664
113        );
114    if (rc != MDB_SUCCESS) {
115        fprintf(stderr, "CLASS_DB_FATAL:
116            Could not open LMDB
117            environment at %s: %s\n", db
118            _path, mdb_strerror(rc));
119        mdb_env_close(env);
120        exit(EXIT_FAILURE);
121    }
122    g_db.env = env;
123
124    MDB_txn *txn;
125    MDB_CHECK(mdb_txn_begin(g_db.env,
126        NULL, 0, &txn), NULL);
127    MDB_CHECK(mdb_dbi_open(txn, "objects"
128        , MDB_CREATE | MDB_INTEGERKEY,
129        &g_db.objects_db), txn);
130    MDB_CHECK(mdb_dbi_open(txn, "groups"
131        , MDB_CREATE | MDB_INTEGERKEY, &
132        g_db.groups_db), txn);
133    MDB_CHECK(mdb_dbi_open(txn, "counters"
134        , MDB_CREATE, &g_db.
135        counters_db), txn);
136
137    // [FIX] Pass the transaction handle
138    // 'txn' to the macro. If commit
139    // fails, the transaction
140    // will now be properly aborted,
141    // preventing database corruption.
142    MDB_CHECK(mdb_txn_commit(txn), txn);
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167

```

```

112 void _class_db_shutdown_internal(void) {
113     if(g_db.env) {
114         mdb_env_close(g_db.env);
115         g_db.env = NULL;
116     }
117 }
118 }
119
120 /* ===== */
121 /* == PUBLIC API IMPLEMENTATIONS == */
122 /* ===== */
123
124 unsigned long _class_db_create_group(
125     const char* description) {
126     MDB_txn *txn;
127     MDB_val key, data;
128     GroupMetadata meta = {0};
129
130     MDB_CHECK(mdb_txn_begin(g_db.env,
131                             NULL, 0, &txn), NULL, 0);
132
133     meta.group_id = _db_get_next_id(txn,
134                                     "next_group_id");
135     if (meta.group_id == 0) { mdb_txn_
136         abort(txn); return 0; }
137
138     strncpy(meta.description,
139             description, sizeof(meta.
140             description) - 1);
141     meta.description[sizeof(meta.
142             description) - 1] = '\0';
143     meta.creation_timestamp = time(NULL)
144             ;
145
146     key.mv_size = sizeof(unsigned long);
147     key.mv_data = &meta.group_id;
148     data.mv_size = sizeof(GroupMetadata)
149             ;
150     data.mv_data = &meta;
151
152     MDB_CHECK(mdb_put(txn, g_db.groups_
153                     .dbi, &key, &data, 0), txn, 0);
154     // [FIX] Pass the transaction handle
155     // 'txn' to the macro for proper
156     // cleanup on failure.
157     MDB_CHECK(mdb_txn_commit(txn), txn,
158             0);
159
160     return meta.group_id;
161 }
162
163 bool _class_db_log_creation(unsigned
164     long object_id, unsigned long group_
165     id, unsigned long parent_id,
166     unsigned long class_hash) {
167     MDB_txn *txn;
168     MDB_val key, data;
169     ObjectMetadata meta = {object_id,
170                           group_id, parent_id, class_hash,
171                           time(NULL)};
172
173     key.mv_size = sizeof(unsigned long);
174     key.mv_data = &meta.object_id;
175     data.mv_size = sizeof(ObjectMetadata)
176             ;
177     data.mv_data = &meta;
178
179     MDB_CHECK(mdb_txn_begin(g_db.env,
180                             NULL, 0, &txn), NULL, false);
181
182     MDB_CHECK(mdb_put(txn, g_db.objects_
183                     .dbi, &key, &data, 0), txn, false
184 );
185     // [FIX] Pass the transaction handle
186     // 'txn' to the macro for proper
187     // cleanup on failure.
188     MDB_CHECK(mdb_txn_commit(txn), txn,
189             false);
190
191     return true;
192 }
193
194 void _class_db_log_destruction(unsigned
195     long object_id) {
196     if (object_id == 0) return;
197     MDB_txn *txn;
198     MDB_val key;
199     key.mv_size = sizeof(unsigned long);
200     key.mv_data = &object_id;
201
202     MDB_CHECK(mdb_txn_begin(g_db.env,
203                             NULL, 0, &txn), NULL);
204     int rc = mdb_del(txn, g_db.objects_
205                     .dbi, &key, NULL);
206     if (rc != MDB_SUCCESS && rc != MDB_
207         NOTFOUND) {
208         fprintf(stderr, "LMDB_WARNING: u
209                 Failed_to_delete_object_ID_u%
210                 lu:u%s\n", object_id, mdb_
211                 strerror(rc));
212         mdb_txn_abort(txn);
213     }
214     // [FIX] Pass the transaction handle
215     // 'txn' to the macro for proper
216     // cleanup on failure.
217     MDB_CHECK(mdb_txn_commit(txn), txn);
218
219 }
220
221 void _class_db_find_by_group(unsigned
222     long group_id, unsigned long** out_
223     ids, size_t* out_count) {
224     *out_ids = NULL;
225     *out_count = 0;
226     if (!g_db.env) return;
227
228     MDB_txn *txn;
229     MDB_cursor *cursor;
230     MDB_val key, data;
231     size_t capacity = 32;
232     unsigned long* ids = (unsigned long
233                           *)malloc(capacity * sizeof(
234                           unsigned long));
235     if (!ids) return;
236
237     MDB_CHECK(mdb_txn_begin(g_db.env,
238                             NULL, MDB_RDONLY, &txn), NULL);
239     MDB_CHECK(mdb_cursor_open(txn, g_db.
240                             objects_dbi, &cursor), txn);
241
242     while (mdb_cursor_get(cursor, &key,
243                           &data, MDB_NEXT) == MDB_SUCCESS)
244     {
245         ObjectMetadata* meta = (
246             ObjectMetadata*)data.mv_data
247             ;
248         if (meta->group_id == group_id)
249         {
250             if (*out_count >= capacity)
251             {

```

```

206         capacity *= 2;
207         unsigned long* new_ids =
208             (unsigned long*)
209                 realloc(ids,
210                     capacity * sizeof(
211                         unsigned long));
212             if (!new_ids) { free(ids);
213                 goto cleanup; }
214             ids = new_ids;
215         }
216         ids[(*out_count)++] = meta->
217             object_id;
218     }
219 }
220 cleanup:
221     mdb_cursor_close(cursor);
222     mdb_txn_abort(txn);
223     *out_ids = ids;
224 }
225 void _class_db_destroy_group(unsigned
226     long group_id) {
227     if (group_id == 0) return;
228     MDB_txn *txn;
229     MDB_val key;
230     key.mv_size = sizeof(unsigned long);
231     key.mv_data = &group_id;
232
233     MDB_CHECK(mdb_txn_begin(g_db.env,
234         NULL, 0, &txn), NULL);
235     int rc = mdb_del(txn, g_db.groups_
236         dbi, &key, NULL);
237     if (rc != MDB_SUCCESS && rc != MDB_
238         NOTFOUND) {
239         fprintf(stderr, "LMDBWARNING: Failed
240             to delete group ID %lu:\n", group_id, mdb_
241                 strerror(rc));
242         mdb_txn_abort(txn);
243         return;
244     }
245 // [FIX] Pass the transaction handle
246 // 'txn' to the macro for proper
247 // cleanup on failure.
248     MDB_CHECK(mdb_txn_commit(txn), txn);
249 }
250 /* ===== */
251 /* === STATIC HELPER DEFINITIONS === */
252 /* ===== */
253 static bool _get_app_data_path(char*
254     buffer, size_t len) {
255     #ifdef WIN32
256         if (!SHGetFolderPathA(NULL, CSIDL_
257             APPDATA, NULL, 0, buffer)
258             != S_OK) return false;
259         strncat(buffer, "\\AuctaSapience
260             _AGENK", len - strlen(buffer)
261             - 1);
262     #else
263         const char* home = getenv("HOME"
264             );
265         if (!home) home = getpwuid(
266             getuid())->pw_dir;
267         if (!home) return false;
268         #ifdef __APPLE__
269             sprintf(buffer, len, "%s/
270             Library/Application_
271             Support/AuctaSapience_
272             AGENK", home);
273         #else
274             snprintf(buffer, len, "%s/
275             local/share/
276             AuctaSapience_AGENK",
277             home);
278         #endif
279     #endif
280     return true;
281 }
282 static void ensure_dir_exists(const
283     char* path) {
284     #ifdef _WIN32
285         CreateDirectoryA(path, NULL);
286     #else
287         mkdir(path, 0755);
288     #endif
289 }
290 static unsigned long _db_get_next_id(MDB_
291     txn* txn, const char* counter_key)
292 {
293     unsigned long next_id = 1;
294     MDB_val key, data;
295     key.mv_size = strlen(counter_key);
296     key.mv_data = (void*)counter_key;
297
298     int rc = mdb_get(txn, g_db.counters_
299         dbi, &key, &data);
300     if (rc == MDB_SUCCESS) {
301         memcpy(&next_id, data.mv_data,
302             sizeof(unsigned long));
303     } else if (rc == MDB_NOTFOUND) {
304         fprintf(stderr, "LMDBERROR: Failed
305             to read counter '%s':
306             %s\n", counter_key, mdb_
307                 strerror(rc));
308         return 0;
309     }
310
311     unsigned long new_val = next_id + 1;
312     data.mv_size = sizeof(unsigned long)
313         ;
314     data.mv_data = &new_val;
315     if (mdb_put(txn, g_db.counters_dbi,
316         &key, &data, 0) != MDB_SUCCESS)
317     {
318         fprintf(stderr, "LMDBERROR: Failed
319             to update counter '%s'
320             : %s\n", counter_key, mdb_
321                 strerror(rc));
322         return 0;
323     }
324     return next_id;
325 }

```

Listing 3: The persistent database layer (src/objects/core/object\_db.c).

**The Core Orchestration Layer** The final component is the orchestration layer, which serves as the “controller” connecting the public API to the specialized backend services. Its primary responsibility is to translate high-level user inten-

tions into a correct sequence of operations across the registry and the database. For example, ‘DESTROY\_BY\_GROUP’ is implemented here, first querying the database to get a list of all object IDs in a group, and then iterating through that list, calling the internal ‘\_class\_destroy\_by\_id’ function for each one. This function, in turn, uses the registry to find the live object pointer and trigger its destruction. This clean separation of concerns is fundamental to the framework’s maintainability and robustness.

```
1 /**
2 * @file object_impl.c
3 * @brief Private core implementation of
4 *        the AuctaSapience Object Runtime.
5 *
6 * @version 1.0.0
7 * @author Ankush Yadav, Ankit Yadav,
8 *        AuctaSapience
9 *
10 * @section description_main Main
11 * Description
12 * This file serves as the central
13 * orchestration layer for the object
14 * framework.
15 * It contains the concrete
16 * implementations of the public API
17 * functions declared
18 * in `object.h` that require
19 * coordination between the different
20 * backend modules
21 * (the in-memory registry and the
22 * persistent database).
23 *
24 * @section design_philosophy Design
25 * Philosophy
26 * The core design principle here is a
27 * clear separation of concerns. This
28 * file
29 * should contain minimal complex logic
30 * itself. Its main purpose is to act
31 * as
32 * a "controller" that correctly calls
33 * the specialized "service" functions
34 * in `object_registry.c` (for high-
35 * speed, in-memory pointer operations
36 * ) and
37 * `object_db.c` (for transactional,
38 * persistent metadata storage).
39 *
40 * By maintaining this separation, the
41 * framework is easier to debug,
42 * maintain,
43 * and extend. For example, replacing
44 * the LMDB database with a different
45 * storage engine would only require
46 * changes to `object_db.c`, leaving
47 * this
48 * orchestration layer untouched.
49 */
50
51 #include "object.h" // The public API
52 header, provides type definitions
53 and prototypes.
54 #include <stdlib.h> // For NULL, free.
55 #include <stdint.h> // For portable
```

```
30      integer types.
31
32  /* ===== */
33  /* FORWARD DECLARATIONS OF PRIVATE APIs */
34  /* ===== */
35
36 // By forward-declaring the private
37 // functions from other core files, we
38 // avoid
39 // the need for a private "core.h"
40 // header. This keeps the dependency
41 // graph
42 // simple: all core implementation files
43 // only need to include the public
44 // `object.h`.
45 // These declarations form the contract
46 // between the framework's internal
47 // modules.
48
49 /* --- From object_registry.c --- */
50 void _class_registry_init_internal(void)
51 ;
52 void _class_registry_shutdown_internal(
53     void);
54 void* _class_registry_find_by_id(
55     unsigned long id);
56
57 /* --- From object_db.c --- */
58 void _class_db_init_internal(void);
59 void _class_db_shutdown_internal(void);
60
61 /* ===== */
62 /* ===== FRAMEWORK HELPERS ===== */
63 /* ===== */
64
65 // FNV-1a hash algorithm constants.
66 // Chosen for speed and good
67 // distribution.
68 #define FNV_OFFSET_BASIS 2166136261U
69 #define FNV_PRIME 16777619U
70
71 /**
72 * @brief Implements the _CLASS_HASH
73 * function declared in object.h.
74 * @details This function provides the
75 * implementation for class name
76 * hashing,
77 * which generates a unique type ID for
78 * each class at runtime. It is linked
79 * into the `libclass_framework.a`
80 * static library.
81 * @param str The string to hash (the
82 * class name).
83 * @return A 64-bit hash value.
84 */
85 unsigned long _CLASS_HASH(const char*
86     str) {
87     unsigned long hash = FNV_OFFSET_
88         BASIS;
89     if (!str) return hash;
90     while (*str) {
91         hash ^= (unsigned long)(*str++);
92         hash *= FNV_PRIME;
93     }
94     return hash;
95 }
96
97 /* ===== */
```

```

78 /* == FRAMEWORK STATE & LIFECYCLE == */
79 /* ===== */
80 // A static global flag to ensure the
81 // initialization logic runs only once.
82 // This makes CLASS_INIT idempotent (safe to call multiple times).
83 static bool g_is_initialized = false;
84 /**
85 * @brief Public API function to initialize the object runtime.
86 * @details Initializes subsystems in the correct order: persistent storage (DB) first, then the volatile cache (Registry).
87 */
88 void CLASS_INIT(void) {
89     if (g_is_initialized) {
90         return;
91     }
92     _class_db_init_internal();
93     _class_registry_init_internal();
94     g_is_initialized = true;
95 }
96 /**
97 * @brief Public API function to shut down the object runtime.
98 * @details Shuts down subsystems in the reverse order of initialization to ensure a clean teardown.
99 */
100 void CLASS_SHUTDOWN(void) {
101     if (!g_is_initialized) {
102         return;
103     }
104     _class_registry_shutdown_internal();
105     _class_db_shutdown_internal();
106     g_is_initialized = false;
107 }
108 /**
109 * @brief Public API function to create a new group.
110 * @details This is a thin wrapper that delegates the actual work to the database layer, which handles the creation transactionally.
111 */
112 unsigned long CREATE_GROUP(const char* description) {
113     return _class_db_create_group(description);
114 }
115 /**
116 * @brief Implements the DESTROY_BY_ID macro logic.
117 * @details This function bridges the gap between the "ID world" (which can be stored and passed around) and the "pointer world" (live objects in
118 * memory).
119 * It looks up the ID in the registry to get a live pointer, then calls the standard DESTROY macro on it.
120 * @param id The unique ID of the object to find and destroy.
121 */
122 void class_destroy_by_id(unsigned long id) {
123     // Find the live object pointer in the in-memory registry.
124     Object* obj_to_destroy = (Object*)_class_registry_find_by_id(id);
125     // If the object is live, destroy it using the public DESTROY macro. This
126     // is crucial because it guarantees the full, correct destruction cascade
127     // is triggered (calling the user's DESTRUCTOR, unregistering, etc.).
128     // If the object is not found (already destroyed), this does nothing.
129     if (obj_to_destroy) {
130         DESTROY(obj_to_destroy);
131     }
132 /**
133 * @brief Public API function to destroy all objects in a group.
134 * @details This is a high-level orchestration function.
135 * 1. It queries the persistent database (the "source of truth") for a list
136     of all object IDs belonging to the group.
137 * 2. It iterates through this list, destroying each object by its ID.
138 * 3. It frees the list of IDs.
139 * 4. Finally, it destroys the group metadata record itself from the database.
140 * @param group_id The ID of the group to destroy.
141 */
142 void DESTROY_BY_GROUP(unsigned long group_id) {
143     if (group_id == 0) return; // 0 is not a valid group ID.
144     unsigned long* ids_to_destroy = NULL;
145     size_t count = 0;
146     // Step 1: Query the database for all members of the group.
147     _class_db_find_by_group(group_id, &ids_to_destroy, &count);
148     if (count > 0 && ids_to_destroy) {
149         // Step 2: Iterate and destroy each object.
150         for (size_t i = 0; i < count; ++i) {
151             _class_destroy_by_id(ids_to_
152         }
153     }
154 }
```

```

    destroy[i]);
}
// Step 3: Free the memory
// allocated by the database
// function.
free(ids_to_destroy);
}

// Step 4: Destroy the group record
// itself.
_class_db_destroy_group(group_id);
}

```

Listing 4: The core orchestration layer (`src/objects/core/object_impl.c`).

### 2.3.2 Project Implementation and Validation

To move from an abstract design to a tangible, high-performance system, we implemented the **\*\*AGENK Object Framework\*\*** as a C project. This section details the project's structure, its build system configuration using CMake, and the rigorous testing methodology used to validate its correctness, robustness, and core features.

### 2.3.3 Project Structure and Build System

A well-organized directory structure is critical for managing complexity. We adopted a standard layout that separates the core framework source code from examples, tests, and third-party dependencies.

```

AGENK/
├── CMakeLists.txt
└── deps/
    └── lmdb/
        ├── lmdb.h
        ├── mdb.c
        ├── midl.c
        └── midl.h
└── src/
    └── objects/
        └── core/
            ├── object_db.c
            ├── object_impl.c
            └── object_registry.c
        └── examples/
            ├── basic_usage.c
            └── lifecycle_and_groups.c
        └── test/
            ├── test_concurrency.c
            ├── test_errors.c
            ├── test_hierarchy.c
            └── test_stress.c
        └── object.h
└── build/

```

### (Directory for generated build files)

The entire build process is managed by CMake, which provides a cross-platform, declarative way to define build targets. The ‘`CMakeLists.txt`’ file defines two primary components:

1. A static library, `class_framework`, which encapsulates the entire runtime system (registry, database, core implementation). This promotes modularity and code reuse.
2. A series of executables for our examples and test suites, each of which links against the core `class_framework` library.

The configuration enforces a strict C99 standard to ensure portability and prevent reliance on compiler-specific extensions, a key consideration for stable, long-term infrastructure.

```

1 # =====
2 # CMAKE PROJECT CONFIGURATION
3 # =====
4 # Set the minimum version of CMake
# required to build this project.
# This ensures that developers use a
# compatible build system version.
cmake_minimum_required(VERSION 3.14)
5
6 # Define the project name, version, and
# primary language.
# This information is used by IDEs and
# packaging tools.
7 project(AGENK VERSION 1.0.0 LANGUAGES C)
8
9
10 # =====
11 # COMPILER AND BUILD STANDARDS
12 # =====
13 # Enforce the C99 standard for the
# entire project. This guarantees that
# portable language features are used
# and prevents reliance on compiler-
# specific extensions. This is critical
# for portability to compilers like
# TCC.
14 set(CMAKE_C_STANDARD 99)
15 set(CMAKE_C_STANDARD_REQUIRED ON)
16 set(CMAKE_C_EXTENSIONS OFF) # Explicitly
# disable GNU extensions
17
18 # Set the default build type to "Debug"
# if none is specified by the user.
19 # Debug builds include debugging symbols
# (-g) and do not define NDEBUG,
# which enables our framework's runtime
# safety assertions.
20 # A user can override this with `cmake -
# -DCMAKE_BUILD_TYPE=Release ..`
21 if(NOT CMAKE_BUILD_TYPE)
22     set(CMAKE_BUILD_TYPE Debug)
23 endif()
24
25 # Add a message to inform the user of
# the build type.
26
27
28
29
30
31

```

```

32 message(STATUS "Build type set to: ${CMAKE_BUILD_TYPE}")
33
34 # =====
35 # DEPENDENCY MANAGEMENT (VENDORED LMDB)
36 # =====
37
38 # --- Define the LMDB library from its
#      source files ---
39 # Since the vendored LMDB source does
#      not have its own CMakeLists.txt, we
40 # define it as a library target here.
#      This tells CMake to compile these
41 # specific source files into a static
#      library named "lmbd".
42 add_library(lmbd STATIC
43   deps/lmbd/mdb.c
44   deps/lmbd/midl.c
45 )
46
47 # Tell any target that links against our
#      `lmbd` library where to find its
48 # public header files (lmbd.h and midi.h
#      ). We declare the `deps/lmbd`
49 # directory as a PUBLIC include
#      directory for the `lmbd` target.
50 target_include_directories(lmbd
51   PUBLIC
52     ${CMAKE_CURRENT_SOURCE_DIR}/deps
53     /lmbd
54 )
55
56 # =====
57 # CORE FRAMEWORK LIBRARY DEFINITION
58 # =====
59
60 # Define our core object system as a
#      static library named "class_
#      framework".
61 # A library is a bundle of pre-compiled
#      code that can be reused by multiple
62 # executables. This is a fundamental
#      principle of good software design.
63 # The paths are now corrected to match
#      your project's file structure.
64 add_library(class_framework STATIC
65   src/objects/core/object_impl.c
66   src/objects/core/object_registry.c
67   src/objects/core/object_db.c
68 )
69
70 # Tell any target that links against `class_
#      framework` where to find its
71 # public header files. The header `object.h` is in `src/objects` .
72 # We declare this PUBLIC so any
#      executable linking to our framework
#      also
73 # gets this include path automatically.
74 target_include_directories(class_
#      framework
75   PUBLIC
76     ${CMAKE_CURRENT_SOURCE_DIR}/src/
#      objects
77 )
78
79 # Link our framework against the LMDB
#      library that we just defined above.
# We declare this as a PRIVATE
80
81 dependency because the end-user's
# code does not
82 # need to know about LMDB; it's an
# internal implementation detail of
# our
83 target_link_libraries(class_framework
84   PRIVATE
85     lmbd
86 )
87
88 # LMDB requires the pthreads library for
#      its locking mechanisms on POSIX
#      systems.
89 # This finds the library and links it.
#      On non-POSIX systems like Windows,
90 # CMake handles this gracefully.
91 find_package(Threads REQUIRED)
92 target_link_libraries(class_framework
93   PRIVATE
94     Threads::Threads
95 )
96
97 # =====
98 # EXECUTABLE TARGETS (EXAMPLES & TESTS)
99 # =====
100
101 # The paths to all source files have
#      been corrected to match your
#      structure.
102
103 # --- Examples ---
104 add_executable(example_basic      src/
#      objects/examples/basic_usage.c)
105 add_executable(example.lifecycle src/
#      objects/examples/lifecycle_and_
#      groups.c)
106
107 # --- Tests ---
108 add_executable(test_stress        src/
#      objects/test/test_stress.c)
109 add_executable(test_errors        src/
#      objects/test/test_errors.c)
110 add_executable(test_hierarchy     src/
#      objects/test/test_hierarchy.c)
111 add_executable(test_concurrency   src/
#      objects/test/test_concurrency.c)
112
113 # --- Link all executables against our
#      framework ---
114 # This makes all the public functions
#      and macros from our object system
115 # available to our example and test
#      programs.
116 target_link_libraries(example_basic
117   PRIVATE class_framework)
118 target_link_libraries(example.lifecycle
119   PRIVATE class_framework)
120 target_link_libraries(test_stress
121   PRIVATE class_framework)
122 target_link_libraries(test_errors
123   PRIVATE class_framework)
124 target_link_libraries(test_hierarchy
125   PRIVATE class_framework)
126
127
128 # =====
129 # INSTALLATION AND PACKAGING
130

```

```

126 # (OPTIONAL BUT GOOD PRACTICE)
127 # =====
128 # These rules define how to "install"
129 # the project, which is useful for
130 # creating distributable packages or for
131 # using this library in other
132 # CMake projects.
133 # install(TARGETS class_framework
# DESTINATION lib)
# install(FILES src/objects/object.h
# DESTINATION include)

```

Listing 5: The root CMakeLists.txt file for the framework project.

### 2.3.4 Validating the Core Design: The Test Suite

A framework this foundational requires a comprehensive test suite to ensure its correctness and stability. We developed a series of unit and integration tests to validate each component of the runtime system. Two key tests are presented here: one for hierarchical integrity and another for error handling resilience.

The first test, ‘test\_hierarchy’, validates that the “Relationships (Location)” aspect of our object definition is correctly implemented. It confirms that a child object created via ‘NEW\_CHILD’ correctly inherits its parent’s ‘GroupID’ and has its ‘ParentID’ set to the parent’s unique ‘ObjectID’. This is fundamental to building structured networks of objects.

```

1 #include "object.h"
2 #include <stdio.h>
3 #include <stdbool.h>
4
5 static int g_tests_passed = 0;
6 #define ASSERT_TRUE(cond, msg) do { if (cond) { g_tests_passed++; printf("[\n    PASS]\u2022%\\n", msg); } else { printf("[\n    FAIL]\u2022%\\n", msg); } } while (0)
7
8 // --- Object Definitions ---
9 OBJECT_START(GrandParentObject)
10 VTABLE(GrandParentObject)
11     METHOD(Object*, create_child)
12 OBJECT_END(GrandParentObject)
13
14 OBJECT_START(ParentObject)
15 VTABLE(ParentObject)
16     METHOD(Object*, create_child)
17 OBJECT_END(ParentObject)
18
19 OBJECT_START(ChildObject)
20 VTABLE(ChildObject)
21 OBJECT_END(ChildObject)
22
23 // --- Constructor / Destructor
24 // Implementations ---
25 CONSTRUCTOR(GrandParentObject)
26 DESTRUCTOR(GrandParentObject) }

```

```

25 CONSTRUCTOR(ParentObject) DESTRUCTOR(
26     ParentObject) }
27 CONSTRUCTOR(ChildObject) DESTRUCTOR(
28     ChildObject) }

// --- Method Implementations ---
29 METHOD_IMPL(GrandParentObject, Object*,
30             create_child) {
31     return NEW_CHILD(ParentObject, self_
32                      obj);
33 } METHOD_IMPL_END

33 METHOD_IMPL(ParentObject, Object*,
34             create_child) {
35     return NEW_CHILD(ChildObject, self_
36                      obj);
37 } METHOD_IMPL_END

// --- VTable Implementations ---
38 VTABLE_IMPL(GrandParentObject, .create_
39             child = _GrandParentObject_create_
40             child)
41 VTABLE_IMPL(ParentObject, .create_child
42             = _ParentObject_create_child)
43 VTABLE_IMPL(ChildObject, )

int main() {
    printf("Hierarchy\u2022\u2022Context\u2022Test
        Suite...\\n");
    CLASS_INIT();

    unsigned long test_group = CREATE_
        GROUP("Hierarchy\u2022Test");

    // 1. Create the root GrandParent
    // object
    Object* gp = NEW_ROOT(
        GrandParentObject, test_group);
    unsigned long gp_id = GET(gp, id,
        unsigned long);

    // 2. Test its properties
    ASSERT_TRUE(GET(gp, group_id,
        unsigned long) == test_group, "
        Root\u2022has\u2022correct\u2022GroupID.");
    ASSERT_TRUE(GET(gp, parent_id,
        unsigned long) == 0, "Root\u2022has\u2022
        ParentID\u2022of\u20220.");

    // 3. Create a child Parent object
    // UPDATED CALL_R: Added '
    // GrandParentObject' as the first
    // argument
    Object* p = CALL_R(GrandParentObject
        , gp, create_child, Object* );
    unsigned long p_id = GET(p, id,
        unsigned long);

    // 4. Test the child's properties
    ASSERT_TRUE(GET(p, group_id,
        unsigned long) == test_group, "
        Child\u2022inherits\u2022GroupID.");
    ASSERT_TRUE(GET(p, parent_id,
        unsigned long) == gp_id, "Child\u2022
        has\u2022correct\u2022ParentID.");

    // 5. Create a grandchild Child
    // object
    // UPDATED CALL_R: Added '

```

```

    ParentObject' as the first
    argument
68 Object* c = CALL_R(ParentObject, p,
    create_child, Object* );
69 ASSERT_TRUE(GET(c, parent_id,
    unsigned long) == p_id, "
    Grandchild has correct ParentID.
    ");
70
71 // 6. Clean up everything
72 DESTROY_BY_GROUP(test_group);
73
74 CLASS_SHUTDOWN();
75 printf("Hierarchy Tests Passed: %d
        /5\n", g_tests_passed);
76 return !(g_tests_passed == 5);
77 }
```

Listing 6: Hierarchy validation test ('test\_hierarchy.c').

The second test, 'test\_errors', demonstrates the framework's robustness. A reliable system must gracefully handle common misuse patterns. This test confirms that attempting to destroy 'NULL' objects or non-existent objects does not cause a crash.

```

1 #include "object.h"
2 #include <stdio.h>
3 #include <stdbool.h>
4
5 static int g_tests_passed = 0;
6 #define ASSERT_TRUE(condition, message)
7     do { if (condition) { g_tests_passed
8         ++; printf("[PASS] %s\n", message)
9     } else { printf("[FAIL] %s\n",
10         message); } } while (0)
11
12 // Define a simple object class for the
13 // test
14 OBJECT_START(ErrorObject)
15 VTABLE(ErrorObject)
16 OBJECT_END(ErrorObject)
17
18 CONSTRUCTOR(ErrorObject)
19 DESTRUCTOR(ErrorObject)
20 }
21 VTABLE_IMPL(ErrorObject, )
22
23 int main() {
24     printf("Error Handling Test Suite
25         ...\n");
26     CLASS_INIT();
27
28     // Test operations on a NULL object
29     Object* null_obj = NULL;
30     DESTROY(null_obj);
31     ASSERT_TRUE(GET(null_obj, id,
32         unsigned long) == 0, "GET on
            NULL returns zero.");
33
34     // Test destroying a non-existent
35     // object ID and group ID
36     // We now use the correct internal
37     // function name: _class_destroy_by_
38     // _id
39     _class_destroy_by_id(999999);
40     DESTROY_BY_GROUP(999999);
41     ASSERT_TRUE(true, "Destroy by
42         invalid ID/Group did not crash."
43     );
44
45     // Test double destruction
46     unsigned long group = CREATE_GROUP("Double
        DTest");
47     Object* obj = NEW_ROOT(ErrorObject,
        group);
48     unsigned long id = GET(obj, id,
49         unsigned long);
50
51     // First destruction via the object
52     // pointer
53     DESTROY(obj);
54
55     // Second destruction attempt via
56     // the (now invalid) ID
57     _class_destroy_by_id(id);
58     ASSERT_TRUE(true, "Double
        destruction did not crash.");
59
60     CLASS_SHUTDOWN();
61     printf("Error Handling Tests Passed:
        %d/3\n", g_tests_passed);
62     return !(g_tests_passed == 3);
63 }
```

```

    invalid ID/Group did not crash."
    );

// Test double destruction
unsigned long group = CREATE_GROUP("Double
    DTest");
Object* obj = NEW_ROOT(ErrorObject,
    group);
unsigned long id = GET(obj, id,
    unsigned long);

// First destruction via the object
// pointer
DESTROY(obj);

// Second destruction attempt via
// the (now invalid) ID
_class_destroy_by_id(id);
ASSERT_TRUE(true, "Double
    destruction did not crash.");

CLASS_SHUTDOWN();
printf("Error Handling Tests Passed:
    %d/3\n", g_tests_passed);
return !(g_tests_passed == 3);
}
```

Listing 7: Error handling and resilience test ('test\_errors.c').

### 2.3.5 High-Level Usage Examples

To illustrate the framework's intended usage, we developed two example programs. The first, 'basic\_usage.c', demonstrates the simplest workflow: creating a single object, interacting with it, and destroying it.

```

1 #define _GNU_SOURCE
2 #include "object.h"
3 #include <stdio.h>
4 #include <string.h>
5
6 OBJECT_START(Greeting)
7     char* target_name;
8 VTABLE(Greeting)
9     METHOD(void, say_hello)
10    OBJECT_END(Greeting, const char* name);
11
12 CONSTRUCTOR(Greeting, const char* name)
13     self->target_name = strdup(name);
14 DESTRUCTOR(Greeting)
15     free(self->target_name);
16 }
17
18 METHOD_IMPL(Greeting, void, say_hello)
19     printf("Hello, %s! (from Object ID: %lu)\n",
20         self->target_name, self
21             ->id);
22 METHOD_IMPL_END
23
24 VTABLE_IMPL(Greeting, .say_hello = _Greeting_say_hello)
25
26 int main() {
27     printf("--- Basic Usage Example ---\n\n");
28 }
```

```

26 CLASS_INIT();
27 printf("1. Creating a 'Greeting' object instance...\n");
28 Object* greeter = NEW_ROOT(Greeting,
29     0, "World");
30 printf("\n2. Interacting with the object...");
31 printf("Object's unique ID is: %lu\n", GET(greeter, id, unsigned long));
32 printf("Calling the 'say_hello' method:\n-> ");
33 CALL(Greeting, greeter, say_hello);
34 printf("\n3. Destroying the object...\n");
35 DESTROY(greeter);
36 printf("\n4. Shutting down the Class Framework...\n");
37 CLASS_SHUTDOWN();
38 printf("\n--- Example Finished Cleanly ---\n");
39 return 0;
}

```

Listing 8: A simple demonstration of the framework API ('basic\_usage.c').

The second example, 'lifecycle\_and\_groups.c', showcases the framework's powerful managed life-cycle features. It demonstrates the creation of an object group and the instantiation of a hierarchy of objects within it. The example culminates in a single call to 'DESTROY\_BY\_GROUP', which cleans up all objects created during the workflow, highlighting the system's ability to prevent memory leaks.

```

1 #define _GNU_SOURCE
2 #include "object.h"
3 #include <stdio.h>
4 #include <string.h>
5
6 OBJECT_START(Task) char* task_name;
7     VTABLE(Task) METHOD(Object*, add_subtask, const char* subtask_name)
8     OBJECT_END(Task, const char* name);
9
10 OBJECT_START(Subtask) char* subtask_name;
11     VTABLE(Subtask) OBJECT_END(Subtask, const char* name);
12
13 CONSTRUCTOR(Task, const char* name) self
14     ->task_name = strdup(name);
15 DESTRUCTOR(Task) free(self->task_name); }
16
17 CONSTRUCTOR(Subtask, const char* name) self
18     ->subtask_name = strdup(name);
19 DESTRUCTOR(Subtask) free(self->subtask_name); }
20
21 METHOD_IMPL(Task, Object*, add_subtask, const char* subtask_name)
22     printf("Creating a child subtask.\n",
23         self->task_name, self->id);
24     return NEW_CHILD(Subtask, self_obj,
25         subtask_name);
26
27 METHOD_IMPL_END

```

```

16 VTABLE_IMPL(Task, .add_subtask = _Task_add_subtask)
17 VTABLE_IMPL(Subtask, )
18
19 void print_object_details(Object* obj,
20     const char* obj_name) {
21     if (!obj) { printf("Cannot print details for NULL object '%s'.\n",
22         obj_name); return; }
23     printf("Details for %s:\n", obj_name);
24     printf("ObjectID: %lu\n", GET(obj, id, unsigned long));
25     printf("GroupID: %lu\n", GET(obj, group_id, unsigned long));
26     printf("ParentID: %lu (0 means it is a root object)\n",
27         GET(obj, parent_id, unsigned long));
28 }
29
30 int main() {
31     printf("--- Lifecycle and Group Management Example ---\n\n");
32     CLASS_INIT();
33     printf("1. Creating a 'Workflow' group...\n");
34     unsigned long workflow_group_id =
35         CREATE_GROUP("Process_daily_sales_report");
36     printf("-> New Group ID: %lu\n\n", workflow_group_id);
37     printf("2. Creating a root 'Task' object...\n");
38     Object* main_task = NEW_ROOT(Task,
39         workflow_group_id, "Generate Report");
40     print_object_details(main_task, "Main Task");
41     printf("\n3. Creating child objects...\n");
42     Object* subtask1 = CALL_R(Task, main_task, add_subtask, Object*, "Fetch data");
43     Object* subtask2 = CALL_R(Task, main_task, add_subtask, Object*, "Format PDF");
44     printf("\nVerifying child object details...\n");
45     print_object_details(subtask1, "Subtask 1");
46     print_object_details(subtask2, "Subtask 2");
47     printf("\n4. Workflow complete. Destroying entire group...\n");
48     DESTROY_BY_GROUP(workflow_group_id);
49     printf("-> Group %lu and all its objects have been destroyed.\n",
50         workflow_group_id);
51     printf("\n5. Shutting down...\n");
52     CLASS_SHUTDOWN();
53     printf("\n--- Example Finished Cleanly ---\n");
54     return 0;
55 }

```

Listing 9: Demonstration of group-based lifecycle management ('lifecycle\_and\_groups.c').

### 2.3.6 Performance and Concurrency Validation

For a system intended to be the backbone of an intelligent agent, performance and thread-safety are core requirements. We developed two final tests to validate these aspects. The ‘`test_stress`’ program measures the raw throughput of the object creation and destruction pipeline by creating and destroying hundreds of thousands of objects. The ‘`test_concurrency`’ program validates the framework’s thread-safety by spawning multiple POSIX threads that simultaneously create and destroy their own sets of object groups.

```

1 #define _GNU_SOURCE
2 #include "object.h"
3 #include <stdio.h>
4 #include <time.h>
5
6 #define NUM_GROUPS 50
7 #define OBJECTS_PER_GROUP 2000
8 #define TOTAL_OBJECTS (NUM_GROUPS *
9     OBJECTS_PER_GROUP)
10
11 OBJECT_START(StressObject) int value;
12     VTABLE(StressObject) OBJECT_END(
13         StressObject, int val);
14 CONSTRUCTOR(StressObject, int val) self
15     ->value = val; DESTRUCTOR(
16         StressObject) }
17 VTABLE_IMPL(StressObject, )
18
19 int main() {
20     printf("Stress\u2014Test...\n");
21     CLASS_INIT();
22     unsigned long groups[NUM_GROUPS];
23     clock_t start = clock();
24     for (int i = 0; i < NUM_GROUPS; ++i)
25         { groups[i] = CREATE_GROUP("Stress\u2014Test\u2014Group"); }
26     long long total_created = 0;
27     for (int i = 0; i < NUM_GROUPS; ++i)
28         {
29             for (int j = 0; j < OBJECTS_PER_
30                 GROUP; ++j) {
31                 NEW_ROOT(StressObject,
32                     groups[i], j);
33                 total_created++;
34                 if (total_created % 1000 ==
35                     0) { printf("\r\u2014->\u2014
36                     Creating:\u2014%lld\u2014/\u2014%d",
37                     total_created, TOTAL_
38                     OBJECTS); fflush(stdout)
39                     ; }
40             }
41         }
42     printf("\r\u2014->\u2014Creation\u2014complete.\u2014\u2014
43         \u2014\u2014\u2014\u2014\u2014\u2014\n");
44     clock_t mid = clock();
45     for (int i = 0; i < NUM_GROUPS; ++i)
46         { DESTROY_BY_GROUP(groups[i]); }
47     clock_t end = clock();
48     printf("\u2014->\u2014Destruction\u2014complete.\n"
49         );
50     double creation_time = ((double)(mid -
51         start)) / CLOCKS_PER_SEC;
52 }
```

```
        - start)) / CLOCKS_PER_SEC;
    double destruction_time = ((double)(end - mid)) / CLOCKS_PER_SEC;
    printf("Creation_time: %.4f s, "
           "Destruction_time: %.4f s\n",
           creation_time, destruction_time)
    ;
    CLASS_SHUTDOWN();
    printf("Stress_Test_Passed.\n");
    return 0;
}
```

Listing 10: A stress test for high-volume object management ('test\_stress.c').

```
1 #define __GNU_SOURCE
2 #include "object.h"
3 #include <stdio.h>
4 #include <pthread.h>
5
6 #define NUM_THREADS 8
7 #define GROUPS_PER_THREAD 10
8 #define OBJECTS_PER_GROUP 100
9
10 OBJECT_START(ThreadObject) int val;
11     VTABLE(ThreadObject) OBJECT_END(
12         ThreadObject, int v);
13 CONSTRUCTOR(ThreadObject, int v) self->
14     val = v; DESTRUCTOR(ThreadObject) }
15 VTABLE_IMPL(ThreadObject, )
16
17 void* worker_thread_func(void* arg) {
18     int thread_id = *(int*)arg;
19     unsigned long my_groups[GROUPS_PER_
20         THREAD];
21     for (int i = 0; i < GROUPS_PER_
22         THREAD; ++i) {
23         my_groups[i] = CREATE_GROUP("_
24             Thread\u_group");
25         for (int j = 0; j < OBJECTS_PER_
26             GROUP; ++j) {
27             if (!NEW_ROOT(ThreadObject,
28                 my_groups[i], j)) {
29                 printf("Thread\u%d:\u NEW
30                     ROOT\u failed.\n",
31                         thread_id); return NULL; }
32         }
33     }
34     for (int i = 0; i < GROUPS_PER_
35         THREAD; ++i) { DESTROY_BY_GROUP(
36             my_groups[i]); }
37     printf("->\uThread\u%d:\u Complete.\n",
38             , thread_id);
39     return NULL;
40 }
41
42 int main() {
43     printf("Concurrency\u Test\u Suite...\n");
44     CLASS_INIT();
45     pthread_t threads[NUM_THREADS];
46     int thread_ids[NUM_THREADS];
47     for (int i = 0; i < NUM_THREADS; ++i)
48     {
49         thread_ids[i] = i;
50         pthread_create(&threads[i], NULL,
51             , worker_thread_func, &
52             thread_ids[i]);
53     }
54 }
```

```

37     for (int i = 0; i < NUM_THREADS; ++i
38         ) { pthread_join(threads[i],
39             NULL); }
40     CLASS_SHUTDOWN();
41     printf("Concurrency\u2014Test\u2014Passed.\u2014
42         Check\u2014for\u2014leak\u2014warnings)\n";
43     return 0;
44 }
```

Listing 11: A concurrency test using multiple threads ('test\_concurrency.c').

Together, this comprehensive suite of examples and tests provides strong validation for the AGENK Object Framework. It demonstrates not only that the core philosophical concepts of abstraction and specialization can be implemented in C, but also that the resulting system is robust, performant, and ready to serve as the foundation for a scalable intelligent agent like AGENK. Having established this complete system for defining and managing individual objects, we now turn to the question of what are contained inside these objects or how each object is defined.

## 2.4 Where the object resides?

Let's go even more deeper. We have built till now the abstract processing/live framework of the object. But an object is not just when it is being used, it maintain its state when not live. To be more precise, an object has data/contents/knowledge (can be properties, functions, etc) associated with it which is dynamic, persistent and specific to the object. So, to build that persistent specialized knowledge base of the objects we will build an *Object Store Database* in addition to our current *Object Processing Framework*.

So, the objectStore data structure is denoted by "{' }" and the properties inside it by "[]" and methods inside it as "()=>{ }". These are properties and methods that are not standard for objects and are specific to objects. Standard properties and methods are attached to each object at runtime via the object-oriented class framework we have defined.

So, we can say that the above framework was for standard properties and methods that are attached to object at runtime while this is a framework for specific properties and methods of objects which are stored in database even when the specific object is not running and it is attached to the live object when the object is called, read, etc.

All objects will have a unique id for storage and retrieval called object\_id and when it is called like it is being processed by our standard object/class framework then it also gets a unique processing\_id

something like processing\_id = object\_id + session\_id (group\_id).

So, let us think how our non-standard objects are fetched to our standard processing framework when we need to get a particular property or call a specific method of(non-standard)/on(standard) our object. So, let's understand with the help of example - **Integration by parts:**  $\int u dv = uv - \int v du$

```

1 some_object {
2     causal_state :
3     [
4         "Integrate\u2014the\u2014inputs\u2014by\u2014the
5             \u2014method\u2014of\u2014integration\u2014
6                 \u2014by\u2014parts."
7         1,
8         // if there is just one data inside
9         a property then it can be
10        written like property: "data"
11        too.
12        some_property: "Hi,\u2014How\u2014Are\u2014You",
13        program_space :
14        [
15            (u, dv, x_start, x_end) => {
16                v = integrate(dv, x_start, x
17                    _end),
18                du = differentiate(u),
19                uv = multiply(u, v),
20                v_du = integrate(multiply(v,
21                    du), x_start, x_end),
22                result = subtract(uv, v_du),
23                return result
24            },
25            (some_other_arguements) => {
26                some_method
27            }
28            // a single object can contain
29            many programs.
30        ],
31        some_data:
32        [
33            property: "some_property"Co
34            data:
35            [
36                {
37                    isCorrect: true,
38                    isProgram: false
39                },
40                {
41                    status:
42                        "Completed"
43                }
44            ]
45        }
46    }
47 }
```

Listing 12: Example of stored Object

The object can be nested too. What i mean is that properties and methods cn be nested and recursive too like how objects are, think like JSON structure, it can be nested too.

## 2.5 Implementation of the object store in C

The architectural blueprint for our intelligent agent rests upon two pillars: a high-speed runtime for ex-

ecuting object behaviors and a robust persistence layer for storing object knowledge. The *Object Processing Framework*, detailed in the previous section, provides the former. This section provides an exhaustive exploration of the latter: the **Object Store Framework**. This framework is not merely a database; it is the carefully engineered foundation of the agent’s long-term memory, its identity, and its capacity to learn and evolve.

### 2.5.1 Architectural Philosophy: The Soul and the Machine

Our fundamental design principle is the strict and deliberate separation of an object’s persistent identity—its “soul”—from its transient, in-memory manifestation—the “machine.”

**The ”Soul” - Persistent Blueprints in the Object Store:** An object’s true essence is its knowledge. This includes its properties (e.g., a name, a set of rules, a target URL) and its specialized methods (e.g., a script to parse a specific data format). This collection of data is the object’s blueprint. It is immutable in the sense that it represents a definitive, versioned state of the object’s knowledge. The Object Store’s sole responsibility is to be the high-fidelity, transactional library for these blueprints. It is optimized for durability, integrity, and efficient retrieval, ensuring that this core knowledge is never corrupted or lost. Each blueprint is assigned a permanent, unique ‘object\_id’ that acts as its globally unique identifier across all time and all processes.

**The ”Machine” - Transient Instances in the Live Framework:** The *Object Processing Framework* is the factory that builds machines from these blueprints. When a task requires an object’s capabilities, it doesn’t access the database directly. Instead, it requests an \*instantiation\* of the object. The live framework loads the blueprint from the Object Store, constructs an in-memory ‘Object\*’ container, and gives this instance a temporary, runtime-specific ‘ObjectID’. This live instance is the ”machine”—a temporary, stateful actor that can be used, modified, and then discarded when a task is complete. Its existence is ephemeral, but it is empowered by the persistent knowledge of its blueprint.

**Key Insight - Enabling Concurrency and Versioning:** This separation is what unlocks true concurrency and scalability. If multiple tasks need

to work with ”user\_profile\_42,” they do not contend for a single, shared in-memory object. Instead, each task gets its own independent instance, a clean copy of the blueprint loaded from the Object Store. Task A can modify its instance’s ‘last\_login’ property without affecting Task B’s instance. If Task A decides to persist its changes, it writes its modified blueprint back to the Object Store, creating a new, definitive state. This model naturally handles concurrency, prevents race conditions on shared memory, and provides a clear path for future versioning and transactional updates.

### 2.5.2 The Technology Stack: A Foundation Built on C99 Portability

Our commitment to universal portability (across operating systems from Linux to Windows, and architectures from x86 to ARM) dictated every technological choice. We exclusively selected libraries written in pure C99, vendoring them directly into our project to eliminate external dependencies.

**Directory Structure as Architectural Expression:** The project’s file layout is a direct reflection of our modular design. The new ‘object\_store’ module was introduced as a sibling to the ‘objects’ module, creating a clear boundary between persistence and runtime.

```

AGENK/
  └── CMakeLists.txt      (The master build or...
  └── deps/
    └── cJSON/
    └── flatcc/
    └── lmdb/
    └── md5/
  └── src/
    └── objects/
      └── core/
      └── examples/
      └── test/
      └── object.h
    └── object_store/     (The new object pers...
      └── core/
        └── fb_serializer.c
        └── object_store.c
        └── object.fbs
      └── test/
        └── test_object_store.c
      └── fb_serializer.h
      └── object_store.h

```

## Serialization: Choosing Speed and Safety with FlatBuffers

**FlatBuffers:** The most critical decision for the persistence layer was the serialization format. We evaluated several options:

- **JSON/XML:** Human-readable and flexible, but their text-based nature requires a costly parsing step on every read, which involves significant string manipulation and memory allocation. This was deemed too slow for an agent that needs instant access to its knowledge.
- **Protocol Buffers:** A strong contender, offering a compact binary format and schema evolution. However, it still requires a parsing step to convert the on-wire format to in-memory C structs.
- **FlatBuffers:** The chosen solution. FlatBuffers was selected because it is not merely a serialization library; it is a **zero-copy binary object-mapping format**. When data is read from disk, it is already in a parsable format. Accessing a field is a simple pointer arithmetic operation, not a deserialization process. This eliminates the single greatest performance bottleneck in persistence-heavy applications. The formal schema ('object.fbs') also provides compile-time safety and a clear, evolving contract for our data structures.

Our schema is designed for maximum flexibility. A 'FieldValue' union allows any field to contain any of our supported types, from simple integers to complex, recursively nested objects.

```

1 // ===== object.fbs =====
2 // Defines the serializable structure
3 // for a generic object in the AGENK
4 // framework.
5 // This schema uses FlatBuffers for high
6 // -performance, zero-copy access to
7 // persistent object data.
8 namespace objstore;
9
10 // A union of all supported primitive
11 // and complex data types for a field's
12 // value.
13 // FlatBuffers automatically adds a
14 // hidden type tag, making this a
15 // tagged union.
16 union FieldValue {
17   IntValue,
18   BoolValue,
19   FloatValue,
20   StringValue,
21   ArrayValue,
22   MapValue,
23   ObjectValue,
24   PointerValue
25 }
```

```

18 }
19 // Wrapper tables for primitive types.
20 table IntValue { value: long; } //
21 // Using 'long' for int64_t
22 table BoolValue { value: bool; }
23 table FloatValue { value: float; }
24 table StringValue { value: string; }
25
26 // An array is a vector of nested Field
27 // tables.
28 table ArrayValue { values: [Field]; }
29
30 // A key-value pair for use in maps.
31 table MapPair { key: string (
32   required); value: FieldValue; }
33
34 // A map is a vector of key-value pairs.
35 table MapValue { pairs: [MapPair]; }
36
37 // A wrapper for embedding another
38 // complete Object.
39 table ObjectValue { object: Object; }
40
41 // A reference to another object in the
42 // store, identified by its persistent
43 // key.
44 table PointerValue { object_id: string (
45   required); }
46
47 // The root type for any stored object.
48 // Contains its unique persistent ID,
49 // the name of the table it belongs to,
50 // and a vector of its dynamic fields.
51 table Object {
52   object_id: string (required);
53   table_name: string (required);
54   fields: [Field];
55 }
56
57 // Declare Object as the root type,
58 // allowing it to be used as the top-
59 // level
60 // element in a FlatBuffer.
61 root_type Object;
62
63 }
```

Listing 13: The FlatBuffers schema defining the universal object structure (`src/object_store/core/object.fbs`).

**Database Engine: The Unparalleled Efficiency of LMDB:** Our database choice, LMDB, was a direct consequence of selecting FlatBuffers. LMDB is a memory-mapped key-value store. This means the operating system maps the database file directly into the application's address space. When we request an object, LMDB provides a direct pointer to that object's data within the memory-

mapped file.

**Insight: The Synergy of LMDB and Flat-Buffers.** This is where the architecture becomes truly elegant. The pointer returned by LMDB points to a raw buffer that is \*already in the Flat-Buffers format\*. Our code can take this pointer and immediately begin navigating the object's data structure with zero parsing and zero memory allocation. This combination provides performance that approaches that of accessing data directly in RAM, while still providing the durability of on-disk storage with full ACID transactional guarantees.

**Key Generation: Engineering for Time and Distribution:** An object's 'object\_id' is not just a random number. We engineered a 32-byte composite key to provide valuable database characteristics out of the box.

- **Bytes 0-7 (Inverted Timestamp):** By subtracting the current nanosecond timestamp from 'UINT64\_MAX', we ensure that keys generated later have a lexicographically smaller value. In a B-tree database like LMDB, this means newly created objects are naturally clustered together at the "front" of the database, dramatically improving the performance of queries that scan for recent items.
- **Bytes 8-23 (MD5 Hash):** To prevent all keys created in a tight loop from clustering around the same timestamp prefix (which can unbalance the B-tree), we introduce a 16-byte MD5 hash of a random nonce. This ensures that even keys created microseconds apart are spread pseudo-randomly across the entire keyspace.
- **Bytes 24-31 (Nonce):** To guarantee uniqueness against the theoretical possibility of a timestamp collision, we append the random nonce itself to the key.

### 2.5.3 Build System Orchestration with CMake

A complex, multi-dependency C project requires a sophisticated and automated build system. Our 'CMakeLists.txt' is engineered to be a "one-touch" build solution.

**Insight: Building the Compiler to Build the Code:** A central challenge was that our code depends on headers that don't exist until a tool

('flatcc\_cli') is run, but that tool doesn't exist until we compile it from our 'deps/' folder. CMake's dependency management elegantly solves this chicken-and-egg problem.

1. **'add\_subdirectory(deps/flatcc)'**: This command is the first key. It tells CMake to treat 'flatcc' as an integrated part of our build. CMake learns about all of 'flatcc''s internal targets, including the 'flatcc\_cli' executable and the 'flatccrt' runtime library.
2. **'add\_custom\_command(...)'**: This defines the code generation step. Its 'COMMAND' argument uses the generator expression '\$<TARGET\_FILE:flatcc\_cli>', which acts as a placeholder that CMake will replace with the actual path to the 'flatcc\_cli' executable \*after\* it has been built. Its 'DEPENDS' argument explicitly lists 'flatcc\_cli', telling CMake this command cannot run until that executable is ready.
3. **'add\_dependencies(object\_store generate\_object\_headers)'**: This is the final link in the chain, telling CMake that the entire 'object\_store' library cannot be compiled until the dummy 'generate\_object\_headers' target (which represents the completion of the custom command) is finished.

This dependency graph guarantees a correct, in-order build every time, completely transparently to the developer.

```
1 # =====
2 # CMakeLists.txt for AGENK
3 #
4 # Version: 1.0.0
5 # =====
6 #
7 # =====
8 # SECTION 1: PROJECT PREAMBLE
9 #           & BUILD STANDARDS
10 # =====
11 cmake_minimum_required(VERSION 3.16)
12 if(POLICY CMP0048)
13     cmake_policy(SET CMP0048 NEW)
14 endif()
15
16 project(AGENK VERSION 1.1.9 LANGUAGES C)
17
18 set(CMAKE_C_STANDARD 99)
19 set(CMAKE_C_STANDARD_REQUIRED ON)
20 set(CMAKE_C_EXTENSIONS OFF)
21
22 if(NOT CMAKE_BUILD_TYPE)
23     set(CMAKE_BUILD_TYPE Debug)
24 endif()
25
26
```

```

27 message(STATUS "Build type set to: ${CMAKE_BUILD_TYPE}")
28
29 # =====
30 # SECTION 2: DEPENDENCY MANAGEMENT
31 # =====
32 # =====
33 find_package(Threads REQUIRED)
34
35 set(FLATCC_TEST OFF CACHE BOOL "Disable flatcc's internal tests")
36 set(FLATCC_SAMPLES OFF CACHE BOOL "Disable flatcc's internal samples")
37 add_subdirectory(deps/flatcc)
38
39 add_library(lmdb STATIC deps/lmdb/mdb.c
40           deps/lmdb/midl.c)
41 target_include_directories(lmdb PUBLIC ${CMAKE_CURRENT_SOURCE_DIR}/deps/lmdb)
42
43 add_library(md5 STATIC deps/md5/md5.c)
44 target_include_directories(md5 PUBLIC ${CMAKE_CURRENT_SOURCE_DIR}/deps/md5)
45
46 set(TEST_EXTRA_LIBS "")
47 if(EXISTS ${CMAKE_CURRENT_SOURCE_DIR}/deps/cJSON/cJSON.c)
48   message(STATUS "cJSON source found, building library.")
49   add_library(cjson STATIC deps/cJSON/cJSON.c)
50   target_include_directories(cjson PUBLIC ${CMAKE_CURRENT_SOURCE_DIR}/deps/cJSON)
51   list(APPEND TEST_EXTRA_LIBS cjson)
52   set(CJSON_FOUND TRUE)
53 endif()
54
55 # =====
56 # SECTION 3: AUTOMATED CODE GENERATION
57 # =====
58 set(FBS_SCHEMA ${CMAKE_CURRENT_SOURCE_DIR}/src/object_store/core/object.fbs)
59 set(GENERATED_INCLUDE_DIR ${CMAKE_CURRENT_BINARY_DIR}/generated/object_store)
60 file(MAKE_DIRECTORY ${GENERATED_INCLUDE_DIR})
61
62 set(GENERATED_FILES
63   ${GENERATED_INCLUDE_DIR}/object_builder.h
64   ${GENERATED_INCLUDE_DIR}/object_reader.h
65   ${GENERATED_INCLUDE_DIR}/object_verifier.h
66   ${GENERATED_INCLUDE_DIR}/flatbuffers_common.h
67 )
68
69 add_custom_command(
70   OUTPUT ${GENERATED_FILES}
71   COMMAND ${CMAKE_COMMAND} -D FBS_SCHEMA=${FBS_SCHEMA} -D TARGET_FILE=flatcc_cli -D
72   DEPENDS ${FBS_SCHEMA} flatcc_cli
73   COMMENT "Generating C headers from "
74   )
75 add_custom_target(generate_object_headers DEPENDS ${GENERATED_FILES})
76
77 # =====
78 # SECTION 4: CORE LIBRARY DEFINITIONS
79 # =====
80
81 add_library(class_framework STATIC
82             src/objects/core/object_impl.c
83             src/objects/core/object_registry.c
84             src/objects/core/object_db.c
85           )
86 target_include_directories(class_framework PUBLIC ${CMAKE_CURRENT_SOURCE_DIR}/src/objects)
87 target_link_libraries(class_framework PRIVATE lmdb Threads::Threads)
88
89 add_library(object_store STATIC
90             src/object_store/core/fb_serializer.c
91             src/object_store/core/object_store.c
92           )
93 add_dependencies(object_store generate_object_headers)
94 target_include_directories(object_store PUBLIC
95   ${CMAKE_CURRENT_SOURCE_DIR}/src/object_store
96   ${GENERATED_INCLUDE_DIR}
97   ${CMAKE_CURRENT_SOURCE_DIR}/deps/flatcc/include
98 )
99 target_link_libraries(object_store
100    PRIVATE
101      flatccrt
102      lmdb
103      md5
104      Threads::Threads
105    )
106 if(CJSON_FOUND)
107   target_link_libraries(object_store PRIVATE cJSON)
108   target_compile_definitions(object_store PRIVATE cJSON_ENABLED)
109 endif()
110
111 # =====
112 # SECTION 5: EXECUTABLE TARGETS (TESTS)
113 # =====
114
115 add_executable(test_stress src/objects/test/test_stress.c)
116 add_executable(test_errors src/objects/test/test_errors.c)
117 add_executable(test_hierarchy src/objects/test/test_hierarchy.c)
118 add_executable(test_concurrency src/objects/test/test_concurrency.c)
119
120 target_link_libraries(test_stress PRIVATE class_framework)
121 target_link_libraries(test_errors PRIVATE class_framework)
122 target_link_libraries(test_hierarchy PRIVATE class_framework)
123

```

```
125 target_link_libraries(test_concurrency
126     PRIVATE class_framework Threads::Threads)
127
128 add_executable(test_object_store src/
129     object_store/test/test_object_store.
130     c)
131 # **FIX**: Add the math library 'm' to
132 # the list of linked libraries.
133 target_link_libraries(test_object_store
134     PRIVATE
135         object_store
136         Threads::Threads
137         ${TEST_EXTRA_LIBS}
138         m # Link against the math
139         library
140     )
141
142 # =====
143 # SECTION 6: TESTING INTEGRATION
144 # =====
145
146 enable_testing()
147 add_test(NAME Framework.StressTest
148     COMMAND test_stress)
149 add_test(NAME Framework.ErrorHandling
150     COMMAND test_errors)
151 add_test(NAME Framework.Hierarchy
152     COMMAND test_hierarchy)
153 add_test(NAME FrameworkConcurrency
154     COMMAND test_concurrency)
155 add_test(NAME ObjectStore.Core
156     COMMAND test_object_store)
```

Listing 14: The root `CMakeLists.txt`, orchestrating dependency builds and code generation.

#### 2.5.4 The Implementation in Detail

**The Serialization Layer ('fb\_serializer'):** This module is the heart of our data translation. The public header, 'fb\_serializer.h', provides the clean C 'struct' definitions that the rest of our application uses, abstracting away the complexities of the FlatBuffers format.

```
1 #ifndef FB_SERIALIZER_H
2 #define FB_SERIALIZER_H
3
4 #include <stddef.h>
5 #include <stdint.h>
6 #include <stdbool.h>
7
8 // Forward declarations for the C data
9 // structures that represent a
10 // serialized object.
11 // This allows them to be used in
12 // pointers within the struct
13 // definitions themselves.
14
15 // =====
```

```

77 * @struct c_field_value_t
78 * @brief A tagged union representing
79     the value of a field.
80 */
81 struct c_field_value_t {
82     c_field_value_type_t type;
83     union {
84         int64_t int_val;
85         bool bool_val;
86         float float_val;
87         char* string_val;           // Used for StringValue and
88                                     // PointerValue
89         c_field_t* array_values;   // Linked list of fields for an
90                                     // array
91         c_map_pair_t* map_pairs;   // Linked list of key-value
92                                     // pairs for a map
93         c_object_t* object_val;    // A
94                                     // nested, embedded object
95     } data;
96 };
97 /**
98 * @struct c_field_t
99 * @brief A single named field (a key-
100    value pair). Forms a linked list
101   for an
102   object's or an array's fields.
103 */
104 struct c_field_t {
105     char* name;
106     c_field_value_t* value;
107     c_field_t* next;
108 };
109 /**
110 * @struct c_map_pair_t
111 * @brief A single key-value pair within
112     a map. Forms a linked list.
113 */
114 struct c_map_pair_t {
115     char* key;
116     c_field_value_t* value;
117     c_map_pair_t* next;
118 };
119 /**
120 * @struct c_object_t
121 * @brief The top-level C representation
122     of a deserialized object.
123 */
124 // =====
125 // SECTION 3: MEMORY MANAGEMENT API
126 // =====
127 // Since the deserialization process
128 // allocates memory for the C structs,
129 // these functions provide the correct
130 // way to recursively free that memory.
131
132 // =====
133 void free_c_object(c_object_t* obj);
134 void free_c_field_list(c_field_t* head);
135 void free_c_field_value(c_field_value_t*
136     fv);
137 void free_c_map_pair_list(c_map_pair_t*
138     head);
139
140 // =====
141 // SECTION 4: C-STRUCT CREATION HELPERS
142 //
143 // Convenience functions for creating
144 // `c_field_value_t` structs for each
145 // data type. These are useful for
146 // programmatically building an object
147 // in C before serialization.
148 //
149 c_field_value_t* create_c_string_value(
150     const char* val);
151 c_field_value_t* create_c_int_value(
152     int64_t val);
153 c_field_value_t* create_c_bool_value(
154     bool val);
155 c_field_value_t* create_c_float_value(
156     float val);
157 c_field_value_t* create_c_pointer_value(
158     const char* object_id);
159 c_field_value_t* create_c_none_value(
160     void);
161
162 /**
163 * @brief Serializes a C object
164   structure into a FlatBuffer binary
165   representation.
166 *
167 * @param obj The C object to serialize.
168 * @param out_buffer Pointer to a void*
169   that will receive the newly
170   allocated buffer.
171 * The caller is
172   responsible for freeing this memory
173 *
174 * @param out_size Pointer to a size_t
175   that will be populated with the
176   buffer's size.
177 * @return FB_SERIALIZER_OK on success,
178   or an error code on failure.
179 */
180 fb_serializer_status_t serialize_c_
181 object(const c_object_t* obj, void** out_buffer, size_t* out_size);
182
183 /**
184 * @brief Deserializes a FlatBuffer from
185   a binary buffer into a C object
186   structure.
187 *
188 * @param buffer Pointer to the
189   FlatBuffer data.
190 * @param size The size of the buffer.
191 * @param out_obj Pointer to a c_object_
192   t* that will receive the newly
193   allocated object.

```

```
179     * allocated C object.  
180     * The caller is  
181     * responsible for freeing this object  
182     * with free_c_object().  
183     */  
184     fb_serializer_status_t deserialize_to_c_  
185     object(const void* buffer, size_t  
186     size, c_object_t** out_obj);  
187  
188     /**  
189     * @brief Parses a JSON string and  
190     * converts it into a C object  
191     * structure.  
192     * @param json_string The null-  
193     * terminated JSON string.  
194     * @param default_object_id An object ID  
195     * to use if "object_id" is not in  
196     * the JSON.  
197     * @param default_table_name A table  
198     * name to use if "table_name" is not  
199     * in the JSON.  
200     * @param out_c_obj Pointer to a c_  
201     * object_t* that will receive the  
202     * allocated C object.  
203     * The caller is  
204     * responsible for freeing this object  
205     *  
206     * @return FB_SERIALIZER_OK on success,  
207     * or an error code on failure.  
208     */  
209     fb_serializer_status_t fb_object_from_  
210     json_string(  
211     const char *json_string,  
212     const char *default_object_id,  
213     const char *default_table_name,  
214     c_object_t **out_c_obj);  
215  
216 #endif // FB_SERIALIZER_H
```

```
8 #include <flatcc/flatcc_verifier.h>
9
10 #ifdef CJSON_ENABLED
11 #include <cjson/cJSON.h>
12 #endif
13
14 #include <stdio.h>
15 #include <stdlib.h>
16 #include <string.h>
17 #include <math.h>
18
19 // --- Static Function Forward
20 // Declarations ---
21 static objstore_Object_ref_t create_fb_
22 object_recursive(flatcc_builder_t *B,
23 , const c_object_t *obj);
24 static objstore_Field_ref_t create_fb_
25 field(flatcc_builder_t *B, const c_
26 field_t *cf);
27 static int create_fb_field_value_union(
28 flatcc_builder_t *B, const c_field_
29 value_t *fv, objstore_FieldValue_
30 union_ref_t *out_union_ref);
31 static fb_serializer_status_t parse_fb_
32 object_recursive(objstore_Object_
33 table_t fb_obj_table, c_object_t **
34 out_c_obj);
35 static fb_serializer_status_t parse_fb_
36 field(objstore_Field_table_t fb_
37 field_table, c_field_t **out_c_field
38 );
39 static fb_serializer_status_t parse_fb_
40 field_value_union(
41 objstore_FieldValue_union_t fb_union_
42 _struct,
43 flatbuffers_union_type_t fb_union_
44 type_tag,
45 c_field_value_t **out_fv);
46 #ifdef CJSON_ENABLED
47 static c_field_value_t* c_field_value_
48 from_cjson_recursive(cJSON *item,
49 const char* parent_key_for_list_
50 items, fb_serializer_status_t *
51 current_status);
52 static fb_serializer_status_t c_object_
53 from_cjson_node(cJSON *json_node,
54 const char* default_object_id,
55 const char *default_table_name, c_
56 object_t **out_obj);
57#endif
58
59 // --- ANSI C Compatible String
60 // Duplication ---
61 static char* ansi_strdup(const char* s)
62 {
63     if (!s) return NULL;
64     size_t len = strlen(s) + 1;
65     char* new_s = (char*)malloc(len);
66     if (!new_s) {
67         perror("ERROR: ansi_strdup - "
68               "malloc failed");
69         return NULL;
70     }
71     memcpy(new_s, s, len);
72     return new_s;
73 }
74
75 // --- Memory Management Functions ---
76 void free_c_map_pair_list(c_map_pair_t *
77 head) {
```

```

49     c_map_pair_t *current = head;
50     c_map_pair_t *next;
51     while (current != NULL) {
52         next = current->next;
53         free(current->key);
54         free_c_field_value(current->
55             value);
56         free(current);
57         current = next;
58     }
59 }
60 void free_c_field_value(c_field_value_t
61     *fv) {
62     if (!fv) return;
63     switch (fv->type) {
64         case C_FIELD_VALUE_TYPE_STRING_
65             VALUE:
66             free(fv->data.string_val);
67             break;
68         case C_FIELD_VALUE_TYPE_POINTER_
69             VALUE:
70             free_c_field_list(fv->data.
71                 array_values);
72             break;
73         case C_FIELD_VALUE_TYPE_MAP_
74             VALUE:
75             free_c_map_pair_list(fv->
76                 data.map_pairs);
77             break;
78         case C_FIELD_VALUE_TYPE_OBJECT_
79             VALUE:
80             free_c_object(fv->data.
81                 object_val);
82             break;
83         default:
84             break;
85     }
86     free(fv);
87 }
88 void free_c_field_list(c_field_t *head)
89 {
90     c_field_t *current = head;
91     c_field_t *next;
92     while (current != NULL) {
93         next = current->next;
94         free(current->name);
95         free_c_field_value(current->
96             value);
97         free(current);
98         current = next;
99     }
100 }
101 void free_c_object(c_object_t *obj) {
102     if (!obj) return;
103     free(obj->object_id);
104     free(obj->table_name);
105     // THE FIX: Check if fields is NULL
106     // before freeing
107     if (obj->fields) {
108         free_c_field_list(obj->fields);
109     }
110     free(obj);
111 }
112 // --- C Value Creation Helper Functions
113
114 --- C Field Value Creation Helper Functions
115
116 c_field_value_t* create_c_string_value(
117     const char* val) {
118     c_field_value_t* fv = (c_field_value_
119         _t*)malloc(sizeof(c_field_value_
120             _t));
121     if (!fv) return NULL;
122     fv->type = C_FIELD_VALUE_TYPE_STRING_
123         VALUE;
124     fv->data.string_val = val ? ansi_
125         strdup(val) : NULL;
126     if (val && !fv->data.string_val) {
127         free(fv); return NULL; }
128     return fv;
129 }
130
131 c_field_value_t* create_c_int_value(
132     int64_t val) {
133     c_field_value_t* fv = (c_field_value_
134         _t*)malloc(sizeof(c_field_value_
135             _t));
136     if (!fv) return NULL;
137     fv->type = C_FIELD_VALUE_TYPE_INT_
138         VALUE;
139     fv->data.int_val = val;
140     return fv;
141 }
142
143 c_field_value_t* create_c_bool_value(
144     bool val) {
145     c_field_value_t* fv = (c_field_value_
146         _t*)malloc(sizeof(c_field_value_
147             _t));
148     if (!fv) return NULL;
149     fv->type = C_FIELD_VALUE_TYPE_BOOL_
150         VALUE;
151     fv->data.bool_val = val;
152     return fv;
153 }
154
155 c_field_value_t* create_c_float_value(
156     float val) {
157     c_field_value_t* fv = (c_field_value_
158         _t*)malloc(sizeof(c_field_value_
159             _t));
160     if (!fv) return NULL;
161     fv->type = C_FIELD_VALUE_TYPE_FLOAT_
162         VALUE;
163     fv->data.float_val = val;
164     return fv;
165 }
166
167 c_field_value_t* create_c_pointer_value(
168     const char* object_id) {
169     c_field_value_t* fv = (c_field_value_
170         _t*)malloc(sizeof(c_field_value_
171             _t));
172     if (!fv) return NULL;
173     fv->type = C_FIELD_VALUE_TYPE_POINTER_
174         VALUE;
175     fv->data.string_val = object_id ?
176         ansi_strdup(object_id) : NULL;
177     if (object_id && !fv->data.string_
178         val) { free(fv); return NULL; }
179     return fv;
180 }
181
182 c_field_value_t* create_c_none_value() {
183     c_field_value_t* fv = (c_field_value_
184         _t*)malloc(sizeof(c_field_value_
185             _t));
186     if (!fv) return NULL;
187     fv->type = C_FIELD_VALUE_TYPE_NONE_
188         VALUE;
189     return fv;
190 }

```

```

150     t));
151     if (!fv) return NULL;
152     fv->type = C_FIELD_VALUE_TYPE_NONE;
153     return fv;
154 }
155 // --- Serialization Implementation ---
156 static int create_fb_field_value_union(
157     flatcc_builder_t *B, const c_field_
158     value_t *fv, objstore_FieldValue_
159     union_ref_t *out_union_ref) {
160     if (!fv || fv->type == C_FIELD_VALUE_
161     _TYPE_NONE) {
162         *out_union_ref = objstore_
163             FieldValue_as_NONE();
164         return 0;
165     }
166     switch (fv->type) {
167         case C_FIELD_VALUE_TYPE_INT_
168             VALUE: {
169             objstore_IntValue_ref_t ref
170                 = objstore_IntValue_
171                     create(B, fv->data.int_
172                     val);
173             *out_union_ref = objstore_
174                 FieldValue_as_IntValue(
175                     ref);
176             break;
177         }
178         case C_FIELD_VALUE_TYPE_BOOL_
179             VALUE: {
180             objstore_Boolean_ref_t ref
181                 = objstore_Boolean_
182                     create(B, fv->data.bool_
183                     val);
184             *out_union_ref = objstore_
185                 FieldValue_as_Boolean(
186                     ref);
187             break;
188         }
189         case C_FIELD_VALUE_TYPE_FLOAT_
190             VALUE: {
191             objstore_FloatValue_ref_t ref
192                 = objstore_FloatValue_
193                     create(B, fv
194                         ->data.float_val);
195             *out_union_ref = objstore_
196                 FieldValue_as_FloatValue(
197                     ref);
198             break;
199         }
200         case C_FIELD_VALUE_TYPE_STRING_
201             VALUE: {
202             flatbuffers_string_ref_t s =
203                 flatcc_builder_create_
204                     string_str(B, fv->data.
205                     string_val);
206             objstore_StringValue_ref_t ref
207                 = objstore_
208                     StringValue_create(B, s);
209             *out_union_ref = objstore_
210                 FieldValue_as_
211                     StringValue(ref);
212             break;
213         }
214         case C_FIELD_VALUE_TYPE_POINTER_
215             VALUE: {
216             flatbuffers_string_ref_t s =
217                 flatcc_builder_create_
218                     string_str(B, fv->data.
219                     string_val);
220             objstore_PointerValue_ref_t
221                 ref = objstore_
222                     PointerValue_create(B, s);
223             *out_union_ref = objstore_
224                 FieldValue_as_
225                     PointerValue(ref);
226             break;
227         }
228     }
229     flatcc_builder_create_
230         string_str(B, fv->data.
231         string_val);
232     objstore_PointerValue_ref_t
233         ref = objstore_
234             PointerValue_create(B, s);
235     *out_union_ref = objstore_
236         FieldValue_as_
237             PointerValue(ref);
238     break;
239 }
240 case C_FIELD_VALUE_TYPE_OBJECT_
241     VALUE: {
242     objstore_Object_ref_t obj_
243         ref = create_fb_object_
244             recursive(B, fv->data.
245             object_val);
246     objstore_ObjectValue_ref_t
247         ref = objstore_
248             ObjectValue_create(B,
249                 obj_ref);
250     *out_union_ref = objstore_
251         FieldValue_as_
252             ObjectValue(ref);
253     break;
254 }
255 case C_FIELD_VALUE_TYPE_ARRAY_
256     VALUE: {
257     c_field_t *p = fv->data.
258         array_values;
259     objstore_Field_vec_start(B);
260     while(p) {
261         objstore_Field_ref_t
262             field_ref = create_
263                 fb_field(B, p);
264         objstore_Field_vec_push(
265             B, field_ref);
266         p = p->next;
267     }
268     flatbuffers_vec_ref_t vec =
269         objstore_Field_vec_end(B
270             );
271     objstore_ArrayValue_ref_t
272         ref = objstore_
273             ArrayValue_create(B, vec
274                 );
275     *out_union_ref = objstore_
276         FieldValue_as_ArrayValue(
277             ref);
278     break;
279 }
280 case C_FIELD_VALUE_TYPE_MAP_
281     VALUE: {
282     c_map_pair_t *p = fv->data.
283         map_pairs;
284     objstore_MapPair_vec_start(B
285             );
286     while(p) {
287         flatbuffers_string_ref_t
288             key = flatcc_
289                 builder_create_
290                     string_str(B, p->key
291                         );
292         objstore_FieldValue_
293             union_ref_t val;
294         create_fb_field_value_
295             union(B, p->value, &
296                 val);
297         objstore_MapPair_ref_t
298             ref = objstore_
299                 MapPair_create(B, key,
300                     val);
301         objstore_MapPair_vec_push(
302             B, ref);
303         p = p->next;
304     }
305     flatcc_builder_create_
306         string_str(B, fv->data.
307         string_val);
308     objstore_MapValue_ref_t
309         ref = objstore_
310             MapValue_create(B, vec
311                 );
312     *out_union_ref = objstore_
313         FieldValue_as_MapValue(
314             ref);
315     break;
316 }

```

```

    pair_ref = objstore_
      MapPair_create(B,
      key, val);
    objstore_MapPair_vec_
      push(B, pair_ref);
    p = p->next;
  }
  flatbuffers_vec_ref_t vec =
    objstore_MapPair_vec_end
    (B);
  objstore_MapValue_ref_t ref
    = objstore_MapValue_
      create(B, vec);
  *out_union_ref = objstore_
    FieldValue_as_MapValue(
    ref);
  break;
}
default: return -1;
}
return 0;
}

static objstore_Field_ref_t create_fb_
field(flatcc_builder_t *B, const c_
field_t *cf) {
if (!cf) return 0;
flatbuffers_string_ref_t name =
  flatcc_builder_create_string_str
  (B, cf->name);
objstoreFieldValue_union_ref_t
  value_union;
create_fb_field_value_union(B, cf->
  value, &value_union);
return objstore_Field_create(B, name
  , value_union);
}

// Replace the old create_fb_object_
// recursive with this robust version
static objstoreObject_ref_t create_fb_
object_recursive(flatcc_builder_t *B
, const c_object_t *obj) {
if (!obj) return 0;

// 1. Build all nested parts first (
//     strings, vectors).
flatbuffers_string_ref_t id_ref =
  flatcc_builder_create_string_str
  (B, obj->object_id);
flatbuffers_string_ref_t table_ref =
  flatcc_builder_create_string_
  str(B, obj->table_name);

flatbuffers_vec_ref_t fields_vec =
  {};
c_field_t *p = obj->fields;
if (p) {
  objstoreField_vec_start(B);
  while(p) {
    // create_fb_field is safe
    // to call recursively.
    objstoreField_vec_push(B,
      create_fb_field(B, p));
    p = p->next;
  }
  fields_vec = objstoreField_vec_
    end(B);
}
}

pair_ref = objstore_
  MapPair_create(B,
  key, val);
  objstore_MapPair_vec_
    push(B, pair_ref);
  p = p->next;
}
flatbuffers_vec_ref_t vec =
  objstore_MapPair_vec_end
  (B);
objstore_MapValue_ref_t ref
  = objstore_MapValue_
    create(B, vec);
*out_union_ref = objstore_
  FieldValue_as_MapValue(
  ref);
break;
}
default: return -1;
}
return 0;
}

fb_serializer_status_t serialize_c_
object(const c_object_t *obj, void
**out_buffer, size_t *out_size) {
flatcc_builder_t builder;
if (!obj || !out_buffer || !out_size)
  return FB_SERIALIZER_ERROR_
  INVALID_INPUT;

flatcc_builder_init(&builder);

// This is the only guaranteed
// correct and safe way to build
// the object
// without risking state corruption
// that leads to silent data loss.
*out_buffer = 0;
*out_size = 0;
flatcc_builder_reset(&builder);

// Manually build the root object
// from primitives.
flatbuffers_string_ref_t id_ref =
  flatcc_builder_create_string_str
  (&builder, obj->object_id);
flatbuffers_string_ref_t table_ref =
  flatcc_builder_create_string_
  str(&builder, obj->table_name);

flatbuffers_vec_ref_t fields_vec =
  {};
c_field_t *p = obj->fields;
if (p) {
  objstoreField_vec_start(&
    builder);
  while (p) {
    objstoreField_vec_push(&
      builder, create_fb_field
      (&builder, p));
    p = p->next;
  }
  fields_vec = objstoreField_vec_
    end(&builder);
}

objstoreObject_start_as_root(&
  builder);
objstoreObject_object_id_add(&
  builder, id_ref);
objstoreObject_table_name_add(&
  builder, table_ref);
if (fields_vec) {
}
}

```

```

301     objstore_Object_fields_add(&
302         builder, fields_vec);
303     objstore_Object_end_as_root(&builder
304         );
305     *out_buffer = flatcc_builder_
306         finalize_aligned_buffer(&builder
307             , out_size);
308     flatcc_builder_clear(&builder);
309     if (!*out_buffer) {
310         return FB_SERIALIZER_ERROR_
311             ALLOCATION;
312     }
313     return FB_SERIALIZER_OK;
314 // --- Deserialization Implementation
315 /**
316  * @brief Parses a flatbuffers union field value.
317  * @param[in] fv The input flatbuffers union field value.
318  * @param[in] fb_union_struct The flatbuffers union struct.
319  * @param[in] type_tag The type tag of the union.
320  * @param[out] out_fv The output c_field_value_t pointer.
321  */
322 static fb_serializer_status_t parse_fb_
323     fieldValue_union(
324         objstore_FieldValue_union_t fb_union_
325             _struct,
326         flatbuffers_union_type_t fb_union_
327             type_tag,
328         c_field_value_t **out_fv) {
329
330     *out_fv = (c_field_value_t*)calloc
331         (1, sizeof(c_field_value_t));
332     if (!*out_fv) return FB_SERIALIZER_
333         ERROR_ALLOCATION;
334
335     (*out_fv)->type = (c_field_value_
336         _type_t)fb_union_type_tag;
337     const void* fb_union_table_ptr = fb_
338         union_struct.value;
339
340     objstore_Object_table_t nested_fb_
341         obj_temp;
342     fb_serializer_status_t status_temp;
343     c_field_t *head_field = NULL, *tail_
344         field = NULL;
345     c_map_pair_t *head_map_pair = NULL,
346         *tail_map_pair = NULL;
347
348     if (fb_union_type_tag != objstore_
349         FieldValue_NONE && !fb_union_
350         table_ptr) {
351         free(*out_fv); *out_fv = NULL;
352         return FB_SERIALIZER_ERROR_
353             INVALID_INPUT;
354     }
355
356     switch (fb_union_type_tag) {
357         case objstore_FieldValue_NONE:
358             break;
359         case objstore_FieldValue_
360             IntValue:
361             (*out_fv)->data.int_val =
362                 objstore_IntValue_value
363                 ((objstore_IntValue_
364                     table_t)fb_union_table_
365                     ptr); break;
366         case objstore_FieldValue_
367             BoolValue:
368             (*out_fv)->data.bool_val =
369                 objstore_BooleanValue_value
370                 ((objstore_BooleanValue_
371                     table_t)fb_union_table_
372                     ptr); break;
373     }
374
375     table_t)fb_union_table_
376         ptr); break;
377     case objstore_FieldValue_
378         FloatValue:
379         (*out_fv)->data.float_val =
380             objstore_FloatValue_
381                 value((objstore_
382                     FloatValue_table_t)fb_
383                     union_table_ptr); break;
384     case objstore_FieldValue_
385         StringValue:
386         (*out_fv)->data.string_val =
387             ansi_strdup(objstore_
388                 StringValue_value((
389                     objstore_StringValue_
390                         table_t)fb_union_table_
391                             ptr));
392         if (!(*out_fv)->data.string_
393             val) { free(*out_fv); *
394             out_fv = NULL; return FB_
395                 SERIALIZER_ERROR_
396                     ALLOCATION; } break;
397     case objstore_FieldValue_
398         PointerValue:
399         (*out_fv)->data.string_val =
400             ansi_strdup(objstore_
401                 PointerValue_object_id((
402                     objstore_PointerValue_
403                         table_t)fb_union_table_
404                             ptr));
405         if (!(*out_fv)->data.string_
406             val) { free(*out_fv); *
407             out_fv = NULL; return FB_
408                 SERIALIZER_ERROR_
409                     ALLOCATION; } break;
410     case objstore_FieldValue_
411         ObjectValue:
412         nested_fb_obj_temp =
413             objstore_ObjectValue_
414                 object((objstore_
415                     ObjectValue_table_t)fb_
416                         union_table_ptr);
417         if (nested_fb_obj_temp) {
418             status_temp = parse_fb_
419                 object_recursive(
420                     nested_fb_obj_temp,
421                     &(*out_fv)->data.
422                         object_val));
423             if (status_temp != FB_
424                 SERIALIZER_OK) {
425                 free(*out_fv); *out_
426                     fv = NULL; return
427                         status_temp; }
428         }
429         break;
430     case objstore_FieldValue_
431         ArrayValue:
432     {
433         objstore_Field_vec_t vec =
434             objstore_ArrayValue_
435                 values((objstore_
436                     ArrayValue_table_t)fb_
437                         union_table_ptr);
438         if (vec) {
439             for (size_t i = 0; i <
440                 objstore_Field_vec_
441                     len(vec); ++i) {
442                 c_field_t *new_item
443                     = NULL;
444                 status_temp = parse_

```

```

364     fb_field(
365         objstore_Field_
366             vec_at(vec, i),
367             &new_item);
368     if (status_temp != FB_SERIALIZER_OK)
369     ) { free_c_field
370         _list(head_field
371             ); free(*out_fv)
372             ; *out_fv = NULL
373             ; return status_
374                 temp; }
375     if (!head_field)
376         head_field =
377             tail_field = new
378                 _item;
379     else { tail_field->
380             next = new_item;
381             tail_field =
382                 new_item; }
383     }
384     (*out_fv)->data.array_values
385         = head_field;
386     break;
387 }
388 case objstoreFieldValue_
389 MapValue:
{ objstore_MapPair_vec_t vec =
390     objstore_MapValue_pairs
391     ((objstore_MapValue_
392         table_t)fb_union_table_
393         ptr);
394     if (vec) {
395         for (size_t i = 0; i <
396             objstore_MapPair_vec_
397                 _len(vec); ++i) {
398             objstore_MapPair_
399                 table_t pair_
400                     table = objstore_
401                         _MapPair_vec_at(
402                             vec, i);
403             c_map_pair_t* new_
404                 pair = (c_map_
405                     pair_t*)calloc
406                         (1, sizeof(c_map_
407                             pair_t));
408             if (!new_pair) {
409                 free_c_map_pair_
410                     list(head_map_
411                         pair); free(*out_
412                             fv); *out_fv =
413                                 NULL; return FB_
414                                     SERIALIZER_ERROR_
415                                         _ALLOCATION; }
416             new_pair->key = ansi_
417                 strdup(objstore_
418                     _MapPair_key(
419                         pair_table));
420             if (!new_pair->key)
421                 { free(new_pair)
422                     ; free_c_map_
423                         pair_list(head_
424                             map_pair); free
425                                 (*out_fv); *out_
426                                     fv = NULL;
427                                     return FB_
428                                         SERIALIZER_ERROR_
429                                         _ALLOCATION; }
430             }
431             (*out_fv)->data.map_pairs =
432                 head_map_pair;
433             break;
434         }
435         default: free(*out_fv); *out_fv
436             = NULL; return FB_SERIALIZER_
437                 _ERROR_TYPE_MISMATCH;
438     }
439     return FB_SERIALIZER_OK;
440 }
441 static fb_serializer_status_t parse_fb_
442     field(objstore_Field_table_t fb_
443         field_table, c_field_t **out_c_field
444             ) {
445     flatbuffers_union_type_t val_type_
446         tag;
447     objstoreFieldValue_union_t val_
448         union_struct;
449     fb_serializer_status_t status;
450     *out_c_field = (c_field_t*)calloc(1,
451         sizeof(c_field_t));
452     if (!*out_c_field) return FB_
453         SERIALIZER_ERROR_ALLOCATION;
454     (*out_c_field)->name = ansi_strdup(
455         objstore_Field_name(fb_field_
456             table));
457     if (!(*out_c_field)->name) { free(*
458         out_c_field); *out_c_field =
459             NULL; return FB_SERIALIZER_ERROR_
460                 _ALLOCATION; }
461     val_union_struct = objstore_Field_

```

```

    value_union(fb_field_table);
412  val_type_tag = val_union_struct.type
413  ;
414
415  status = parse_fb_field_value_union(
416      val_union_struct, val_type_tag,
417      &(*out_c_field)->value));
418  if (status != FB_SERIALIZER_OK) {
419      free((*out_c_field)->name); free
420      (*out_c_field); *out_c_field =
421      NULL; return status; }
422
423  return FB_SERIALIZER_OK;
424 }
425
426 static fb_serializer_status_t parse_fb_
427 object_recursive(objstore_Object_
428 table_t fb_obj_table, c_object_t **
429 out_obj_param) {
430     c_field_t *head = NULL, *tail = NULL
431     ;
432     fb_serializer_status_t status;
433
434     if (!fb_obj_table) return FB_
435         SERIALIZER_ERROR_INVALID_INPUT;
436
437     *out_obj_param = (c_object_t*)calloc
438         (1, sizeof(c_object_t));
439     if (!*out_obj_param) return FB_
440         SERIALIZER_ERROR_ALLOCATION;
441
442     (*out_obj_param)->object_id = ansi_
443         strdup(objstore_Object_object_id
444         (fb_obj_table));
445     (*out_obj_param)->table_name = ansi_
446         strdup(objstore_Object_table_
447         name(fb_obj_table));
448
449     if (!(*out_obj_param)->object_id || !
450         (*out_obj_param)->table_name) {
451         free_c_object(*out_obj_param); *
452         out_obj_param = NULL; return
453             FB_SERIALIZER_ERROR_
454                 ALLOCATION;
455     }
456
457     objstore_Field_vec_t fields_vec =
458         objstore_Object_fields(fb_obj_
459         table);
460     if (fields_vec) {
461         for (size_t i = 0; i < objstore_
462             Field_vec_len(fields_vec);
463             ++i) {
464             c_field_t *new_item = NULL;
465             status = parse_fb_field(
466                 objstore_Field_vec_at(
467                     fields_vec, i), &new_
468                     item);
469             if (status != FB_SERIALIZER_
470                 OK) { free_c_field_list(
471                     head); free_c_object(*
472                     out_obj_param); *out_obj_
473                     _param = NULL; return
474                     status; }
475             if (new_item) {
476                 if (!head) head = tail =
477                     new_item;
478                 else { tail->next = new_
479                     item; tail = new_
480                     item; }
481             }
482         }
483     }
484
485     (*out_obj_param)->fields = head;
486     return FB_SERIALIZER_OK;
487 }
488
489 fb_serializer_status_t deserialize_to_c_
490 object(const void *buffer, size_t
491 size, c_object_t **out_obj) {
492     if (!buffer || size == 0 || !out_obj
493         ) {
494         return FB_SERIALIZER_ERROR_
495             INVALID_INPUT;
496     }
497
498     int verify_ret = objstore_Object_
499         verify_as_root_with_identifier(
500             buffer, size, 0);
501     if (verify_ret != flatcc_verify_ok)
502     {
503         fprintf(stderr, "FlatBuffer_
504             verification failed: %s\n",
505             flatcc_verify_error_string(
506                 verify_ret));
507         return FB_SERIALIZER_ERROR_
508             FLATCC_VERIFIER;
509     }
510
511     objstore_Object_table_t root_table =
512         objstore_Object_as_root(buffer)
513         ;
514     if (!root_table) {
515         return FB_SERIALIZER_ERROR_
516             INVALID_INPUT;
517     }
518
519     return parse_fb_object_recursive(
520         root_table, out_obj);
521 }
522
523 // --- JSON Transformation Logic ---
524 static fb_serializer_status_t c_object_
525 from_cjson_node(cJSON *json_node,
526 const char *default_object_id, const
527 char *default_table_name, c_object_
528 t **out_obj) {
529     if (!cJSON_IsObject(json_node))
530         return FB_SERIALIZER_ERROR_TYPE_
531             MISMATCH;
532     *out_obj = NULL;
533
534     c_object_t *obj = (c_object_t*)
535         calloc(1, sizeof(c_object_t));
536     if (!obj) return FB_SERIALIZER_ERROR_
537             _ALLOCATION;
538
539     fb_serializer_status_t status = FB_
540         SERIALIZER_OK;
541     cJSON *json_val;
542
543     json_val = cJSON_
544         GetObjectItemCaseSensitive(json_
545             node, "object_id");
546     if (cJSON_IsString(json_val) && json_
547         _val->valuestring) {
548         obj->object_id = ansi_strdup(
549             json_val->valuestring);
550     } else if (default_object_id) {
551         obj->object_id = ansi_strdup(
552             default_object_id);
553     }

```



```

562     fv->type = C_FIELD_VALUE_
563         TYPE_STRING_VALUE;
564     }
565     fv->data.string_val = ansi_
566         strdup(item->valuestring);
567     if (!fv->data.string_val && item
568         ->valuestring) {
569         free(fv); *current_status =
570             FB_SERIALIZER_ERROR_
571             ALLOCATION; return NULL;
572     }
573 } else if (cJSON_IsArray(item)) {
574     fv->type = C_FIELD_VALUE_TYPE_
575         ARRAY_VALUE;
576     c_field_t *head = NULL, *tail =
577         NULL;
578     cJSON *elem = NULL;
579     int index = 0;
580     cJSON_ArrayForEach(elem, item) {
581         char synthetic_name[32];
582         snprintf(synthetic_name,
583             sizeof(synthetic_name),
584             "item_%d", index++);
585
586         c_field_value_t* item_fv = c_
587             _field_value_from_cjson_
588             recursive(elem,
589                 synthetic_name, current_
590                 status);
591         if (*current_status != FB_
592             SERIALIZER_OK) { free_c_
593                 _field_list(head); free
594                 (fv); return NULL; }
595
596         c_field_t* new_item_field =
597             (c_field_t*)calloc(1,
598                 sizeof(c_field_t));
599         if (!new_item_field) { free_c_
600             _field_value(item_fv);
601             free_c_field_list(head);
602             free(fv); *current_
603                 status = FB_SERIALIZER_
604                 ERROR_ALLOCATION; return
605                 NULL; }
606
607         new_item_field->name = ansi_
608             strdup(synthetic_name);
609         if (!new_item_field->name) {
610             free(new_item_field);
611             free_c_field_value(item_
612                 fv); free_c_field_list(
613                     head); free(fv); *
614                     current_status = FB_
615                     SERIALIZER_ERROR_
616                     ALLOCATION; return NULL
617             ; }
618
619         new_item_field->value = item_
620             _fv;
621
622         if (!head) head = tail = new_
623             _item_field;
624         else { tail->next = new_item_
625             _field; tail = new_item_
626             _field; }
627     }
628     fv->data.array_values = head;
629 } else if (cJSON_IsObject(item)) {
630     cJSON *obj_id_json = cJSON_
631         GetObjectItemCaseSensitive(
632             item, "object_id");
633     cJSON *tbl_name_json = cJSON_
634         GetObjectItemCaseSensitive(
635             item, "table_name");
636
637     if (cJSON_IsString(obj_id_json)
638         && obj_id_json->valuestring
639         &&
640             cJSON_IsString(tbl_name_json
641                 ) && tbl_name_json->
642                 valuestring) {
643         fv->type = C_FIELD_VALUE_
644             TYPE_OBJECT_VALUE;
645         *current_status = c_object_
646             from_cjson_node(item,
647                 NULL, NULL, &fv->data.
648                 object_val);
649         if (*current_status != FB_
650             SERIALIZER_OK) {
651             free(fv); return NULL;
652         }
653     } else {
654         fv->type = C_FIELD_VALUE_
655             TYPE_MAP_VALUE;
656         c_map_pair_t *head = NULL, *
657             tail = NULL;
658         cJSON *pair_item_json = item
659             ->child;
660         while(pair_item_json) {
661             if (!pair_item_json->
662                 string) {
663                 pair_item_json =
664                     pair_item_json->
665                     next;
666                 continue;
667             }
668             c_map_pair_t *new_pair =
669                 (c_map_pair_t*)
670                     calloc(1, sizeof(c_
671                         map_pair_t));
672             if (!new_pair) { free_c_
673                 map_pair_list(head);
674                 free(fv); *current_
675                     status = FB_
676                     SERIALIZER_ERROR_
677                     ALLOCATION; return
678                     NULL; }
679
680             new_pair->key = ansi_
681                 strdup(pair_item_
682                     json->string);
683             if (!new_pair->key) { free
684                 (new_pair); free_c_
685                 map_pair_list(head);
686                 free(fv); *current_
687                     status = FB_
688                     SERIALIZER_ERROR_
689                     ALLOCATION; return
690                     NULL; }
691
692             new_pair->value = c_
693                 _field_value_from_
694                 cjson_recursive(pair_
695                     item_json, pair_
696                     item_json->string,
697                     current_status);
698             if (*current_status != FB_
699                 SERIALIZER_OK) {
700                 free(new_pair->key);
701                 free(new_pair);
702             }
703         }
704     }
705 }

```

```
621     free_c_map_pair
622     list(head);
623     free(fv);
624     return NULL;
625   }
626
627   if (!head) head = tail =
628     new_pair;
629   else { tail->next = new_
630     pair; tail = new_
631     pair; }
632   pair_item_json = pair_
633     item_json->next;
634 }
635 fv->data.map_pairs = head;
636 }
637 }
638
639 fb_serializer_status_t fb_object_from_
640 json_string(const char *json_string,
641   const char *default_object_id,
642   const char *default_table_name,c_
643 object_t **out_c_obj) {
644
645   if (!json_string || !out_c_obj)
646     return FB_SERIALIZER_ERROR_
647     INVALID_INPUT;
648   *out_c_obj = NULL;
649
650   cJSON *root_json = cJSON_Parse(json_
651     string);
652   if (!root_json) {
653     return FB_SERIALIZER_ERROR_JSON_
654     PARSE;
655   }
656   if(!cJSON_IsObject(root_json)){
657     cJSON_Delete(root_json);
658     return FB_SERIALIZER_ERROR_TYPE_
659     MISMATCH;
660   }
661
662   fb_serializer_status_t status = c_
663     object_from_cjson_node(root_json
664     , default_object_id, default_
665     table_name, out_c_obj);
666
667   cJSON_Delete(root_json);
668   return status;
669 }
```

```
ing the database and performing CRUD operations.

#ifndef OBJECT_STORE_H
#define OBJECT_STORE_H

#include "fb_serializer.h" // For status
    enum and c_object_t
#include <lmdb.h>           // For MDB_
    env, MDB_db
#include <stddef.h>
#include <stdint.h>
#include <stdbool.h>

// --- Portability Defines for Threading
---
// Selects the appropriate threading
// primitive based on the operating
// system.
#if defined(_WIN32) && !defined(__CYGWIN__)
    #include <windows.h>
    #define OS_USE_WINDOWS_CRITICAL_
        SECTION
#elif defined(__linux__) || defined(__
    APPLE__) || defined(__FreeBSD__) |||
    defined(__NetBSD__) ||| defined(__
    OpenBSD__) ||| defined(__sun) |||
    defined(__QNXNTO__))
    #include <pthread.h>
    #define OS_USE_PTHREADS
#endif

// =====
// SECTION 1: CONSTANTS
// =====

// The fixed size of a key in the
// database.
// 8 bytes (inverted timestamp) + 16
// bytes (MD5 hash) + 8 bytes (nonce)
#define OS_MAX_KEY_SIZE 32
// Default size for the LMDB memory map
// (1 GiB).
#define OS_DEFAULT_MAPSIZE (1024 * 1024
    * 1024)
// Default maximum number of named
// databases (tables) within the
// environment.
#define OS_DEFAULT_MAX_DBs 128

// =====
// SECTION 2: CORE STRUCTURES
// =====

/***
 * @struct os_table_info_t
 * @brief Caches the name and LMDB
 *         handle (DBI) of an opened table.
 */
typedef struct {
    char* name;
    MDB_db dbi;
    bool valid;
} os_table_info_t;

/***
 * @struct object_store_t
 * @brief The main handle for an
 */
```

```

    instance of the object store.
51 * This struct holds the LMDB
   environment, a cache of table
   handles, and
52 * a mutex to protect concurrent access
   to the cache.
53 */
54 typedef struct object_store_t {
55     MDB_env *env;
56     os_table_info_t *tables;
57     size_t num_tables;
58     size_t tables_capacity;
59
60 #if defined(OS_USE_PTHREADS)
61     pthread_mutex_t table_cache_mutex;
62 #elif defined(OS_USE_WINDOWS_CRITICAL_
   SECTION)
63     CRITICAL_SECTION table_cache_mutex;
64 #endif
65 } object_store_t;
66
67 // =====
68 // SECTION 3: LIFECYCLE AND
69 // INITIALIZATION API
70 // =====
71
72 /**
73 * @brief Creates and initializes a new
   object_store_t handle.
74 * @param store_ptr A pointer to an
   object_store_t pointer that will be
   allocated.
75 * @return 0 on success, -1 on failure.
76 */
77 int object_store_create(object_store_t
   **store_ptr);
78
79 /**
80 * @brief Closes the database and frees
   all resources associated with an
   object store handle.
81 * @param store_ptr A pointer to the
   object_store_t pointer to be freed
   and set to NULL.
82 */
83 void object_store_destroy(object_store_t
   **store_ptr);
84
85 /**
86 * @brief Initializes the LMDB
   environment for the object store.
87 * @param store The object store handle.
88 * @param path The filesystem path where
   the database will be stored.
89 * @param mapsize The maximum size of
   the database file. Pass 0 for
   default.
90 * @param maxdbs The maximum number of
   tables. Pass 0 for default.
91 * @return FB_SERIALIZER_OK on success,
   or an error code on failure.
92 */
93 fb_serializer_status_t object_store_init
   _db(object_store_t *store, const
   char *path, size_t mapsize, unsigned
   int maxdbs);
94
95 // =====
96
97 // =====
98 // SECTION 4: TABLE AND
99 // KEY MANAGEMENT API
100 // =====
101
102 /**
103 * @brief Creates a new table (a named
   database) within the object store.
104 * @param store The object store handle.
105 * @param name The name of the table to
   create.
106 * @return FB_SERIALIZER_OK on success,
   or an error code.
107 */
108 fb_serializer_status_t object_store_
   create_table(object_store_t *store,
   const char *name);
109
110 /**
111 * @brief Deletes a table and all
   objects within it.
112 * @param store The object store handle.
113 * @param name The name of the table to
   delete.
114 * @return FB_SERIALIZER_OK on success,
   or an error code.
115 */
116 fb_serializer_status_t object_store_
   delete_table(object_store_t *store,
   const char *name);
117
118 /**
119 * @brief Generates a new, unique, time-
   sortable key.
120 * @param key_buffer A buffer of size OS
   _MAX_KEY_SIZE to hold the generated
   key.
121 * @return FB_SERIALIZER_OK on success.
122 */
123 fb_serializer_status_t object_store_
   generate_key(unsigned char key_
   buffer[OS_MAX_KEY_SIZE]);
124
125 // =====
126 // SECTION 5: OBJECT CRUD (CREATE,
127 // READ, UPDATE, DELETE) API
128 // =====
129
130 /**
131 * @brief Creates a new object in the
   database. Fails if the key already
   exists.
132 * @param store The object store handle.
133 * @param table_name The name of the
   table to insert into.
134 * @param key The unique key for the
   object.
135 * @param value The C object to
   serialize and store.
136 * @return FB_SERIALIZER_OK on success,
   FB_SERIALIZER_ERROR_LMDB_KEY_EXISTS
   if the key exists, or another
   error code.
137 */
138 fb_serializer_status_t object_store_
   create_object(object_store_t *store,
   const char *table_name, const
   unsigned char key[OS_MAX_KEY_SIZE],
   const c_object_t *value);
139
140

```

```

141 /**
142 * @brief Puts an object into the
143 * database. Creates it if it doesn't
144 * exist, or overwrites it if it does.
145 * @param store The object store handle.
146 * @param table_name The name of the
147 * table.
148 * @param key The unique key for the
149 * object.
150 * @param value The C object to
151 * serialize and store.
152 * @return FB_SERIALIZER_OK on success,
153 * or an error code.
154 */
155 fb_serializer_status_t object_store_put_
156     object(object_store_t *store, const
157         char *table_name, const unsigned
158         char key[OS_MAX_KEY_SIZE], const c_
159         object_t *value);
160
161 /**
162 * @brief Retrieves and deserializes an
163 * object from the database.
164 * @param store The object store handle.
165 * @param table_name The name of the
166 * table.
167 * @param key The key of the object to
168 * retrieve.
169 * @param out_value A pointer to a c_
170 * object_t pointer that will receive
171 * the new, deserialized object. The
172 * caller must free this with free_c_
173 * object().
174 * @return FB_SERIALIZER_OK on success,
175 * FB_SERIALIZER_ERROR_LMDB_NOT_FOUND
176 * if not found, or another error code
177 */
178 fb_serializer_status_t object_store_get_
179     object(object_store_t *store, const
180         char *table_name, const unsigned
181         char key[OS_MAX_KEY_SIZE], c_object_
182         t **out_value);
183
184 /**
185 * @brief Deletes an object from the
186 * database.
187 * @param store The object store handle.
188 * @param table_name The name of the
189 * table.
190 * @param key The key of the object to
191 * delete.
192 * @return FB_SERIALIZER_OK on success,
193 * FB_SERIALIZER_ERROR_LMDB_NOT_FOUND
194 * if the key did not exist.
195 */
196 fb_serializer_status_t object_store_
197     delete_object(object_store_t *store,
198         const char *table_name, const
199         unsigned char key[OS_MAX_KEY_SIZE]);
200
201 /**
202 * @brief Parses a JSON string and puts
203 * the resulting object into the
204 * database.
205 * @param store The object store handle.
206 * @param table_name The name of the
207 * table.
208 * @param key The unique key for the
209 * object.
210 */
211 fb_serializer_status_t object_store_put_
212     object_from_json(
213         object_store_t *store,
214         const char *table_name,
215         const unsigned char key[OS_MAX_KEY_
216             SIZE],
217         const char *object_json_string);
218
219 #endif // OBJECT_STORE_H

```

Listing 17: The public API for the Object Store database  
(src/object\_store/object\_store.h).

The implementation in ‘object\_store.c’ handles the practical details of file I/O and concurrency. It manages the LMDB environment handle and uses a mutex-protected cache for table handles (‘MDB\_dbi’) to prevent the performance penalty of re-opening tables for every transaction. All database write operations (‘put’, ‘create’, ‘delete’) are wrapped in LMDB transactions, ensuring that the agent’s knowledge base remains in a consistent state at all times.

```

1 // object_store.c
2 // Core implementation of the persistent
3 // object storage layer.
4
5 // ***FIX**: Define _POSIX_C_SOURCE to
6 // get access to clock_gettime on POSIX
7 // systems.
8 // This must be the very first thing in
9 // the file, before any #includes.
10 #if defined(_linux_) || defined(
11     _APPLE_) || defined(_FreeBSD_) || defined(
12     _NetBSD_) || defined(_OpenBSD_) || defined(_sun) ||
13     defined(_QNXNT0_)
14     #ifndef _POSIX_C_SOURCE
15         #define _POSIX_C_SOURCE 200809L
16     #endif
17 #endif
18
19 #include "object_store.h"
20 #include "fb_serializer.h"
21 #include "md5.h"
22
23 #include <stdio.h>
24 #include <stdlib.h>
25 #include <string.h>
26 #include <time.h>
27 #include <errno.h>
28
29 #if defined(_WIN32) && !defined(__CYGWIN__)
30     #include <windows.h>
31     #include <wincrypt.h>
32     #pragma comment(lib, "Advapi32.lib")
33     #include <direct.h>
34
35 #endif

```

```

27 #else // POSIX-like
28     #include <sys/stat.h>
29     #include <sys/types.h>
30     #include <fcntl.h>
31     #include <unistd.h>
32 #endif
33
34 // --- Local String Duplication ---
35 static char* os_strdup(const char* s) {
36     if (!s) return NULL;
37     size_t len = strlen(s) + 1;
38     char* new_s = (char*)malloc(len);
39     if (!new_s) {
40         perror("ERROR: os_strdup - malloc failed");
41         return NULL;
42     }
43     memcpy(new_s, s, len);
44     return new_s;
45 }
46
47 // --- Helper Function for LMDB Errors
48 /**
49 * @param func_name - function name
50 * @param rc - error code
51 */
52 static void print_lmdb_error(const char* func_name, int rc) {
53     fprintf(stderr, "LMDB_ERROR in %s: %s (%d)\n",
54             func_name, mdb_strerror(rc), rc);
55 }
56
57 // --- Table DBI Management (Thread-Safe
58 // Cache Access) ---
59 static fb_serializer_status_t get_table_
60 dbi(object_store_t *store, const
61 char *name, MDB_db *dbi_out, bool
62 create_if_not_exists) {
63     if (!store || !store->env || !name
64     || !dbi_out) return FB_
65     SERIALIZER_ERROR_INVALID_INPUT;
66
67 #if defined(OS_USE_PTHREADS)
68     pthread_mutex_lock(&store->table_
69     cache_mutex);
70 #elif defined(OS_USE_WINDOWS_CRITICAL_
71 SECTION)
72     EnterCriticalSection(&store->table_
73     cache_mutex);
74 #endif
75
76     for (size_t i = 0; i < store->num_
77     tables; ++i) {
78         if (store->tables[i].valid &&
79             strcmp(store->tables[i].name
80             , name) == 0) {
81             *dbi_out = store->tables[i].
82             dbi;
83
84 #if defined(OS_USE_PTHREADS)
85             pthread_mutex_unlock(&
86                     store->table_cache_
87                     mutex);
88 #elif defined(OS_USE_WINDOWS_
89             CRITICAL_SECTION)
90             LeaveCriticalSection(&
91                     store->table_cache_
92                     mutex);
93 #endif
94
95         if (store->num_tables >= store->
96             tables_capacity) {
97             size_t new_capacity = (store->
98                 tables_capacity == 0) ? 16 :
99
100 #if defined(OS_USE_PTHREADS)
101             pthread_mutex_lock(&store->table_
102             cache_mutex);
103 #elif defined(OS_USE_WINDOWS_CRITICAL_
104 SECTION)
105             EnterCriticalSection(&store->table_
106             cache_mutex);
107 #endif
108
109         for (size_t i = 0; i < store->num_
110         tables; ++i) {
111             if (store->tables[i].valid &&
112                 strcmp(store->tables[i].name
113                 , name) == 0) {
114                 goto get_table_dbi_cleanup;
115             }
116         }
117
118         if (store->num_tables >= store->
119             tables_capacity) {
120             size_t new_capacity = (store->
121                 tables_capacity == 0) ? 16 :
122
123 #if defined(OS_USE_PTHREADS)
124             pthread_mutex_lock(&store->table_
125             cache_mutex);
126 #elif defined(OS_USE_WINDOWS_CRITICAL_
127 SECTION)
128             EnterCriticalSection(&store->table_
129             cache_mutex);
130 #endif
131
132         for (size_t i = 0; i < store->num_
133         tables; ++i) {
134             if (store->tables[i].valid &&
135                 strcmp(store->tables[i].name
136                 , name) == 0) {
137                 goto get_table_dbi_cleanup;
138             }
139         }
140
141         if (store->num_tables >= store->
142             tables_capacity) {
143             size_t new_capacity = (store->
144                 tables_capacity == 0) ? 16 :
145
146 #if defined(OS_USE_PTHREADS)
147             pthread_mutex_lock(&store->table_
148             cache_mutex);
149 #elif defined(OS_USE_WINDOWS_CRITICAL_
150 SECTION)
151             EnterCriticalSection(&store->table_
152             cache_mutex);
153 #endif
154
155         for (size_t i = 0; i < store->num_
156         tables; ++i) {
157             if (store->tables[i].valid &&
158                 strcmp(store->tables[i].name
159                 , name) == 0) {
160                 goto get_table_dbi_cleanup;
161             }
162         }
163
164         if (store->num_tables >= store->
165             tables_capacity) {
166             size_t new_capacity = (store->
167                 tables_capacity == 0) ? 16 :
168
169 #if defined(OS_USE_PTHREADS)
170             pthread_mutex_lock(&store->table_
171             cache_mutex);
172 #elif defined(OS_USE_WINDOWS_CRITICAL_
173 SECTION)
174             EnterCriticalSection(&store->table_
175             cache_mutex);
176 #endif
177
178         for (size_t i = 0; i < store->num_
179         tables; ++i) {
180             if (store->tables[i].valid &&
181                 strcmp(store->tables[i].name
182                 , name) == 0) {
183                 goto get_table_dbi_cleanup;
184             }
185         }
186
187         if (store->num_tables >= store->
188             tables_capacity) {
189             size_t new_capacity = (store->
190                 tables_capacity == 0) ? 16 :
191
192 #if defined(OS_USE_PTHREADS)
193             pthread_mutex_lock(&store->table_
194             cache_mutex);
195 #elif defined(OS_USE_WINDOWS_CRITICAL_
196 SECTION)
197             EnterCriticalSection(&store->table_
198             cache_mutex);
199 #endif
200
201         for (size_t i = 0; i < store->num_
202         tables; ++i) {
203             if (store->tables[i].valid &&
204                 strcmp(store->tables[i].name
205                 , name) == 0) {
206                 goto get_table_dbi_cleanup;
207             }
208         }
209
210         if (store->num_tables >= store->
211             tables_capacity) {
212             size_t new_capacity = (store->
213                 tables_capacity == 0) ? 16 :
214
215 #if defined(OS_USE_PTHREADS)
216             pthread_mutex_lock(&store->table_
217             cache_mutex);
218 #elif defined(OS_USE_WINDOWS_CRITICAL_
219 SECTION)
220             EnterCriticalSection(&store->table_
221             cache_mutex);
222 #endif
223
224         for (size_t i = 0; i < store->num_
225         tables; ++i) {
226             if (store->tables[i].valid &&
227                 strcmp(store->tables[i].name
228                 , name) == 0) {
229                 goto get_table_dbi_cleanup;
230             }
231         }
232
233         if (store->num_tables >= store->
234             tables_capacity) {
235             size_t new_capacity = (store->
236                 tables_capacity == 0) ? 16 :
237
238 #if defined(OS_USE_PTHREADS)
239             pthread_mutex_lock(&store->table_
240             cache_mutex);
241 #elif defined(OS_USE_WINDOWS_CRITICAL_
242 SECTION)
243             EnterCriticalSection(&store->table_
244             cache_mutex);
245 #endif
246
247         for (size_t i = 0; i < store->num_
248         tables; ++i) {
249             if (store->tables[i].valid &&
250                 strcmp(store->tables[i].name
251                 , name) == 0) {
252                 goto get_table_dbi_cleanup;
253             }
254         }
255
256         if (store->num_tables >= store->
257             tables_capacity) {
258             size_t new_capacity = (store->
259                 tables_capacity == 0) ? 16 :
260
261 #if defined(OS_USE_PTHREADS)
262             pthread_mutex_lock(&store->table_
263             cache_mutex);
264 #elif defined(OS_USE_WINDOWS_CRITICAL_
265 SECTION)
266             EnterCriticalSection(&store->table_
267             cache_mutex);
268 #endif
269
270         for (size_t i = 0; i < store->num_
271         tables; ++i) {
272             if (store->tables[i].valid &&
273                 strcmp(store->tables[i].name
274                 , name) == 0) {
275                 goto get_table_dbi_cleanup;
276             }
277         }
278
279         if (store->num_tables >= store->
280             tables_capacity) {
281             size_t new_capacity = (store->
282                 tables_capacity == 0) ? 16 :
283
284 #if defined(OS_USE_PTHREADS)
285             pthread_mutex_lock(&store->table_
286             cache_mutex);
287 #elif defined(OS_USE_WINDOWS_CRITICAL_
288 SECTION)
289             EnterCriticalSection(&store->table_
290             cache_mutex);
291 #endif
292
293         for (size_t i = 0; i < store->num_
294         tables; ++i) {
295             if (store->tables[i].valid &&
296                 strcmp(store->tables[i].name
297                 , name) == 0) {
298                 goto get_table_dbi_cleanup;
299             }
300         }
301
302         if (store->num_tables >= store->
303             tables_capacity) {
304             size_t new_capacity = (store->
305                 tables_capacity == 0) ? 16 :
306
307 #if defined(OS_USE_PTHREADS)
308             pthread_mutex_lock(&store->table_
309             cache_mutex);
310 #elif defined(OS_USE_WINDOWS_CRITICAL_
311 SECTION)
312             EnterCriticalSection(&store->table_
313             cache_mutex);
314 #endif
315
316         for (size_t i = 0; i < store->num_
317         tables; ++i) {
318             if (store->tables[i].valid &&
319                 strcmp(store->tables[i].name
320                 , name) == 0) {
321                 goto get_table_dbi_cleanup;
322             }
323         }
324
325         if (store->num_tables >= store->
326             tables_capacity) {
327             size_t new_capacity = (store->
328                 tables_capacity == 0) ? 16 :
329
330 #if defined(OS_USE_PTHREADS)
331             pthread_mutex_lock(&store->table_
332             cache_mutex);
333 #elif defined(OS_USE_WINDOWS_CRITICAL_
334 SECTION)
335             EnterCriticalSection(&store->table_
336             cache_mutex);
337 #endif
338
339         for (size_t i = 0; i < store->num_
340         tables; ++i) {
341             if (store->tables[i].valid &&
342                 strcmp(store->tables[i].name
343                 , name) == 0) {
344                 goto get_table_dbi_cleanup;
345             }
346         }
347
348         if (store->num_tables >= store->
349             tables_capacity) {
350             size_t new_capacity = (store->
351                 tables_capacity == 0) ? 16 :
352
353 #if defined(OS_USE_PTHREADS)
354             pthread_mutex_lock(&store->table_
355             cache_mutex);
356 #elif defined(OS_USE_WINDOWS_CRITICAL_
357 SECTION)
358             EnterCriticalSection(&store->table_
359             cache_mutex);
360 #endif
361
362         for (size_t i = 0; i < store->num_
363         tables; ++i) {
364             if (store->tables[i].valid &&
365                 strcmp(store->tables[i].name
366                 , name) == 0) {
367                 goto get_table_dbi_cleanup;
368             }
369         }
370
371         if (store->num_tables >= store->
372             tables_capacity) {
373             size_t new_capacity = (store->
374                 tables_capacity == 0) ? 16 :
375
376 #if defined(OS_USE_PTHREADS)
377             pthread_mutex_lock(&store->table_
378             cache_mutex);
379 #elif defined(OS_USE_WINDOWS_CRITICAL_
380 SECTION)
381             EnterCriticalSection(&store->table_
382             cache_mutex);
383 #endif
384
385         for (size_t i = 0; i < store->num_
386         tables; ++i) {
387             if (store->tables[i].valid &&
388                 strcmp(store->tables[i].name
389                 , name) == 0) {
390                 goto get_table_dbi_cleanup;
391             }
392         }
393
394         if (store->num_tables >= store->
395             tables_capacity) {
396             size_t new_capacity = (store->
397                 tables_capacity == 0) ? 16 :
398
399 #if defined(OS_USE_PTHREADS)
400             pthread_mutex_lock(&store->table_
401             cache_mutex);
402 #elif defined(OS_USE_WINDOWS_CRITICAL_
403 SECTION)
404             EnterCriticalSection(&store->table_
405             cache_mutex);
406 #endif
407
408         for (size_t i = 0; i < store->num_
409         tables; ++i) {
410             if (store->tables[i].valid &&
411                 strcmp(store->tables[i].name
412                 , name) == 0) {
413                 goto get_table_dbi_cleanup;
414             }
415         }
416
417         if (store->num_tables >= store->
418             tables_capacity) {
419             size_t new_capacity = (store->
420                 tables_capacity == 0) ? 16 :
421
422 #if defined(OS_USE_PTHREADS)
423             pthread_mutex_lock(&store->table_
424             cache_mutex);
425 #elif defined(OS_USE_WINDOWS_CRITICAL_
426 SECTION)
427             EnterCriticalSection(&store->table_
428             cache_mutex);
429 #endif
430
431         for (size_t i = 0; i < store->num_
432         tables; ++i) {
433             if (store->tables[i].valid &&
434                 strcmp(store->tables[i].name
435                 , name) == 0) {
436                 goto get_table_dbi_cleanup;
437             }
438         }
439
440         if (store->num_tables >= store->
441             tables_capacity) {
442             size_t new_capacity = (store->
443                 tables_capacity == 0) ? 16 :
444
445 #if defined(OS_USE_PTHREADS)
446             pthread_mutex_lock(&store->table_
447             cache_mutex);
448 #elif defined(OS_USE_WINDOWS_CRITICAL_
449 SECTION)
450             EnterCriticalSection(&store->table_
451             cache_mutex);
452 #endif
453
454         for (size_t i = 0; i < store->num_
455         tables; ++i) {
456             if (store->tables[i].valid &&
457                 strcmp(store->tables[i].name
458                 , name) == 0) {
459                 goto get_table_dbi_cleanup;
460             }
461         }
462
463         if (store->num_tables >= store->
464             tables_capacity) {
465             size_t new_capacity = (store->
466                 tables_capacity == 0) ? 16 :
467
468 #if defined(OS_USE_PTHREADS)
469             pthread_mutex_lock(&store->table_
470             cache_mutex);
471 #elif defined(OS_USE_WINDOWS_CRITICAL_
472 SECTION)
473             EnterCriticalSection(&store->table_
474             cache_mutex);
475 #endif
476
477         for (size_t i = 0; i < store->num_
478         tables; ++i) {
479             if (store->tables[i].valid &&
480                 strcmp(store->tables[i].name
481                 , name) == 0) {
482                 goto get_table_dbi_cleanup;
483             }
484         }
485
486         if (store->num_tables >= store->
487             tables_capacity) {
488             size_t new_capacity = (store->
489                 tables_capacity == 0) ? 16 :
490
491 #if defined(OS_USE_PTHREADS)
492             pthread_mutex_lock(&store->table_
493             cache_mutex);
494 #elif defined(OS_USE_WINDOWS_CRITICAL_
495 SECTION)
496             EnterCriticalSection(&store->table_
497             cache_mutex);
498 #endif
499
500         for (size_t i = 0; i < store->num_
501         tables; ++i) {
502             if (store->tables[i].valid &&
503                 strcmp(store->tables[i].name
504                 , name) == 0) {
505                 goto get_table_dbi_cleanup;
506             }
507         }
508
509         if (store->num_tables >= store->
510             tables_capacity) {
511             size_t new_capacity = (store->
512                 tables_capacity == 0) ? 16 :
513
514 #if defined(OS_USE_PTHREADS)
515             pthread_mutex_lock(&store->table_
516             cache_mutex);
517 #elif defined(OS_USE_WINDOWS_CRITICAL_
518 SECTION)
519             EnterCriticalSection(&store->table_
520             cache_mutex);
521 #endif
522
523         for (size_t i = 0; i < store->num_
524         tables; ++i) {
525             if (store->tables[i].valid &&
526                 strcmp(store->tables[i].name
527                 , name) == 0) {
528                 goto get_table_dbi_cleanup;
529             }
530         }
531
532         if (store->num_tables >= store->
533             tables_capacity) {
534             size_t new_capacity = (store->
535                 tables_capacity == 0) ? 16 :
536
537 #if defined(OS_USE_PTHREADS)
538             pthread_mutex_lock(&store->table_
539             cache_mutex);
540 #elif defined(OS_USE_WINDOWS_CRITICAL_
541 SECTION)
542             EnterCriticalSection(&store->table_
543             cache_mutex);
544 #endif
545
546         for (size_t i = 0; i < store->num_
547         tables; ++i) {
548             if (store->tables[i].valid &&
549                 strcmp(store->tables[i].name
550                 , name) == 0) {
551                 goto get_table_dbi_cleanup;
552             }
553         }
554
555         if (store->num_tables >= store->
556             tables_capacity) {
557             size_t new_capacity = (store->
558                 tables_capacity == 0) ? 16 :
559
560 #if defined(OS_USE_PTHREADS)
561             pthread_mutex_lock(&store->table_
562             cache_mutex);
563 #elif defined(OS_USE_WINDOWS_CRITICAL_
564 SECTION)
565             EnterCriticalSection(&store->table_
566             cache_mutex);
567 #endif
568
569         for (size_t i = 0; i < store->num_
570         tables; ++i) {
571             if (store->tables[i].valid &&
572                 strcmp(store->tables[i].name
573                 , name) == 0) {
574                 goto get_table_dbi_cleanup;
575             }
576         }
577
578         if (store->num_tables >= store->
579             tables_capacity) {
580             size_t new_capacity = (store->
581                 tables_capacity == 0) ? 16 :
582
583 #if defined(OS_USE_PTHREADS)
584             pthread_mutex_lock(&store->table_
585             cache_mutex);
586 #elif defined(OS_USE_WINDOWS_CRITICAL_
587 SECTION)
588             EnterCriticalSection(&store->table_
589             cache_mutex);
590 #endif
591
592         for (size_t i = 0; i < store->num_
593         tables; ++i) {
594             if (store->tables[i].valid &&
595                 strcmp(store->tables[i].name
596                 , name) == 0) {
597                 goto get_table_dbi_cleanup;
598             }
599         }
600
601         if (store->num_tables >= store->
602             tables_capacity) {
603             size_t new_capacity = (store->
604                 tables_capacity == 0) ? 16 :
605
606 #if defined(OS_USE_PTHREADS)
607             pthread_mutex_lock(&store->table_
608             cache_mutex);
609 #elif defined(OS_USE_WINDOWS_CRITICAL_
610 SECTION)
611             EnterCriticalSection(&store->table_
612             cache_mutex);
613 #endif
614
615         for (size_t i = 0; i < store->num_
616         tables; ++i) {
617             if (store->tables[i].valid &&
618                 strcmp(store->tables[i].name
619                 , name) == 0) {
620                 goto get_table_dbi_cleanup;
621             }
622         }
623
624         if (store->num_tables >= store->
625             tables_capacity) {
626             size_t new_capacity = (store->
627                 tables_capacity == 0) ? 16 :
628
629 #if defined(OS_USE_PTHREADS)
630             pthread_mutex_lock(&store->table_
631             cache_mutex);
632 #elif defined(OS_USE_WINDOWS_CRITICAL_
633 SECTION)
634             EnterCriticalSection(&store->table_
635             cache_mutex);
636 #endif
637
638         for (size_t i = 0; i < store->num_
639         tables; ++i) {
640             if (store->tables[i].valid &&
641                 strcmp(store->tables[i].name
642                 , name) == 0) {
643                 goto get_table_dbi_cleanup;
644             }
645         }
646
647         if (store->num_tables >= store->
648             tables_capacity) {
649             size_t new_capacity = (store->
650                 tables_capacity == 0) ? 16 :
651
652 #if defined(OS_USE_PTHREADS)
653             pthread_mutex_lock(&store->table_
654             cache_mutex);
655 #elif defined(OS_USE_WINDOWS_CRITICAL_
656 SECTION)
657             EnterCriticalSection(&store->table_
658             cache_mutex);
659 #endif
660
661         for (size_t i = 0; i < store->num_
662         tables; ++i) {
663             if (store->tables[i].valid &&
664                 strcmp(store->tables[i].name
665                 , name) == 0) {
666                 goto get_table_dbi_cleanup;
667             }
668         }
669
670         if (store->num_tables >= store->
671             tables_capacity) {
672             size_t new_capacity = (store->
673                 tables_capacity == 0) ? 16 :
674
675 #if defined(OS_USE_PTHREADS)
676             pthread_mutex_lock(&store->table_
677             cache_mutex);
678 #elif defined(OS_USE_WINDOWS_CRITICAL_
679 SECTION)
680             EnterCriticalSection(&store->table_
681             cache_mutex);
682 #endif
683
684         for (size_t i = 0; i < store->num_
685         tables; ++i) {
686             if (store->tables[i].valid &&
687                 strcmp(store->tables[i].name
688                 , name) == 0) {
689                 goto get_table_dbi_cleanup;
690             }
691         }
692
693         if (store->num_tables >= store->
694             tables_capacity) {
695             size_t new_capacity = (store->
696                 tables_capacity == 0) ? 16 :
697
698 #if defined(OS_USE_PTHREADS)
699             pthread_mutex_lock(&store->table_
700             cache_mutex);
701 #elif defined(OS_USE_WINDOWS_CRITICAL_
702 SECTION)
703             EnterCriticalSection(&store->table_
704             cache_mutex);
705 #endif
706
707         for (size_t i = 0; i < store->num_
708         tables; ++i) {
709             if (store->tables[i].valid &&
710                 strcmp(store->tables[i].name
711                 , name) == 0) {
712                 goto get_table_dbi_cleanup;
713             }
714         }
715
716         if (store->num_tables >= store->
717             tables_capacity) {
718             size_t new_capacity = (store->
719                 tables_capacity == 0) ? 16 :
720
721 #if defined(OS_USE_PTHREADS)
722             pthread_mutex_lock(&store->table_
723             cache_mutex);
724 #elif defined(OS_USE_WINDOWS_CRITICAL_
725 SECTION)
726             EnterCriticalSection(&store->table_
727             cache_mutex);
728 #endif
729
730         for (size_t i = 0; i < store->num_
731         tables; ++i) {
732             if (store->tables[i].valid &&
733                 strcmp(store->tables[i].name
734                 , name) == 0) {
735                 goto get_table_dbi_cleanup;
736             }
737         }
738
739         if (store->num_tables >= store->
740             tables_capacity) {
741             size_t new_capacity = (store->
742                 tables_capacity == 0) ? 16 :
743
744 #if defined(OS_USE_PTHREADS)
745             pthread_mutex_lock(&store->table_
746             cache_mutex);
747 #elif defined(OS_USE_WINDOWS_CRITICAL_
748 SECTION)
749             EnterCriticalSection(&store->table_
750             cache_mutex);
751 #endif
752
753         for (size_t i = 0; i < store->num_
754         tables; ++i) {
755             if (store->tables[i].valid &&
756                 strcmp(store->tables[i].name
757                 , name) == 0) {
758                 goto get_table_dbi_cleanup;
759             }
760         }
761
762         if (store->num_tables >= store->
763             tables_capacity) {
764             size_t new_capacity = (store->
765                 tables_capacity == 0) ? 16 :
766
767 #if defined(OS_USE_PTHREADS)
768             pthread_mutex_lock(&store->table_
769             cache_mutex);
770 #elif defined(OS_USE_WINDOWS_CRITICAL_
771 SECTION)
772             EnterCriticalSection(&store->table_
773             cache_mutex);
774 #endif
775
776         for (size_t i = 0; i < store->num_
777         tables; ++i) {
778             if (store->tables[i].valid &&
779                 strcmp(store->tables[i].name
780                 , name) == 0) {
781                 goto get_table_dbi_cleanup;
782             }
783         }
784
785         if (store->num_tables >= store->
786             tables_capacity) {
787             size_t new_capacity = (store->
788                 tables_capacity == 0) ? 16 :
789
790 #if defined(OS_USE_PTHREADS)
791             pthread_mutex_lock(&store->table_
792             cache_mutex);
793 #elif defined(OS_USE_WINDOWS_CRITICAL_
794 SECTION)
795             EnterCriticalSection(&store->table_
796             cache_mutex);
797 #endif
798
799         for (size_t i = 0; i < store->num_
800         tables; ++i) {
801             if (store->tables[i].valid &&
802                 strcmp(store->tables[i].name
803                 , name) == 0) {
804                 goto get_table_dbi_cleanup;
805             }
806         }
807
808         if (store->num_tables >= store->
809             tables_capacity) {
810             size_t new_capacity = (store->
811                 tables_capacity == 0) ? 16 :
812
813 #if defined(OS_USE_PTHREADS)
814             pthread_mutex_lock(&store->table_
815             cache_mutex);
816 #elif defined(OS_USE_WINDOWS_CRITICAL_
817 SECTION)
818             EnterCriticalSection(&store->table_
819             cache_mutex);
820 #endif
821
822         for (size_t i = 0; i < store->num_
823         tables; ++i) {
824             if (store->tables[i].valid &&
825                 strcmp(store->tables[i].name
826                 , name) == 0) {
827                 goto get_table_dbi_cleanup;
828             }
829         }
830
831         if (store->num_tables >= store->
832             tables_capacity) {
833             size_t new_capacity = (store->
834                 tables_capacity == 0) ? 16 :
835
836 #if defined(OS_USE_PTHREADS)
837             pthread_mutex_lock(&store->table_
838             cache_mutex);
839 #elif defined(OS_USE_WINDOWS_CRITICAL_
840 SECTION)
841             EnterCriticalSection(&store->table_
842             cache_mutex);
843 #endif
844
845         for (size_t i = 0; i < store->num_
846         tables; ++i) {
847             if (store->tables[i].valid &&
848                 strcmp(store->tables[i].name
849                 , name) == 0) {
850                 goto get_table_dbi_cleanup;
851             }
852         }
853
854         if (store->num_tables >= store->
855             tables_capacity) {
856             size_t new_capacity = (store->
857                 tables_capacity == 0) ? 16 :
858
859 #if defined(OS_USE_PTHREADS)
860             pthread_mutex_lock(&store->table_
861             cache_mutex);
862 #elif defined(OS_USE_WINDOWS_CRITICAL_
863 SECTION)
864             EnterCriticalSection(&store->table_
865             cache_mutex);
866 #endif
867
868         for (size_t i = 0; i < store->num_
869         tables; ++i) {
870             if (store->tables[i].valid &&
871                 strcmp(store->tables[i].name
872                 , name) == 0) {
873                 goto get_table_dbi_cleanup;
874             }
875         }
876
877         if (store->num_tables >= store->
878             tables_capacity) {
879             size_t new_capacity = (store->
880                 tables_capacity == 0) ? 16 :
881
882 #if defined(OS_USE_PTHREADS)
883             pthread_mutex_lock(&store->table_
884             cache_mutex);
885 #elif defined(OS_USE_WINDOWS_CRITICAL_
886 SECTION)
887             EnterCriticalSection(&store->table_
888             cache_mutex);
889 #endif
890
891         for (size_t i = 0; i < store->num_
892         tables; ++i) {
893             if (store->tables[i].valid &&
894                 strcmp(store->tables[i].name
895                 , name) == 0) {
896                 goto get_table_dbi_cleanup;
897             }
898         }
899
900         if (store->num_tables >= store->
901             tables_capacity) {
902             size_t new_capacity = (store->
903                 tables_capacity == 0) ? 16 :
904
905 #if defined(OS_USE_PTHREADS)
906             pthread_mutex_lock(&store->table_
907             cache_mutex);
908 #elif defined(OS_USE_WINDOWS_CRITICAL_
909 SECTION)
910             EnterCriticalSection(&store->table_
911             cache_mutex);
912 #endif
913
914         for (size_t i = 0; i < store->num_
915         tables; ++i) {
916             if (store->tables[i].valid &&
917                 strcmp(store->tables[i].name
918                 , name) == 0) {
919                 goto get_table_dbi_cleanup;
920             }
921         }
922
923         if (store->num_tables >= store->
924             tables_capacity) {
925             size_t new_capacity = (store->
926                 tables_capacity == 0) ? 16 :
927
928 #if defined(OS_USE_PTHREADS)
929             pthread_mutex_lock(&store->table_
930             cache_mutex);
931 #elif defined(OS_USE_WINDOWS_CRITICAL_
932 SECTION)
933             EnterCriticalSection(&store->table_
934             cache_mutex);
935 #endif
936
937         for (size_t i = 0; i < store->num_
938         tables; ++i) {
939             if (store->tables[i].valid &&
940                 strcmp(store->tables[i].name
941                 , name) == 0) {
942                 goto get_table_dbi_cleanup;
943             }
944         }
945
946         if (store->num_tables >= store->
947             tables_capacity) {
948             size_t new_capacity = (store->
949                 tables_capacity == 0) ? 16 :
950
951 #if defined(OS_USE_PTHREADS)
952             pthread_mutex_lock(&store->table_
953             cache_mutex);
954 #elif defined(OS_USE_WINDOWS_CRITICAL_
955 SECTION)
956             EnterCriticalSection(&store->table_
957             cache_mutex);
958 #endif
959
960         for (size_t i = 0; i < store->num_
961         tables; ++i) {
962             if (store->tables[i].valid &&
963                 strcmp(store->tables[i].name
964                 , name) == 0) {
965                 goto get_table_dbi_cleanup;
966             }
967         }
968
969         if (store->num_tables >= store->
970             tables_capacity) {
971             size_t new_capacity = (store->
972                 tables_capacity == 0) ? 16 :
973
974 #if defined(OS_USE_PTHREADS)
975             pthread_mutex_lock(&store->table_
976             cache_mutex);
977 #elif defined(OS_USE_WINDOWS_CRITICAL_
978 SECTION)
979             EnterCriticalSection(&store->table_
980             cache_mutex);
981 #endif
982
983         for (size_t i = 0; i < store->num_
984         tables; ++i) {
985             if (store->tables[i].valid &&
986                 strcmp(store->tables[i].name
987                 , name) == 0) {
988                 goto get_table_dbi_cleanup;
989             }
990         }
991
992         if (store->num_tables >= store->
993             tables_capacity) {
994             size_t new_capacity = (store->
995                 tables_capacity == 0) ? 16 :
996
997 #if defined(OS_USE_PTHREADS)
998             pthread_mutex_lock(&store->table_
999             cache_mutex);
1000 #elif defined(OS_USE_WINDOWS_CRITICAL_
1001 SECTION)
1002             EnterCriticalSection(&store->table_
1003             cache_mutex);
1004 #endif
1005
1006         for (size_t i = 0; i < store->num_
1007         tables; ++i) {
1008             if (store->tables[i].valid &&
1009                 strcmp(store->tables[i].name
1010                 , name) == 0) {
1011                 goto get_table_dbi_cleanup;
1012             }
1013         }
1014
1015         if (store->num_tables >= store->
1016             tables_capacity) {
1017             size_t new_capacity = (store->
1018                 tables_capacity == 0) ? 16 :
1019
1020 #if defined(OS_USE_PTHREADS)
1021             pthread_mutex_lock(&store->table_
1022             cache_mutex);
1023 #elif defined(OS_USE_WINDOWS_CRITICAL_
1024 SECTION)
1025             EnterCriticalSection(&store->table_
1026             cache_mutex);
1027 #endif
1028
1029         for (size_t i = 0; i < store->num_
1030         tables; ++i) {
1031             if (store->tables[i].valid &&
1032                 strcmp(store->tables[i].name
1033                 , name) == 0) {
1034                 goto get_table_dbi_cleanup;
1035             }
1036         }
1037
1038         if (store->num_tables >= store->
1039             tables_capacity) {
1040             size_t new_capacity = (store->
1041                 tables_capacity == 0) ? 16 :
1042
1043 #if defined(OS_USE_PTHREADS)
1044             pthread_mutex_lock(&store->table_
1045             cache_mutex);
1046 #elif defined(OS_USE_WINDOWS_CRITICAL_
1047 SECTION)
1048             EnterCriticalSection(&store->table_
1049             cache_mutex);
1050 #endif
1051
1052         for (size_t i = 0; i < store->num_
1053         tables; ++i) {
1054             if (store->tables[i].valid &&
1055                 strcmp(store->tables[i].name
1056                 , name) == 0) {
1057                 goto get_table_dbi_cleanup;
1058             }
1059         }
1060
1061         if (store->num_tables >= store->
1062             tables_capacity) {
1063             size_t new_capacity = (store->
1064                 tables_capacity == 0) ? 16 :
1065
1066 #if defined(OS_USE_PTHREADS)
1067             pthread_mutex_lock(&amp
```

```

119     store->tables_capacity * 2;
120     os_table_info_t *new_tables = (
121         os_table_info_t *)realloc(
122             store->tables, new_capacity
123             * sizeof(os_table_info_t));
124     if (!new_tables) {
125         #if defined(OS_USE_PTHREADS)
126             pthread_mutex_unlock(&
127                 store->table_cache_
128                 mutex);
129     #elif defined(OS_USE_WINDOWS_
130         _CRITICAL_SECTION)
131         LeaveCriticalSection(&
132             store->table_cache_
133             mutex);
134     #endif
135         return FB_SERIALIZER_ERROR_
136             ALLOCATION;
137     }
138     store->tables = new_tables;
139     store->tables_capacity = new_
140         capacity;
141 }
142 store->tables[store->num_tables].name =
143     os_strdup(name);
144 if (!store->tables[store->num_tables].
145     name) {
146     #if defined(OS_USE_PTHREADS)
147         pthread_mutex_unlock(&store
148             ->table_cache_mutex);
149     #elif defined(OS_USE_WINDOWS_
150         _CRITICAL_SECTION)
151         LeaveCriticalSection(&store
152             ->table_cache_mutex);
153     #endif
154         return FB_SERIALIZER_ERROR_
155             ALLOCATION;
156     }
157 store->tables[store->num_tables].dbi
158     = *dbi_out;
159 store->tables[store->num_tables].
160     valid = true;
161 store->num_tables++;

162 get_table_db_cleanups:
163 #if defined(OS_USE_PTHREADS)
164     pthread_mutex_unlock(&store->table_
165         cache_mutex);
166 #elif defined(OS_USE_WINDOWS_CRITICAL_
167     SECTION)
168     LeaveCriticalSection(&store->table_
169         cache_mutex);
170 #endif
171     return FB_SERIALIZER_OK;
172 }

173 int object_store_create(object_store_t
174     **store_ptr) {
175     if (!store_ptr) return -1;
176     *store_ptr = (object_store_t*)calloc
177         (1, sizeof(object_store_t));
178     if (!*store_ptr) return -1;

179 #if defined(OS_USE_PTHREADS)
180     if (pthread_mutex_init(&(*store_ptr)
181         ->table_cache_mutex, NULL) != 0)
182     {
183         perror("Failed to initialize_
184             table_cache_mutex");
185     }
186 #endif
187     free(*store_ptr);
188     *store_ptr = NULL;
189     return -1;
190 }

191 #elif defined(OS_USE_WINDOWS_CRITICAL_
192     SECTION)
193     InitializeCriticalSection(&(*store_
194         ptr)->table_cache_mutex);
195 #endif
196     return 0;
197 }

198 void object_store_destroy(object_store_t
199     **store_ptr) {
200     if (!store_ptr || !*store_ptr)
201         return;
202     object_store_t *store = *store_ptr;
203     if (store->env) {
204         mdb_env_close(store->env);
205     }
206     if (store->tables) {
207         for (size_t i = 0; i < store->
208             num_tables; ++i) {
209             free(store->tables[i].name);
210         }
211         free(store->tables);
212     }
213 #if defined(OS_USE_PTHREADS)
214     pthread_mutex_destroy(&store->table_
215         cache_mutex);
216 #elif defined(OS_USE_WINDOWS_CRITICAL_
217     SECTION)
218     DeleteCriticalSection(&store->table_
219         cache_mutex);
220 #endif
221     free(store);
222     *store_ptr = NULL;
223 }

224 fb_serializer_status_t object_store_init
225     _db(object_store_t *store, const
226         char *path, size_t mapsize, unsigned
227         int maxdbs) {
228     if (!store || !path) return FB_
229         SERIALIZER_ERROR_INVALID_INPUT;
230     if (store->env) return FB_SERIALIZER_
231         _ERROR_INVALID_INPUT;

232 #if defined(_WIN32) && !defined(__
233         CYGWIN__)
234     if (!.CreateDirectoryA(path, NULL)
235         && GetLastError() != ERROR_
236             _ALREADY_EXISTS) {
237         return FB_SERIALIZER_ERROR_
238             SYSTEM_CALL;
239     }
240 #else
241     if (mkdir(path, 0755) != 0 &&
242         errno != EEXIST) {
243         return FB_SERIALIZER_ERROR_
244             SYSTEM_CALL;
245     }
246 #endif

247     int rc = mdb_env_create(&store->env)
248         ;
249     if (rc != MDB_SUCCESS) {
250         print_lmdb_error("mdb_env_create
251             ", rc);
252     }
253 }
```

```

210     return FB_SERIALIZER_ERROR_LMDB_
211     ENV_CREATE;
212 }
213 mdb_env_set_mapsize(store->env,
214     mapsize > 0 ? mapsize : OS_
215     DEFAULT_MAPSIZE);
216 mdb_env_set_maxdbs(store->env,
217     maxdbs > 0 ? maxdbs : OS_DEFAULT_
218     _MAX_DBMS);
219 rc = mdb_env_open(store->env, path,
220     MDB_WRITEMAP | MDB_NOSYNC, 0644)
221 ;
222 if (rc != MDB_SUCCESS) {
223     print_lmdb_error("mdb_env_open",
224         rc);
225     mdb_env_close(store->env);
226     store->env = NULL;
227     return FB_SERIALIZER_ERROR_LMDB_
228         ENV_OPEN;
229 }
230 return FB_SERIALIZER_OK;
231 }
232
233 fb_serializer_status_t object_store_
234     create_table(object_store_t *store,
235     const char *name) {
236     if (!store || !name) return FB_
237         SERIALIZER_ERROR_INVALID_INPUT;
238     if (!store->env) return FB_
239         SERIALIZER_ERROR_NOT_INITIALIZED
240     ;
241     MDB_dbi dbi;
242     return get_table_dbi(store, name, &
243         dbi, true);
244 }
245
246 fb_serializer_status_t object_store_
247     delete_table(object_store_t *store,
248     const char *name) {
249     if (!store || !name) return FB_
250         SERIALIZER_ERROR_INVALID_INPUT;
251     if (!store->env) return FB_
252         SERIALIZER_ERROR_NOT_INITIALIZED
253     ;
254     MDB_dbi dbi;
255     fb_serializer_status_t status = get_
256         table_dbi(store, name, &dbi,
257         false);
258     if (status != FB_SERIALIZER_OK)
259         return status;
260
261     MDB_txn *txn;
262     int rc = mdb_txn_begin(store->env,
263         NULL, 0, &txn);
264     if (rc != MDB_SUCCESS) { print_lmdb_
265         error("delete_table_(mdb_txn_
266         begin)", rc); return FB_
267         SERIALIZER_ERROR_LMDB_TXN_BEGIN;
268     }
269
270     rc = mdb_drop(txn, dbi, 1);
271     if (rc != MDB_SUCCESS) {
272         mdb_txn_abort(txn);
273         if (rc == MDB_NOTFOUND) return
274             FB_SERIALIZER_OK;
275         print_lmdb_error("delete_table_(
276             mdb_drop)", rc);
277     }
278
279     return FB_SERIALIZER_ERROR_LMDB_
280         DROP;
281 }
282
283 rc = mdb_txn_commit(txn);
284 if (rc != MDB_SUCCESS) {
285     print_lmdb_error("delete_table_(
286         mdb_txn_commit)", rc);
287     return FB_SERIALIZER_ERROR_LMDB_
288         TXN_COMMIT;
289 }
290
291 #if defined(OS_USE_PTHREADS)
292     pthread_mutex_lock(&store->table_
293         cache_mutex);
294 #elif defined(OS_USE_WINDOWS_CRITICAL_
295     SECTION)
296     EnterCriticalSection(&store->table_
297         cache_mutex);
298 #endif
299
300 for (size_t i = 0; i < store->num_
301     tables; ++i) {
302     if (store->tables[i].valid &&
303         strcmp(store->tables[i].name
304             , name) == 0) {
305         free(store->tables[i].name);
306         store->tables[i].name = NULL
307         ;
308         store->tables[i].valid =
309             false;
310         break;
311     }
312 }
313
314 #if defined(OS_USE_PTHREADS)
315     pthread_mutex_unlock(&store->table_
316         cache_mutex);
317 #elif defined(OS_USE_WINDOWS_CRITICAL_
318     SECTION)
319     LeaveCriticalSection(&store->table_
320         cache_mutex);
321 #endif
322
323 return FB_SERIALIZER_OK;
324 }
325
326 static uint64_t os_time_nanoseconds() {
327 #if defined(_WIN32) && !defined(__CYGWIN
328     )
329     FILETIME ft;
330     ULARGE_INTEGER ulti;
331     GetSystemTimeAsFileTime(&ft);
332     ulti.LowPart = ft.dwLowDateTime;
333     ulti.HighPart = ft.dwHighDateTime;
334     return (uli.QuadPart -
335         116444736000000000ULL) * 100ULL;
336 #else
337     struct timespec ts;
338     if (clock_gettime(CLOCK_REALTIME, &
339         ts) == -1) {
340         return 0;
341     }
342     return (uint64_t)ts.tv_sec *
343         1000000000ULL + (uint64_t)ts.tv_
344         nsec;
345 #endif
346 }
347
348 static void generate_random_bytes(
349     unsigned char* buffer, size_t length
350     ) {
351 #if defined(_WIN32) && !defined(__CYGWIN
352     )

```

```

299     }
300     HCRYPTPROV hCryptProv;
301     if (CryptAcquireContextA(&hCryptProv,
302         , NULL, NULL, PROV_RSA_AES,
303         CRYPT_VERIFYCONTEXT)) {
304         if(CryptGenRandom(hCryptProv, (
305             DWORD)length, buffer)) {
306             CryptReleaseContext(
307                 hCryptProv, 0);
308             return;
309         }
310         CryptReleaseContext(hCryptProv,
311             0);
312     }
313 #else
314     int fd = open("/dev/urandom", 0_
315         RONLY);
316     if (fd != -1) {
317         ssize_t result = read(fd, buffer
318             , length);
319         close(fd);
320         if (result == (ssize_t)length) {
321             return;
322         }
323     }
324 #endif
325     for (size_t i = 0; i < length; ++i)
326         buffer[i] = (unsigned char)(rand
327             () % 256);
328 }
329 fb_serializer_status_t object_store_
330     generate_key(unsigned char key_
331         buffer[OS_MAX_KEY_SIZE]) {
332     if(!key_buffer) return FB_SERIALIZER_
333         _ERROR_INVALID_INPUT;
334     uint64_t inverted_timestamp = UINT64_
335         MAX - os_time_nanoseconds();
336     for (int i = 0; i < 8; ++i) {
337         key_buffer[i] = (unsigned char)
338             ((inverted_timestamp >> (56
339                 - i * 8)) & 0xFF);
340     }
341     unsigned char random_data[16];
342     generate_random_bytes(random_data,
343         16);
344     md5_state_t md5_ctx;
345     md5_init(&md5_ctx);
346     md5_append(&md5_ctx, random_data,
347         16);
348     md5_finish(&md5_ctx, key_buffer + 8)
349         ;
350     generate_random_bytes(key_buffer +
351         24, 8);
352 }
353 return FB_SERIALIZER_OK;
354 }
355 static fb_serializer_status_t do_put(
356     object_store_t *store, const char *
357     table_name, const unsigned char key[
358         OS_MAX_KEY_SIZE], const c_object_t *
359     value, unsigned int flags) {
360     if (!store || !table_name || !key || !
361         value) return FB_SERIALIZER_
362         _ERROR_INVALID_INPUT;
363     if (!store->env) return FB_
364         SERIALIZER_ERROR_NOT_INITIALIZED
365         ;
366     MDB_db* dbi;
367     fb_serializer_status_t status = get_
368         table_dbi(store, table_name, &
369         dbi, (flags & MDB_NOOVERWRITE) ?
370             false : true);
371     if (status != FB_SERIALIZER_OK)
372         return status;
373     void *fb_buffer = NULL;
374     size_t fb_size = 0;
375     status = serialize_c_object(value, &
376         fb_buffer, &fb_size);
377     if (status != FB_SERIALIZER_OK) {
378         free(fb_buffer);
379         return status;
380     }
381     MDB_txn *txn;
382     MDB_val lmdb_key, lmdb_data;
383     int rc = mdb_txn_begin(store->env,
384         NULL, 0, &txn);
385     if (rc != MDB_SUCCESS) { free(fb_
386         buffer); print_lmdb_error("do_
387         put(mdb_txn_begin)", rc);
388         return FB_SERIALIZER_ERROR_LMDB_
389         TXN_BEGIN; }
390     lmdb_key.mv_size = OS_MAX_KEY_SIZE;
391     lmdb_key.mv_data = (void*)key;
392     lmdb_data.mv_size = fb_size;
393     lmdb_data.mv_data = fb_buffer;
394     rc = mdb_put(txn, dbi, &lmdb_key, &
395         lmdb_data, flags);
396     if (rc != MDB_SUCCESS) {
397         mdb_txn_abort(txn);
398         free(fb_buffer);
399         if (rc == MDB_KEYEXIST) return
400             FB_SERIALIZER_ERROR_LMDB_KEY_
401             EXISTS;
402         print_lmdb_error("do_put(mdb_
403             put)", rc);
404         return FB_SERIALIZER_ERROR_LMDB_
405             PUT;
406     }
407     rc = mdb_txn_commit(txn);
408     free(fb_buffer);
409     if (rc != MDB_SUCCESS) {
410         print_lmdb_error("do_put(mdb_
411             txn_commit)", rc);
412         return FB_SERIALIZER_ERROR_LMDB_
413             TXN_COMMIT;
414     }
415     return FB_SERIALIZER_OK;
416 }
417 fb_serializer_status_t object_store_put_
418     create_object(object_store_t *store,
419         const char *table_name, const
420         unsigned char key[OS_MAX_KEY_SIZE],
421         const c_object_t *value) {
422     return do_put(store, table_name, key
423         , value, MDB_NOOVERWRITE);
424 }
425 fb_serializer_status_t object_store_put_

```

```

object(object_store_t *store, const
      char *table_name, const unsigned
      char key[OS_MAX_KEY_SIZE], const c_
      object_t *value) {
    return do_put(store, table_name, key
                  , value, 0);
}
fb_serializer_status_t object_store_get_
object(object_store_t *store, const
      char *table_name, const unsigned
      char key[OS_MAX_KEY_SIZE], c_object_
      t **out_value) {
    if (!store || !table_name || !key || !
        !out_value) return FB_
        SERIALIZER_ERROR_INVALID_INPUT;
    *out_value = NULL;
    if (!store->env) return FB_
        SERIALIZER_ERROR_NOT_INITIALIZED
        ;
    MDB_db dbi;
    fb_serializer_status_t status = get_
        table_db(store, table_name, &
        dbi, false);
    if (status != FB_SERIALIZER_OK)
        return status;
    MDB_txn *txn;
    MDB_val lmbd_key, lmbd_data;
    int rc = mdb_txn_begin(store->env,
                           NULL, MDB_RDONLY, &txn);
    if (rc != MDB_SUCCESS) { print_lmbd_
        error("get_object_(mdb_txn_begin
        )", rc); return FB_SERIALIZER_
        ERROR_LMDB_TXN_BEGIN; }
    lmbd_key.mv_size = OS_MAX_KEY_SIZE;
    lmbd_key.mv_data = (void*)key;
    rc = mdb_get(txn, dbi, &lmbd_key, &
    lmbd_data);
    mdb_txn_abort(txn);
    if (rc == MDB_NOTFOUND) return FB_
        SERIALIZER_ERROR_LMDB_NOT_FOUND;
    if (rc != MDB_SUCCESS) { print_lmbd_
        error("get_object_(mdb_get)", rc)
        ; return FB_SERIALIZER_ERROR_
        LMDB_GET; }
    return deserialize_to_c_object(lmbd_
        data.mv_data, lmbd_data.mv_size,
        out_value);
}
fb_serializer_status_t object_store_delete_
object(object_store_t *store,
      const char *table_name, const
      unsigned char key[OS_MAX_KEY_SIZE])
{
    if (!store || !table_name || !key)
        return FB_SERIALIZER_ERROR_
        INVALID_INPUT;
    if (!store->env) return FB_
        SERIALIZER_ERROR_NOT_INITIALIZED
        ;
    MDB_db dbi;
    fb_serializer_status_t status = get_
        table_db(store, table_name, &
        dbi, false);
    if (status != FB_SERIALIZER_OK)
        return status;
    MDB_txn *txn;
    MDB_val lmbd_key;
    int rc = mdb_txn_begin(store->env,
                           NULL, 0, &txn);
    if (rc != MDB_SUCCESS) { print_lmbd_
        error("delete_object_(mdb_txn_
        begin)", rc); return FB_
        SERIALIZER_ERROR_LMDB_TXN_BEGIN;
        }
    lmbd_key.mv_size = OS_MAX_KEY_SIZE;
    lmbd_key.mv_data = (void*)key;
    rc = mdb_del(txn, dbi, &lmbd_key,
                 NULL);
    if (rc != MDB_SUCCESS) {
        mdb_txn_abort(txn);
        if (rc == MDB_NOTFOUND) return
            FB_SERIALIZER_ERROR_LMDB_NOT
            FOUND;
        print_lmbd_error("delete_object_
            (mdb_del)", rc);
        return FB_SERIALIZER_ERROR_LMDB_
            DEL;
    }
    rc = mdb_txn_commit(txn);
    if (rc != MDB_SUCCESS) {
        print_lmbd_error("delete_object_
            (mdb_txn_commit)", rc);
        return FB_SERIALIZER_ERROR_LMDB_
            TXN_COMMIT;
    }
    return FB_SERIALIZER_OK;
}
fb_serializer_status_t object_store_put_
object_from_json(
object_store_t *store,
const char *table_name,
const unsigned char key[OS_MAX_KEY_SIZE],
const char *object_json_string) {
#ifdef CJSON_ENABLED
    if (!store || !table_name || !key || !
        !object_json_string) return FB_
        SERIALIZER_ERROR_INVALID_INPUT;
    c_object_t *temp_c_obj = NULL;
    fb_serializer_status_t status = fb_
        object_from_json_string(object_
        json_string, NULL, NULL, &temp_c_
        _obj);
    if (status != FB_SERIALIZER_OK)
        return status;
    if (!temp_c_obj) return FB_
        SERIALIZER_ERROR_JSON_PARSE;
    status = object_store_put_object(
        store, table_name, key, temp_c_
        obj);
#endif
}

```

```

468     free_c_object(temp_c_obj);
469     return status;
470 #else
471     (void)store;
472     (void)table_name;
473     (void)key;
474     (void)object_json_string;
475     return FB_SERIALIZER_ERROR_INVALID_
476         INPUT; // JSON support not
477         compiled in
478 #endif
479 }
```

Listing 18: The implementation of the core database operations (`src/object_store/core/object_store.c`).

### 2.5.5 Validation

No system is complete without rigorous proof of its correctness. Our test suite, ‘`test_object_store.c`’, integrated with CTest, validates the entire persistence pipeline. The most important test is the “round-trip” validation, which programmatically builds a complex ‘`c_object_t`’, serializes it, writes it to the database, reads the raw bytes back, deserializes them into a new ‘`c_object_t`’, and performs a deep, field-by-field comparison to ensure the retrieved object is a perfect reconstruction of the original. Passing this test suite confirms that our foundation is solid, reliable, and ready to support the higher-level intelligent functions of the agent.

```

1 // test_object_store.c
2 #if defined(_linux_) || defined(_APPLE_) || defined(_FreeBSD_) ||
3     defined(_NetBSD_) || defined(_OpenBSD_) || defined(_sun) ||
4     defined(_QNXNT0_)
5     #ifndef _POSIX_C_SOURCE
6         #define _POSIX_C_SOURCE 200809L
7     #endif
8
9 #include "object_store.h"
10 #include "fb_serializer.h"
11 #include <stdio.h>
12 #include <string.h>
13 #include <stdlib.h>
14 #include <stdbool.h>
15 #include <math.h>
16 #include <time.h>
17
18 #ifdef _WIN32
19 #include <direct.h>
20 #else
21 #include <sys/stat.h>
22 #include <unistd.h>
23 #endif
24
25 #if defined(OS_USE_PTHREADS)
26 #include <pthread.h>
27 #endif
28
29 // --- Forward Declarations ---
30 void print_c_object(const c_object_t *
31                     obj, int indent);
32 bool compare_c_object(const c_object_t*
33                      o1, const c_object_t* o2);
34 c_object_t* create_test_user_object(
35     const char* id_suffix, const char*
36     name, int age);
37
38 // --- Test Assertion Macro ---
39 static int tests_passed = 0;
40 static int tests_total = 0;
41 #define ASSERT(condition, message) do {
42     \
43     tests_total++; \
44     if (condition) { \
45         tests_passed++; \
46         printf("[PASS] %s\n", message); \
47     } else { \
48         printf("[FAIL] %s\n", message); \
49     } \
50 } while (0)
51
52 // --- Test Helper: Cleanup ---
53 void cleanup_test_db_dir() {
54     #if defined(_WIN32)
55         _rmdir(TEST_DB_PATH);
56     #else
57         char command[256];
58         sprintf(command, "rm -rf %s",
59                 TEST_DB_PATH);
60         system(command);
61     #endif
62 }
63
64 // --- Main Test Functions ---
65 void run_basic_crud_tests(object_store_t
66                           *store) {
67     printf("\n--- Running Basic CRUD Tests ---\n");
68     const char* table = "users";
69     ASSERT(object_store_create_table(
70             store, table) == FB_SERIALIZER_OK,
71             "Create table 'users'");
72
73     unsigned char key1[OS_MAX_KEY_SIZE];
74     ASSERT(object_store_generate_key(
75             key1) == FB_SERIALIZER_OK,
76             "Generate key1");
77
78     c_object_t* user1 = create_test_user
79         _object("001", "Alice", 30);
80     ASSERT(object_store_create_object(
81             store, table, key1, user1) == FB
82             _SERIALIZER_OK,
83             "Create object user1");
84
85     c_object_t* retrieved_user = NULL;
86     ASSERT(object_store_get_object(store
87             , table, key1, &retrieved_user)
88             == FB_SERIALIZER_OK,
89             "Get object user1");
90     ASSERT(retrieved_user != NULL,
91             "Retrieved user1 is not null");
92     if (retrieved_user) {
93         ASSERT(compare_c_object(user1,
```

```

    retrieved_user), "Compare_
    original_and_retrieved_user1
    ");
}
free_c_object(retrieved_user);
retrieved_user = NULL;

c_object_t* user1_updated = create_
    test_user_object("001_upd", "
    AliceUpdated", 31);
ASSERT(object_store_put_object(store
    , table, key1, user1_updated) ==
    FB_SERIALIZER_OK, "Update_
    objectUser1");

ASSERT(object_store_get_object(store
    , table, key1, &retrieved_user)
    == FB_SERIALIZER_OK, "Get_
    updatedUser1");
ASSERT(retrieved_user != NULL, "
    RetrievedupdatedUser1isnotu
    null");
if (retrieved_user) {
    ASSERT(compare_c_object(user1_
        updated, retrieved_user), "
    Compareupdatedand_
    retrievedUser1");
}
free_c_object(retrieved_user);

ASSERT(object_store_delete_object(
    store, table, key1) == FB_
    SERIALIZER_OK, "Deleteobject_
    user1");
ASSERT(object_store_get_object(store
    , table, key1, &retrieved_user)
    == FB_SERIALIZER_ERROR_LMDB_NOT_
    FOUND, "VerifyUser1isdeleted"
    );

free_c_object(user1);
free_c_object(user1_updated);
}

void run_error_condition_tests(object_
    store_t *store) {
printf("\n---RunningError_
    ConditionTests---\n");
unsigned char key[OS_MAX_KEY_SIZE];
object_store_generate_key(key);
c_object_t* dummy_obj = create_test_
    user_object("dummy", "dummy", 0)
    ;
c_object_t* retrieved = NULL;

// FIX: The 'put' command is
// designed to be an "upsert",
// which includes creating
// the table if it doesn't exist.
// Therefore, this operation SHOULD
// succeed.
ASSERT(object_store_put_object(store
    , "non_existent_table", key,
    dummy_obj) == FB_SERIALIZER_OK,
    "Puttonon-existenttable_
    shouldsucceedbycreatingit");

// FIX: The 'get' command should
// correctly fail with NOT_FOUND on
// a non-existent table.
}

int main() {
srand((unsigned int)time(NULL));
cleanup_test_db_dir();

object_store_t *store = NULL;
ASSERT(object_store_create(&store)
    == 0, "Objectstorecreation");
ASSERT(store != NULL, "Storehandle
    isnotnull");

ASSERT(object_store_init_db(store,
    TEST_DB_PATH, OS_DEFAULT_MAPSIZE
    , OS_DEFAULT_MAX_DBS) == FB_
    SERIALIZER_OK, "DB_
    Initialization");

run_basic_crud_tests(store);
run_error_condition_tests(store);

object_store_destroy(&store);
ASSERT(store == NULL, "Storehandle
    isnullafterdestruction");

printf("\n-----\n");
printf("TestSuiteFinished.Passed:
    %d/%d\n", tests_passed, tests_
    total);
printf("-----\n");

cleanup_test_db_dir();
return (tests_passed == tests_total)
    ? 0 : 1;
}

// Helper Implementations
c_object_t* create_test_user_object(
    const char* id_suffix, const char*
    name, int age) {
c_object_t* obj = (c_object_t*)
    calloc(1, sizeof(c_object_t));
char buf[128];
snprintf(buf, sizeof(buf), "user_%
    , id_suffix);
obj->object_id = strdup(buf);
obj->table_name = strdup("users");
c_field_t* tail = NULL;

c_field_t* name_field = (c_field_t*)
    calloc(1, sizeof(c_field_t));
name_field->name = strdup("name");
name_field->value = create_c_string_
    value(name);
obj->fields = name field;
}

```

```

154     tail = name_field;
155
156     c_field_t* age_field = (c_field_t*)
157         calloc(1, sizeof(c_field_t));
158     age_field->name = strdup("age");
159     age_field->value = create_c_int_
160         value(age);
161     tail->next = age_field;
162
163     return obj;
164 }
165
166 bool compare_c_field_value(const c_field_
167     _value_t* v1, const c_field_value_t*
168     v2);
169
170 bool compare_c_fields(const c_field_t*
171     f1, const c_field_t* f2) {
172     while(f1 && f2) {
173         if (strcmp(f1->name, f2->name)
174             != 0) return false;
175         if (!compare_c_field_value(f1->
176             value, f2->value)) return
177             false;
178         f1 = f1->next;
179         f2 = f2->next;
180     }
181     return f1 == NULL && f2 == NULL; // Both must be null
182 }
183
184 bool compare_c_field_value(const c_field_
185     _value_t* v1, const c_field_value_t*
186     v2) {
187     if (!v1 || !v2 || v1->type != v2->
188         type) return false;
189     switch(v1->type) {
190         case C_FIELD_VALUE_TYPE_INT_
191             VALUE: return v1->data.int_
192                 val == v2->data.int_val;
193         case C_FIELD_VALUE_TYPE_STRING_
194             VALUE: return strcmp(v1->
195                 data.string_val, v2->data.
196                     string_val) == 0;
197         default: return false;
198     }
199 }
200
201 bool compare_c_object(const c_object_t*
202     o1, const c_object_t* o2) {
203     if (!o1 || !o2) return false;
204     if (strcmp(o1->object_id, o2->object_
205         _id) != 0) return false;
206     if (strcmp(o1->table_name, o2->table_
207         _name) != 0) return false;
208     return compare_c_fields(o1->fields,
209         o2->fields);
210 }

```

Listing 19: The Validation suite for the Object Store  
(src/object\_store/test/test\_object\_store)

With this robust and validated persistence layer, we have completed the foundational architecture. The agent now has both a body (the runtime) and a mind (the persistent store). We are now equipped to define the content that will bring it to life.

### 3 Content Of Objects

As we have developed the framework/language of building and interacting with and within the objects. Let us think of now the case how objects would behave, what i mean is that all objects have different functionalities they show on interaction which is caused by methods, properties that are either predefined, generated, or learned. Let us think of what properties or methods should be contained within an object so that we make the most abstract representation of the object. I guess it can be based on the fact that objects need to solve a particular task well like they are causes of some effect. This means that objects will specialize in a particular direction/cause-effect/task. What I mean is that what should be the most abstract representation in terms of content that should be in every object that helps every object intelligently specialize in their particular task as well as interacting with other objects in a network more efficiently.

#### 3.1 Objects As Functions!

See, every object needs to interact with other object so basically think of every object as a function which takes input and outputs some logical combination of functions in that functional space or combination of code that runs on the system or combination of both code and functions/objects. So, their are two questions that arise - how these objects interact? What do objects do in interaction?

#### 3.2 How these objects interact?

Every function takes input in a particular defined form and does action or returns value to the caller of function in a particular defined form which is their in its definition. So, a function is basically a combination of input (action by other object-/cause), computation, action, reaction and output (reaction to cause).

So, two functions interact by one function calling the other function by mentioning its location/reference (or a copy of that function or algorithm) in desired format and mentioning the arguments/inputs of that function along with its location/reference. The object on getting a call from a particular location extracts the inputs and performs the computations and actions (for reactions) as per the algorithm and returns the output/data to that location.

It is the responsibility of caller to keep in mind the output type of the called func-

tion/object. So, for two objects to interact, one object need to call the other object like their is call/cause-action/computation/effect-output/return(optional) architecture like the call wants to perform some action or get some data or both. The caller needs to keep in mind for a successful call from the side of the called objects, the location of called objects as well as inputs and from the mind for successful call for itself it needs to keep in mind the type of returned data also. So, let us talk about the inputs in the objects.

### 3.3 Inputs in Objects

Inputs are basically half-definition of a function. So, every objects need a good mechanism to manage the inputs. So, what should a good input mechanism for function be? If we think clearly, we can understand that inputs are associated with particular objects which we can call processing/functional object or simply agent object too and have id, type, name, cause, space, time, memory, can zoom, combine/filter in space, combine/filter in time, compare in memory and cause, etc. So, when an agent receives an input/cause, it produces some series of actions and based on the reactions of those actions produces an output or effect for the cause.

But how to represent inputs? Inputs are basically senses of the agent. So, it should be in the topological space. Like inputs of humans are eyes (type - video), ears (type - audio), skin (type - touch), nose (type - smell) and tongue (type - taste). So, all these are in topological space or tensor space. Like the video is in 2-D direction and 1-D time topological space, Image is in 2-D direction topological space. We can represent inputs in topological space via tensors. These inputs can be pre-processed and can be further processed by intelligence (logic/syntax + meaning/semantics + language(meaning/semantics + logic/syntax)) unit of object.

How the inputs are connected with objects? Every input is attached to its object of reference or the object to which it is given. Whenever an object receives a new input then it becomes live using its contents stored/defined in Object Store which tries to produce a suitable output for that input using data/intelligence from previous interactions as well as by intelligent reasoning on the interactions that have happened in other objects.

Inputs are basically context for the objects. So, each object has to get a context, or cause, to act upon. It then produces an output, or effect, for

the causal object and provides a new cause to any subsequent objects it acts on. This process is best understood as a complex graph of interactions, as shown in Figure 1.

The difference between a *Cause* and an *Action* is one of perspective and target. A *Cause* can be seen as an agent's internal impetus to act, which becomes an *Action* when directed at an external agent. Similarly, the relationship between an *Effect* and a *Reaction* depends on perspective. An *Effect* is the direct outcome of an *Action*. When this *Effect* is received by the agent that initiated the *Action*, it is perceived as a self-reaction; when it is received by a different agent, it elicits a *Reaction* from that agent. The entire cycle—where one agent's *Action* produces an *Effect* that prompts a *Reaction* in another—constitutes an *Interaction*. In this sequence, *Cause* and *Action* always precede *Effect* and *Reaction*. Inputs provide the initial context for this entire chain of events but do not, by themselves, produce the direct changes that *Actions* and *Effects* do.

Like everything which happens on cause is effect while the result of effect on the causal object is reaction. So, a cause is action when it deals with other objects and just cause when it originates in itself. Are you able to understand? So, we can say action on oneself is cause, cause on others is action. Reaction on me caused by me effect and effect on me caused by others is reaction and reaction on others caused me is called interaction. So, it is basically where is the cause and on whom we are observing its effect.

So, we have learned that inputs are not direct cause of effect but provide context for the agent to operate which serves its own defined cause. So, we now have to define the framework for handling inputs in C.

## 4 The Cognitive Substrate: An Architecture for Input

The creation of a human-level artificial agent necessitates a radical departure from conventional input handling paradigms. Modern AI systems, while powerful, typically employ monolithic, pre-processing pipelines that treat the conversion of external data into internal representations as a solved, mechanical problem. This approach conflates the fundamentally distinct cognitive processes of raw sensation, structured perception, and associative memory. Our central thesis is that this conflation is

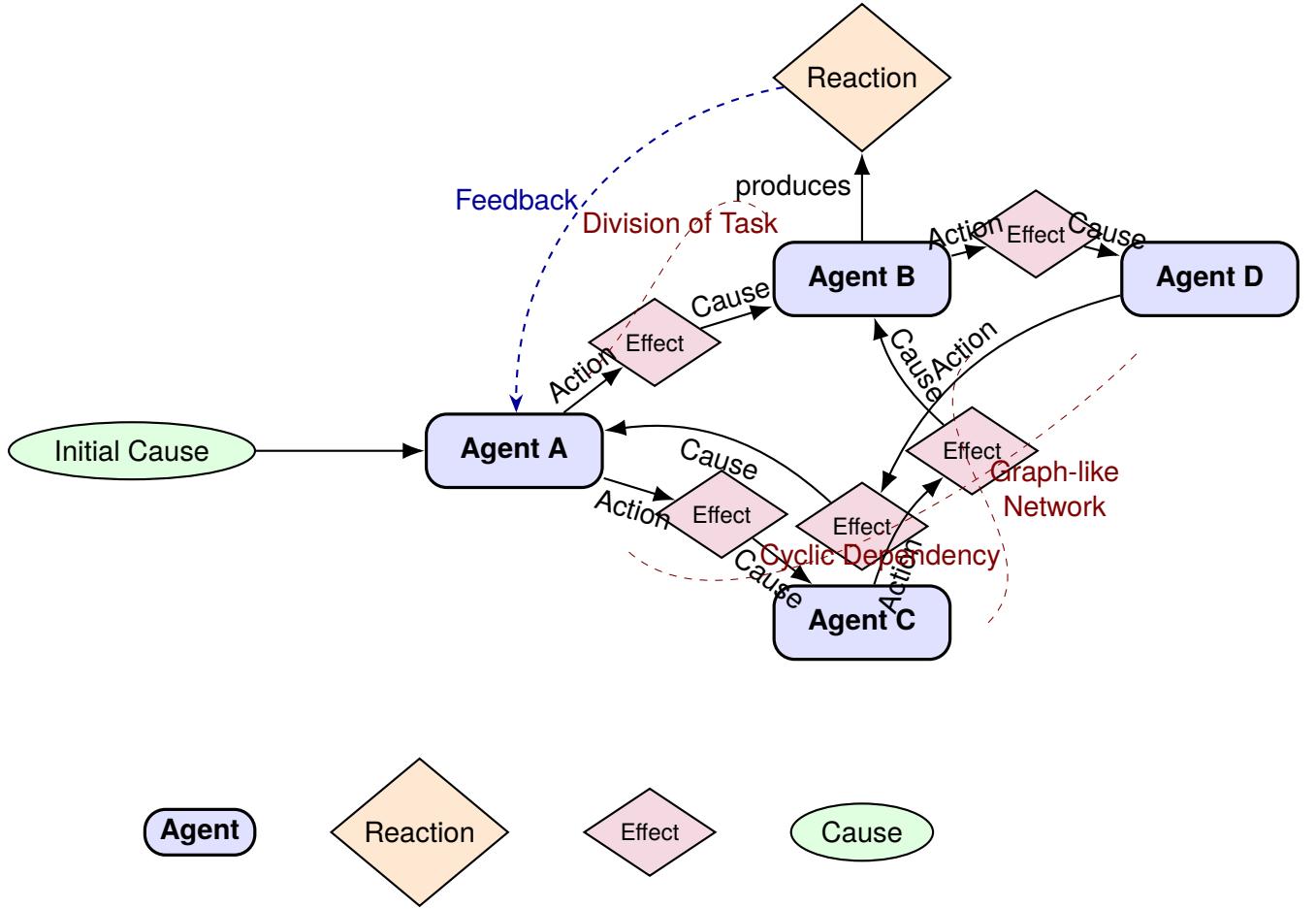


Figure 1: A comprehensive model of the agent interaction graph.

a primary obstacle to achieving true artificial general intelligence. It creates systems that are powerful data processors but poor learners—brittle, unable to adapt to novelty, and incapable of forming the deep, grounded understanding that characterizes biological cognition.

To overcome these limitations, we propose a **Cognitive Substrate**, an input architecture designed from first principles to mirror the hierarchical, self-organizing, and deeply interconnected nature of a biological cognitive system. Our framework is founded on a principled decomposition of input handling into three distinct, yet symbiotic, subsystems: **Sensation**, **Perception**, and **Memory**. These subsystems are not merely sequential processing stages; they represent a universal cognitive foundation—a shared “bedrock of reality”—available to all agents within the AGENT system. This design ensures that all agents operate within a shared conceptual universe, enabling communication and collective learning, while simultaneously allowing for extreme specialization through the unique structure of their individual experiences.

This section provides a detailed exposition of the theoretical foundations and logical reasoning that underpin this architecture.

#### 4.1 Sensation: The Principled Separation of Physics and Cognition

The external world presents itself as a chaotic multitude of disparate physical signals and data encodings. The first logical necessity for any cognitive system is to establish a clean and unambiguous interface between this external chaos and its internal cognitive processes. This is the sole and exclusive purpose of the Sensation Subsystem.

##### 4.1.1 Core Insight: Transduction Precedes Interpretation

Our architecture is founded on the principle that the act of converting a physical signal into a digital one (*sensation*) is fundamentally distinct from the act of interpreting what that signal means (*perception*). In biological systems, the retina does not “see” a cat; it transduces photons into a raw neural signal. The cognitive work of identifying the “cat”

happens much deeper in the brain. We model this separation explicitly. The Sensation Subsystem is a library of *sense organs* whose only function is to act as universal translators, converting external formats into a standardized, uncompressed **Raw Tensor**. It is a system of pure physics, intentionally devoid of intelligence.

#### 4.1.2 Architectural Implementation: The Autonomous Sense Organ

To achieve maximum portability and autonomy, the Sensation Subsystem is designed as a library of **SenseObject** blueprints. Each blueprint is a self-contained manifest detailing the necessary heavy libraries (e.g., `libavcodec.so`), their canonical public source URI, and the build commands required for on-device compilation. The blueprint also contains a lightweight Just-in-Time (JIT) C script that orchestrates the use of these libraries. This design choice is critical: it means the AGENK system is not dependent on a central provider for pre-compiled binaries. Upon encountering a novel data type, the system’s internal **Builder Module** can autonomously fetch the required source code, compile it on-device, and install the new *sense organ* into a shared `runtime_libs` directory, dynamically extending the perceptual capabilities of the entire system.

### 4.2 Perception: The Emergence of Structure from a Universal Engine

Once a raw signal is available, the system must impose structure upon it. This is the domain of the Perception Subsystem. Its purpose is to move beyond the raw data and discover the inherent, compositional patterns within, answering the question, “What is the underlying grammar of this piece of reality?”

#### 4.2.1 Core Insight: Reality is Compositional and Can Be Learned

Our central hypothesis is that all sensory data, regardless of modality, has a compositional grammar. We reject complex, hand-designed feature extraction pipelines in favor of a single, universal, and maximally abstract mechanism designed to discover this grammar: the **Universal Pattern Engine**. This engine, implemented as a dynamic, online Vector Quantization model (a “Living Codebook”), learns a shared “alphabet of reality” for each modality. There is one, and only one, **AlgorithmObject** for each modality’s

codebook (e.g., `global_vision_codebook`), which is trained on the collective experiences of all agents. This provides a common language for reality, ensuring a foundation for communication and generalization. Foundational concepts, like the visual pattern of a “straight line,” are learned once collectively and instantly become part of the shared perceptual substrate for all agents.

#### 4.2.2 The Perceptual Unit: A Hierarchy of Emergent Properties

The true power of the Universal Pattern Engine lies not just in its primary output, but in the rich hierarchy of interpretations that naturally emerge from its single, simple operation. The final output of this stage is a multi-layered **Perceptual Unit**, which is then consolidated into a permanent **MemoryObject**.

- **Layer 1: The Tokenized Representation.** The primary output of the engine. A grid or sequence of discrete token IDs from the universal codebook. This is the raw data represented in the agent’s learned, compositional language. The engine’s online learning rule, shown in Equation 1, ensures this alphabet continuously adapts to the statistical properties of the system’s total experience.

$$\mathbf{v}_w \leftarrow \mathbf{v}_w + \eta \cdot (\mathbf{p} - \mathbf{v}_w) \quad (1)$$

where  $\mathbf{v}_w$  is the vector of the winning neuron,  $\mathbf{p}$  is the input patch vector, and  $\eta$  is the learning rate.

- **Layer 2: The Saliency Map.** An emergent property of the quantization error, shown in Equation 2. Regions of the raw tensor that are a poor match for any existing token in the universal alphabet are, by definition, novel, surprising, and thus salient. This provides a computationally free attention mechanism.

$$\varepsilon = \|\mathbf{p} - \mathbf{v}_w\| \quad (2)$$

- **Layer 3: The Object Segments.** An emergent property of the tokenized representation’s grammar. By analyzing the co-occurrence statistics of the universal tokens, the system can identify object boundaries at points of low grammatical probability. A contiguous region of tokens with high co-occurrence probability is, by definition, a coherent object.

- **Layer 4: The Semantic Embedding.** A “cultural bootstrap.” To ground the agent’s internal grammar in the shared space of human knowledge, a powerful vendor embedding model is used to assign a holistic semantic vector to the perception. This provides a top-down, conceptual “gist” of the scene.

### 4.3 Memory: A Differentiated Substrate for Holistic Recall

A perception is fleeting unless it is consolidated into a permanent, queryable memory. The Memory Subsystem provides the substrate for storing, connecting, and retrieving the agent’s entire lifetime of experience. It answers the fundamental question, “How does this new perception relate to everything I already know?”

#### 4.3.1 Core Insight: Recall is Multi-faceted and Context-Grounded

Memory is not a simple database table. Different cognitive tasks require different types of recall. Furthermore, true understanding does not come from a pre-trained model alone, but emerges from discovering the statistical regularities within a coherent context. Our architecture is therefore designed to support four distinct modes of recall, moving from simple chronological access to deep, context-aware, cross-modal association.

#### 4.3.2 The Tiers of Recall

1. **Episodic Recall (The Chronological Chain):** Each `MemoryObject` is linked to the previous memory from the same agent’s input stream via a `previous_memory_ref`. This creates a per-agent, per-input linked list, enabling blazingly fast  $O(k)$  traversal of personal, chronological history (“*What just happened in this specific feed?*”).
2. **Personal Associative Recall (The Specialized Mind):** An agent’s expertise is embodied in its **private, personal HNSW index**. This index uses a **“Structural/Contextual Fingerprint”** of the agent’s own memories—a vector representing the unique combination of *universal tokens* within its personal experiences—to answer the question, “*When have I, personally, ever encountered a similar composition of patterns before?*”

3. **Universal Semantic Recall (The ”Cultural” Knowledge Graph):** To enable broad, analogical reasoning, we employ a global **Hierarchical Navigable Small World (HNSW) graph**. This index, built upon the semantic embeddings of *all* memories from *all* agents, answers the question, “*What does the shared body of human knowledge know that is conceptually similar to this?*”

4. **Grounded Cross-Modal Recall (The Contextual Co-occurrence Graph):** This is the most powerful tier of memory, as it allows the agent to build its own model of how the world works based on direct, contextual experience. It is the engine for discovering grounded, multimodal associations by answering the question, “*What sensory patterns consistently appear together when a specific, coherent task is being performed?*”

- **The Core Problem with Time-Based Association:** A naive approach that links sensory events based purely on temporal proximity is fundamentally flawed. Two independent agents working on unrelated tasks may perceive events at the same millisecond; this is a meaningless coincidence. Learning from such noise would corrupt the agent’s understanding of causality.
- **The Solution – The Contextual Graph:** True association is bound by context. In our system, the fundamental unit of context is a **Task**, which is uniquely identified by the combination of an `agent_id` and a `call_id`. The Co-occurrence Graph is therefore built upon this contextual foundation.
- **The Mechanism:** We maintain a global **Contextual Co-occurrence Graph**. The nodes in this graph are the universal tokens from our “Living Codebooks” (e.g., `Vision_Token_123`, `Audio_Token_456`). An edge between two token nodes is created or strengthened only when those tokens co-occur in memories that share the **exact same `agent_id` and `call_id`**.
- **The Query Workflow:** This enables a powerful, noise-free associative query. An agent can ask: “I am

seeing a pattern corresponding to `Vision_Token_123`. What other sensory patterns are most strongly associated with this pattern across all coherent tasks?” The query engine performs a fast graph traversal on the Co-occurrence Graph, following the strongest edges to find the most likely co-occurring tokens from other modalities.

This four-tiered memory system provides a complete cognitive framework. It allows an agent to recall its personal history, search its own experiences for similar patterns, tap into universal human knowledge, and, most powerfully, use the Contextual Co-occurrence Graph to discover the fundamental, cross-modal statistical relationships that are grounded in coherent, task-oriented experiences. This prevents the agent from learning spurious correlations and allows it to build a robust, causal model of its world.

## 5 The Tensor Substrate: A Universal Canvas for Cognition

At the axiomatic core of any computational model of intelligence lies its representation of data. This representation is not a mere implementation detail; it is the ontological commitment of the system. It defines what the agent can perceive, how it can reason, and the ultimate boundaries of its potential understanding. For an agent designed for general intelligence, this representation cannot be specific to any single modality or task. It must be a universal medium, a *lingua franca* for all forms of knowledge. We posit that this medium must be both topologically sound, to faithfully model the inherent structure of sensory data, and computationally versatile, to serve as the foundation for dynamic reasoning.

To this end, we have designed and implemented the **Tensor Substrate**. This is not simply a multi-dimensional array library, but a complete, C-based framework that provides the foundational layer for all cognitive processes within the AGENK system. It is the universal “canvas” upon which all sensory information, perceptual interpretations, and cognitive operations are rendered. The entire substrate is built upon two core, symbiotic abstractions: the **AgenkTensor**, representing materialized, in-memory data, and the **AgenkTensorView**, a

powerful abstraction representing a logical description of data or a computational operation.

### 5.1 Core Insight: Separating Computational Intent from Execution

A primary source of computational and cognitive inefficiency in large-scale data systems is the entanglement of computational intent with immediate execution. Consider an agent’s cognitive process: it might need to analyze a specific region of a video frame, but only after subtracting the previous frame to isolate motion, and then scaling the brightness to normalize for lighting conditions. A naive implementation would execute each of these steps sequentially, creating a chain of large, intermediate, materialized tensors in memory. This approach is profoundly wasteful, analogous to a human artist needing to create a completely new, full-sized physical painting for every single brushstroke modification.

Our architecture is founded on the principle of separating the *description of a computation* from its *execution*. This is the essence of lazy evaluation, a powerful paradigm for building efficient and expressive systems.

- **The AgenkTensor** is the container for materialized, physical data. It is a “heavyweight” object that owns a block of memory containing the actual numerical values that constitute a piece of reality. It is the final *result* of a computation.
- **The AgenkTensorView** is a lightweight, logical descriptor. It is the embodiment of *intent*. It contains no raw data itself. In its simplest form, it is a mathematical “pointer” to data that may reside on disk. More powerfully, we elevate this concept to a node in a \*\*lazy computational graph\*\*. A complex chain of operations is thereby represented as a Directed Acyclic Graph (DAG) of these view objects, which is constructed at virtually no cost.

All high-level cognitive operations, such as attentional focusing (“zoom”) or composing arithmetic transformations, are performed by building a graph of these cheap, lightweight **TensorView** objects. The expensive operation of actually performing the computation and loading the data—a process we term *materialization*—is deferred until the absolute last moment. When the final re-

sult is requested, a “materialization engine” traverses the entire computational graph, performs algebraic simplification where possible, and executes the chain of operations in the most optimal way to produce the final `AgenkTensor`, minimizing intermediate memory allocations. This lazy computation pattern is the cornerstone of the substrate’s power and efficiency.

## 5.2 Architectural Design and Implementation

The entire system is designed for maximum performance, portability (strict ANSI C99), and safety. It is implemented as a new foundational module, `src/tensors/`, which has zero external dependencies, ensuring it can be compiled and run on any platform, from high-performance servers to embedded systems.

### 5.2.1 The AgenkTensor: The Materialized Canvas

The canonical structure for in-memory data is designed for optimal data locality and computational speed. For maximum performance and to reduce memory fragmentation, the `tensor_lifecycle.c` module implements a single-block allocation strategy, where one call to `malloc` reserves a contiguous block of memory for the `AgenkTensor` struct, its metadata arrays (`shape` and `strides`), and the main data buffer.

```

1 // A type-safe enum for all supported
2 // primitive data types
3 typedef enum {
4     AGENK_DATA_TYPE_FLOAT32, // For
5         // embeddings, saliency maps
6     AGENK_DATA_TYPE_UINT8, // For
7         // standard images, raw bytes
8     AGENK_DATA_TYPE_INT16, // For raw
9         // audio (PCM)
10    AGENK_DATA_TYPE_UINT16, // For
11        // token grids
12 } AgenkDataType;
13
14 // The core in-memory tensor object
15 typedef struct AgenkTensor {
16     AgenkDataType data_type; // The type
17         // of each element
18     size_t element_size; // Size of
19         // one element in bytes
20
21     void* data; // Pointer
22         // to the raw, contiguous data
23         // buffer
24     size_t data_size_bytes; // Total
25         // size of the data buffer
26
27     size_t ndim; // Number
28         // of dimensions
29     size_t* shape; // Array of
30         // dimension sizes
31 }
```

```

19     size_t* strides; // Array of
20         // byte-strides for each dimension
21 } AgenkTensor;
```

Listing 20: The `AgenkTensor` structure definition.

A critical component of this design is the `strides` array. A stride for a given dimension is the number of bytes one must skip in memory to get to the next element along that dimension. This single feature allows us to abstract away the memory layout entirely. The address of any element at coordinates  $(c_0, c_1, \dots, c_{n-1})$  can be calculated instantly with the universal formula:

$$\text{address} = \text{base\_ptr} + \sum_{i=0}^{n-1} c_i \cdot \text{stride}_i \quad (3)$$

This enables incredibly fast, generic operations. For example, a matrix transpose is no longer a costly data-shuffling operation; it is an  $O(1)$  metadata operation that simply swaps the values in the `shape` and `strides` arrays. The data itself never moves.

### 5.2.2 The AgenkTensorView: From Logical Spotlight to Computational Node

The view object is the core of our lazy access and computation engine. We first introduce its base case: a logical pointer to persistently stored data. This view acts as a “spotlight,” allowing an agent to define a region of interest without loading it.

```

1 // The base view acts as a "pointer" to
2 // data in the Memory Subsystem
3 typedef struct AgenkTensorView {
4     // Reference to the persistent
5         // source of data
6     uint64_t source_memory_id;
7     uint32_t layer_hash_id; // Hash
8         // of the layer name (e.g., "raw_
9             // sensor_data")
10
11     // The logical structure of this
12         // specific view
13     AgenkDataType data_type;
14     size_t ndim;
15     size_t* shape;
16     size_t* strides;
17
18     // The offset within the source blob
19         // where this view begins
20     size_t offset_bytes;
21 } AgenkTensorView;
```

Listing 21: The base `AgenkTensorView` structure for data access.

This initial structure is intentionally lightweight. To enable dynamic computation, we extend this

concept. A view can represent not just a source of data, but a computational operation to be performed on other views. This transforms the view object into a node in a computational graph, creating a unified structure.

```

1 // Defines the operation this node
2     // represents
3 typedef enum {
4     VIEW_OP_SOURCE,           // Base case:
5         // as defined above
6     VIEW_OP_SLICE,            // Structural:
7         // A sub-region of a parent view
8     VIEW_OP_CONCATENATE,      // Structural:
9         // Concatenation of multiple
10        // parents
11     VIEW_OP_ADD,              // Arithmetic:
12         // Element-wise addition of
13        // parents
14     // ... other primitive operations
15 } AgenkViewOperationType;
16
17 // The unified structure for a node in
18 // the computational graph
19 typedef struct AgenkTensorView {
20     AgenkViewOperationType op_type;
21
22     // Logical properties of the tensor
23     // this view describes
24     AgenkDataType data_type;
25     size_t ndim;
26     size_t* shape;
27     size_t* strides;
28
29     // Links to parent nodes in the
30     // computational graph
31     struct AgenkTensorView** parents;
32     size_t num_parents;
33
34     // Operation-specific parameters
35     void* op_params;
36 } AgenkTensorView;
```

Listing 22: The extended AgenkTensorView for lazy computation.

The public API provides a small, powerful set of primitive operations (e.g., `view_create_add(view_A, view_B)`). These functions build the computational graph by instantiating and linking these view nodes. The final graph is executed by a single function, `view_materialize()`, which traverses the graph and produces a new, perfectly-sized AgenkTensor.

### 5.3 Generalization and Specialization through Composition

The true power of this architecture lies in its ability to support both extreme generalization and extreme specialization through the composition of these simple, abstract primitives. This resolves the classic trade-off between generalist and specialist

models.

- **Generalization:** The system provides a small, universal “instruction set” of tensor operations (`slice`, `add`, etc.). Because every cognitive module uses this same set, and because the operations are defined abstractly on N-dimensional tensors via the stride mechanism, the same core logic can be applied to any modality. A routine that computes a difference by composing views works identically on a video stream for motion detection, on an audio spectrogram for frequency analysis, or on co-occurrence matrices for grammatical analysis. This provides a powerful foundation for cross-domain generalization.

- **Specialization:** An agent’s specialization and intelligence emerge from its learned ability to \*\*compose these primitives into complex computational graphs\*\*. A novice agent might only perform simple, single-step operations. An expert agent, through learning and experience, can dynamically construct a deep and intricate `TensorView` graph to perform a sophisticated, task-specific transformation. For example, it could learn to construct a view representing the operation: “Take the last three video frames, find the difference between them, apply a sharpening filter (a convolution, which is a series of additions and scaled multiplications), and then find the brightest patch in the result.” The agent is not just using pre-defined tools; it is **inventing its own algorithms on the fly** by building these lazy computational graphs.

This substrate provides a stable, efficient, and profoundly versatile canvas. It abstracts away the complexities of memory management and execution logic, enabling higher-level systems to define and compose complex data manipulations dynamically—a crucial capability for any truly generalist learning agent.

### 5.4 Reference Implementation: Anatomy of the Tensor Substrate

To validate our architectural design and provide a concrete instantiation of its principles, we implemented the complete Tensor Substrate as a self-contained C99 module. The implementation prioritizes portability, safety, and performance. This

section provides a detailed analysis of each component file, explaining the key design choices and engineering trade-offs that enable the substrate's power and flexibility.

#### 5.4.1 Directory and Module Structure

A clean separation of concerns is enforced through a strict module structure. The `src/tensors/` module is organized as follows:

```
src/
└── tensors/
    ├── include/
    │   └── agenk_tensor.h
    │       (Public API)
    ├── private/
    │   └── tensor_private.h
    │       (Internal Definitions)
    ├── tensor_lifecycle.c
    │   (Memory Management)
    ├── tensor_views.c
    │   (Graph Construction)
    ├── tensor_ops.c
    │   (Compute Kernels)
    ├── tensor_materialize.c
    │   (Graph Execution)
    └── test/
        └── test_tensors.c
            (Validation Suite)
```

This structure ensures a strong boundary between the public interface and the internal implementation, allowing for future optimizations without breaking dependent systems.

#### 5.4.2 The Public API (`agenk_tensor.h`)

The public API, presented in Listing 23, is the formal contract for the entire AGENK system. Its design is governed by the principle of information hiding.

- Opaque Types:** The core structs, `AgenkTensor` and `AgenkTensorView`, are exposed only as forward-declared ‘type-def’s. This is a critical design choice that prevents users of the library from directly manipulating the internal members of the structs. This allows us to change the internal memory layout of these structs in the future (e.g., to add new metadata fields) without breaking any code in the higher-level Sensation, Perception, or Memory modules. All interaction must occur through the provided API functions.

- Clear Function Naming:** The API is divided into logical sections: Lifecycle, View Management, Computation, and Accessors. The function names are explicit (e.g., `view_create_source`, `view_materialize`) to make the user’s intent clear and the code self-documenting.

This clean, minimal interface provides the stable foundation upon which all other cognitive systems are built.

```
1 /**
2  * @file agenk_tensor.h
3  * @brief Public API for the AGENK
4  * Tensor Substrate.
5  *
6  * This header defines the universal
7  * data structures and functions for
8  * creating,
9  * manipulating, and accessing N-
10 * dimensional data within the AGENK
11 * system.
12 * It is the foundational layer for all
13 * cognitive modules.
14 *
15 * The architecture is built on two core
16 * concepts:
17 * 1. AgenkTensor: A "heavyweight"
18 * struct representing materialized,
19 * in-memory data.
20 * 2. AgenkTensorView: A "lightweight"
21 * struct representing a logical
22 * description
23 * of a tensor or a node in a lazy
24 * computational graph.
25 *
26 * All operations are designed to be
27 * type-safe, memory-safe, and highly
28 * efficient,
29 * adhering to strict ANSI C99 for
30 * maximum portability.
31 *
32 * @version 1.0.0
33 * @author Ankush Yadav, Ankit Yadav,
34 * AuctaSapience
35 */
36
37 #ifndef AGENK_TENSOR_H
38 #define AGENK_TENSOR_H
39
40 #include <stddef.h> // For size_t
41 #include <stdint.h> // For portable
42     integer types like uint64_t
43 #include <stdbool.h>
44
45 //=====
46 // SECTION 1: CORE ENUMS
47 // AND TYPE DEFINITIONS
48 //=====
49
50 /**
51 * @enum AgenkDataType
52 * @brief Defines the primitive data
53 * type of the elements within a
54 * tensor.
55 */
56
57 typedef enum {
```

```

38     AGENK_DATA_TYPE_UNDEFINED = 0,
39     AGENK_DATA_TYPE_FLOAT32, // For
40         embeddings, saliency maps,
41         continuous data
40     AGENK_DATA_TYPE_UINT8, // For
41         standard 8-bit images, raw bytes
41     AGENK_DATA_TYPE_INT16, // For 16-
42         bit raw audio (PCM)
42     AGENK_DATA_TYPE_UINT16, // For
43         token grids from the Pattern
43         Engine
43 } AgenkDataType;
44 /**
45 * @enum AgenkContentType
46 * @brief Defines the LOGICAL
47     interpretation of the data within a
48     tensor.
48 * This is orthogonal to the physical
48     data type.
49 */
50 typedef enum {
51     AGENK_CONTENT_TYPE_UNDEFINED = 0, //
51         Content is unknown or generic
52     AGENK_CONTENT_TYPE_RAW_BYTES, //
52         Generic byte data
53     AGENK_CONTENT_TYPE_IMAGE_PIXELS, //
53         Data represents image pixels
54     AGENK_CONTENT_TYPE_AUDIO_PCM, //
54         Data represents raw audio
54         samples
55     AGENK_CONTENT_TYPE_UTF8_TEXT, //
55         Data is a sequence of UTF-8
55         characters
56     AGENK_CONTENT_TYPE_TOKEN_IDS, //
56         Data is a sequence of learned
56         token IDs
57     AGENK_CONTENT_TYPE_EMBEDDING // Data is a semantic embedding
57         vector
58 } AgenkContentType;
59 /**
60 * @enum AgenkViewOperationType
61 * @brief Defines the operation a
61     TensorView node represents in the
61     computational graph.
63 */
64 typedef enum {
65     VIEW_OP_SOURCE, // Base case:
65         A view of data in persistent
65         memory
66     VIEW_OP_SLICE, // Structural:
66         A sub-region of a parent view
67     VIEW_OP_CONCATENATE, // Structural:
67         Concatenation of multiple
67         parents
68     VIEW_OP_TRANSPOSE, // Structural:
68         Permutation of axes
69     VIEW_OP_BROADCAST, // Structural:
69         Expands a dimension of size 1
70     VIEW_OP_ADD, // Arithmetic:
70         Element-wise addition of
70         parents
71     VIEW_OP_SUBTRACT, // Arithmetic:
71         Element-wise subtraction
72     VIEW_OP_MULTIPLY, // Arithmetic:
72         Element-wise multiplication
73     VIEW_OP_SCALE // Arithmetic:
73         Multiplication by a scalar
74 } AgenkViewOperationType;
75 /**
76 * @brief Opaque forward declarations
76     for the core data structures.
77 * The full definitions are hidden in
77     private headers to enforce a clean
77     API.
79 * Users of this API should only ever
79     interact with pointers to these
79     structs.
80 */
81 typedef struct AgenkTensor AgenkTensor;
82 typedef struct AgenkTensorView
82     AgenkTensorView;
83 //=====
84 // SECTION 2: TENSOR
85 // LIFECYCLE MANAGEMENT
86 //=====
88 /**
89 * @brief Creates a new, contiguous,
89     zero-initialized in-memory tensor.
90 *
91 * This function performs a single
91     memory allocation for the tensor
91     struct, its
91     metadata (shape, strides), and the
91     data buffer itself to improve data
91     locality
91     and reduce allocation overhead.
92 *
93 * @param shape An array of sizes for
93     each dimension. Must not be NULL.
93 * @param ndim The number of dimensions
93     (the length of the shape array).
93     Must be > 0.
94 * @param type The primitive data type
94     of each element in the tensor.
95 * @return A pointer to the new
95     AgenkTensor, or NULL on allocation
95     failure or invalid input.
96 *
96     The caller is responsible for
96     freeing the tensor with tensor_
96     free().
97 */
98 AgenkTensor* tensor_create(const size_t*
98     shape, size_t ndim, AgenkDataType
98     type, AgenkContentType content);
99 /**
100 * @brief Frees all memory associated
100     with an AgenkTensor created by
100     tensor_create().
101 * @param tensor A pointer to the tensor
101     to be freed. Can safely be called
101     on NULL.
102 */
103 void tensor_free(AgenkTensor* tensor);
104 //=====
105 // SECTION 3: TENSOR
106 // VIEW MANAGEMENT
106 //=====
107 /**
108 * @brief Creates a base-case "source"
108     view, a logical pointer to data in
108     memory.
109 */
110
111 //=====
112 // SECTION 3: TENSOR
113 // VIEW MANAGEMENT
113 //=====
114 /**
115 * @brief Creates a base-case "source"
115     view, a logical pointer to data in
115     memory.
116 */
117

```

```

118     persistent memory.  

119     * This is the root node for many  

120       computational graphs.  

121     *  

122     * @param memory_id The unique ID of the  

123       MemoryObject in the database.  

124     * @param layer_name A string identifier  

125       for the layer within the  

126       MemoryObject (e.g., "raw_sensor_  

127         data").  

128     * @return A pointer to the new  

129       AgenkTensorView, or NULL on failure  

130     .  

131     * The caller is responsible for  

132       freeing the view with view_free().  

133   */  

134   AgenkTensorView* view_create_source(  

135     uint64_t memory_id, const char*  

136     layer_name);  

137  

138   /**  

139     * @brief Frees a TensorView object and  

140       any resources it owns.  

141     * Note: This does NOT recursively free  

142       parent views. Each view must be  

143       managed independently.  

144     * @param view A pointer to the view to  

145       be freed. Can safely be called on  

146       NULL.  

147   */  

148   void view_free(AgenkTensorView* view);  

149  

150   // --- Primitive Operations for building  

151   // the computational graph ---  

152  

153   /**  

154     * @brief Creates a new view  

155       representing an element-wise  

156       addition of two parent views.  

157     * The shapes of the parent views must  

158       be compatible for broadcasting.  

159     * @param view_a The first parent view.  

160     * @param view_b The second parent view.  

161     * @return A new AgenkTensorView  

162       representing the lazy addition, or  

163       NULL on failure.  

164   */  

165   AgenkTensorView* view_create_add(const  

166     AgenkTensorView* view_a, const  

167     AgenkTensorView* view_b);  

168  

169   /**  

170     * @brief Creates a new view that  

171       represents a slice of a parent view  

172     .  

173     * This is a lazy, metadata-only  

174       operation.  

175     * @param parent The parent view to  

176       slice.  

177     * @param offsets An array of starting  

178       indices for the slice in each  

179       dimension.  

180     * @param shape An array of sizes for  

181       the slice in each dimension.  

182     * @return A new AgenkTensorView, or  

183       NULL on failure.  

184   */  

185   AgenkTensorView* view_create_slice(const  

186     AgenkTensorView* parent, const size_  

187     _t* offsets, const size_t* shape);  

188  

189   /**  

190     * @brief Creates a new view  

191       representing an element-wise  

192       subtraction.  

193   */  

194   AgenkTensorView* view_create_subtract(  

195     const AgenkTensorView* view_a, const  

196     AgenkTensorView* view_b);  

197  

198   /**  

199     * @brief Creates a new view  

200       representing an element-wise  

201       multiplication.  

202   */  

203   AgenkTensorView* view_create_multiply(  

204     const AgenkTensorView* view_a, const  

205     AgenkTensorView* view_b);  

206  

207   /**  

208     * @brief Creates a new view  

209       representing the scaling of a  

210       parent view by a scalar.  

211     * @param parent The parent view to  

212       scale.  

213     * @param scalar The float value to  

214       scale by.  

215     * @return A new AgenkTensorView, or  

216       NULL on failure.  

217   */  

218   AgenkTensorView* view_create_scale(const  

219     AgenkTensorView* parent, float  

220     scalar);  

221  

222   //=====  

223   // SECTION 4: TENSOR  

224   // COMPUTATION & DATA ACCESS  

225   //=====  

226  

227   /**  

228     * @brief Materializes a view into a new  

229       , concrete in-memory tensor.  

230     *  

231     * This is the primary execution  

232       function. It traverses the  

233       computational graph  

234     * described by the view, performs the  

235       necessary database lookups and  

236       computations,  

237     * and returns the final result in a new  

238       , contiguous AgenkTensor.  

239     *  

240     * @param view The logical view  

241       describing the data and operations.  

242       Must not be NULL.  

243     * @return A new AgenkTensor containing  

244       the final computed data, or NULL on  

245       failure.  

246     * The caller is responsible for  

247       freeing this tensor with tensor_  

248       free().  

249   */  

250   AgenkTensor* view_materialize(const  

251     AgenkTensorView* view);  

252  

253   //=====  

254   // SECTION 5: PUBLIC ACCESSORS  

255

```

```

196 //=====
197 /**
198 * @brief Retrieves the content type of
199 * the specified tensor.
200 *
201 * This function returns the content
202 * type associated with the given
203 * tensor,
204 * which indicates the type of data
205 * stored within the tensor (e.g.,
206 * float, int).
207 *
208 * @param tensor Pointer to the
209 * AgenkTensor whose content type is
210 * to be retrieved.
211 * @return The content type of the
212 * tensor as an AgenkContentType value
213 *
214 */
215 AgenkContentType tensor_get_content_type(
216     const AgenkTensor* tensor);
217
218 /**
219 * @brief Gets the number of dimensions
220 * of a materialized tensor.
221 * @param tensor A non-NULL,
222 * materialized tensor.
223 * @return The number of dimensions.
224 */
225 size_t tensor_get_ndim(const AgenkTensor*
226     * tensor);
227
228 /**
229 * @brief Gets a pointer to the shape
230 * array of a materialized tensor.
231 * @param tensor A non-NULL,
232 * materialized tensor.
233 * @return A read-only pointer to the
234 * shape array. The lifetime of this
235 * pointer
236 *      is tied to the lifetime of
237 * the tensor. Do not free it.
238 */
239 const size_t* tensor_get_shape(const
240     AgenkTensor* tensor);
241
242 /**
243 * @brief Retrieves the data type of the
244 * specified tensor.
245 *
246 * This function returns the data type
247 * associated with the given tensor.
248 *
249 * @param tensor Pointer to the
250 * AgenkTensor whose data type is to
251 * be retrieved.
252 * @return The data type of the tensor
253 * as an AgenkDataType enum value.
254 */
255 AgenkDataType tensor_get_data_type(const
256     AgenkTensor* tensor);
257
258 /**
259 * @brief Gets a pointer to a specific
260 * element within a materialized
261 * tensor.
262 *
263 * This is typically implemented as a
264 * static inline function for maximum
265 * performance,
266 * allowing the compiler to eliminate
267 * function call overhead.
268 *
269 * @param tensor A non-NULL,
270 * materialized tensor.
271 * @param coords An array of coordinates
272 * , one for each dimension.
273 * @return A void pointer to the
274 * requested element. The caller must
275 * cast this to the
276 * correct type (e.g., `float*`,
277 * `uint8_t*`). Returns NULL if
278 * inputs
279 *      are invalid or coordinates
280 * are out of bounds.
281 */
282 void* tensor_get_element_ptr(const
283     AgenkTensor* tensor, const size_t*
284     coords);
285
286 /**
287 * @brief Gets a read-only pointer to
288 * the raw data buffer of a
289 * materialized tensor.
290 * @param tensor A non-NULL,
291 * materialized tensor.
292 * @return A const void pointer to the
293 * start of the data buffer.
294 */
295 const void* tensor_get_data_ptr(const
296     AgenkTensor* tensor);
297
298 /**
299 * @brief Gets the total size in bytes
300 * of the tensor's data buffer.
301 * @param tensor A non-NULL,
302 * materialized tensor.
303 * @return The size of the data buffer
304 * in bytes.
305 */
306 size_t tensor_get_data_size_bytes(const
307     AgenkTensor* tensor);
308
309 bool tensor_op_copy(AgenkTensor* dest,
310     const AgenkTensor* src);
311 bool tensor_op_copy_from_view(
312     AgenkTensor* dest, const AgenkTensor*
313     * src_view);
314 bool tensor_op_concatenate(AgenkTensor*
315     dest, const AgenkTensor** srcs, size_
316     _t num_srcs, size_t axis);
317
318#endif // AGENK_TENSOR_H

```

Listing 23: The public API header for the Tensor Substrate.

#### 5.4.3 Internal Definitions and Safety (**tensor\_private.h**)

The private header, shown in Listing 24, is the internal backbone of the module.

- **Full Struct Definitions:** It contains the complete definitions of **struct AgenkTensor** and **struct**

**AgenkTensorView.** A key feature is the inclusion of a ‘magic’ number in each struct. In debug builds, our validation macros check for this number. If an invalid pointer is passed to a function, it will likely read garbage from memory, fail the magic number check, and cause an immediate and obvious assertion failure. This technique is invaluable for catching common C errors like use-after-free or passing uninitialized pointers.

- **Shared Prototypes:** It contains the prototypes for all internal functions, including the computational kernels from `tensor_ops.c`. This ensures that all `'.c'` files within the module have a consistent view of the internal API and allows the compiler to perform type checking across files.
  - **Conditional Debugging Macros:** The `'TENSOR_VALIDATE'` and `'TENSOR_LOG_ERROR'` macros are defined using `'ifndef NDEBUG'`. This is the standard C mechanism for conditional compilation. When compiling in "Debug" mode, these macros expand to rigorous pointer checks and detailed error messages. When compiling in "Release" mode (where `'NDEBUG'` is defined), they compile to absolutely nothing ('`(void)0`'), ensuring zero performance overhead in production.

```
1  /**
2   * @file tensor_private.h
3   * @brief Private header for the AGENK
4   *        Tensor Substrate implementation.
5   *
6   * This header contains the full
7   * definitions of the opaque structs,
8   * internal
9   * helper functions, and macros used by
10  * the various .c files within the
11  * tensor module. It should NOT be
12  * included by any file outside of
13  * this module,
14  * with the exception of the test suite
15  * which requires white-box access.
16  *
17  * @version 1.0.0 (Corrected, Iterator
18  * logic moved to tensor_ops.c)
19  * @author Ankush Yadav, Ankit Yadav,
20  * AuctaSapience
21  */
22
23 #ifndef TENSOR_PRIVATE_H
24 #define TENSOR_PRIVATE_H
25
26 #include "../include/agenk_tensor.h" //
27           Include the public APT
```

```
18 #include <assert.h> //  
19     For debug assertions  
20 #include <stdio.h> //  
21     For logging/error messages  
22  
23 //=====  
24 // SECTION 1: FULL STRUCT DEFINITIONS  
25 //=====  
26  
27 #define AGENK_TENSOR_MAGIC 0xDEADBEEF  
28 #define AGENK_VIEW_MAGIC   0xCAFEBAE  
29  
30 struct AgenkTensor {  
31     AgenkDataType data_type;  
32     AgenkContentType content_type;  
33     size_t element_size;  
34     void* data;  
35     size_t data_size_bytes;  
36     size_t ndim;  
37     size_t* shape;  
38     size_t* strides;  
39     uint32_t magic;  
40 };  
41  
42 struct AgenkTensorView {  
43     AgenkViewOperationType op_type;  
44     AgenkDataType data_type;  
45     AgenkContentType content_type;  
46     size_t ndim;  
47     size_t* shape;  
48     size_t* strides;  
49     struct AgenkTensorView** parents;  
50     size_t num_parents;  
51     void* op_params;  
52     uint32_t magic;  
53     uint64_t source_memory_id;  
54     uint32_t layer_hash_id;  
55     size_t offset_bytes;  
56 };  
57  
58 //=====  
59 // SECTION 2: INTERNAL HELPER  
60 // FUNCTION PROTOTYPES  
61 //=====  
62  
63 size_t internal_get_element_size(  
64     AgenkDataType type);  
65 bool internal_calculate_num_elements(  
66     const size_t* shape, size_t ndim,  
67     size_t* out_num_elements);  
68 uint32_t internal_hash_string(const char* str);  
69  
70 //=====  
71 // SECTION 3: INTERNAL COMPUTATIONAL  
72 // KERNEL PROTOTYPES  
73 //=====  
74  
75 // --- Arithmetic Kernels ---  
76 bool tensor_op_add(AgenkTensor* dest,  
77     const AgenkTensor* src_a, const  
78     AgenkTensor* src_b);  
79 bool tensor_op_subtract(AgenkTensor*  
80     dest, const AgenkTensor* src_a,  
81     const AgenkTensor* src_b);  
82 bool tensor_op_multiply(AgenkTensor*  
83     dest, const AgenkTensor* src_a,  
84     const AgenkTensor* src_b);  
85 bool tensor_op_scale(AgenkTensor* dest,  
86     const AgenkTensor* src, float scalar)
```

```

    );
75 // --- Structural Kernels ---
76 bool tensor_op_copy_from_view(
77     AgenkTensor* dest, const AgenkTensor
78     * src_view);
79 bool tensor_op_concatenate(AgenkTensor*
80     dest, const AgenkTensor** srcs, size
81     _t num_srcs, size_t axis);
82 // --- Non-Owning Slice Helpers ---
83 AgenkTensor* tensor_slice(const
84     AgenkTensor* parent, const size_t*
85     offsets, const size_t* shape);
86 void tensor_slice_free(AgenkTensor*
87     slice);

88 //=====
89 // SECTION 4: DEBUGGING AND
90 // ERROR HANDLING MACROS
91 //=====

92 #ifndef NDEBUG
93     #define TENSOR_LOG_ERROR(format,
94         ...) \
95         fprintf(stderr, "[TENSOR_ERROR] %s:%d: " format "\n", FILE
96         __, __LINE__, ##VA_ARGS__)
97
98     #define TENSOR_ASSERT(condition,
99         message) \
100        do { \
101            if (!(condition)) { \
102                TENSOR_LOG_ERROR("Assertion failed: %s",
103                    message); \
104                assert(condition); \
105            } \
106        } while (0)
107
108     #define TENSOR_VALIDATE(tensor) \
109        do { \
110            TENSOR_ASSERT(tensor != NULL,
111                "Tensor pointer is NULL."); \
112            TENSOR_ASSERT(((AgenkTensor
113                *)tensor)->magic == AGENK_TENSOR_MAGIC,
114                "Invalid tensor pointer (bad magic number or use-
115                after-free)."); \
116        } while (0)
117
118     #define VIEW_VALIDATE(view) \
119        do { \
120            TENSOR_ASSERT(view != NULL,
121                "View pointer is NULL.");
122            TENSOR_ASSERT(((AgenkTensorView*)view)->
123                magic == AGENK_VIEW_
124                MAGIC, "Invalid view pointer (bad magic
125                number or use-after-free)."); \
126        } while (0)
127
128 #else // Release build
129     #define TENSOR_LOG_ERROR(format,

```

```

130     ...) ((void)0)
131     #define TENSOR_ASSERT(condition,
132         message) ((void)0)
133     #define TENSOR_VALIDATE(tensor) ((void)
134         (void)0)
135     #define VIEW_VALIDATE(view) ((void)
136         (void)0)
137 #endif // NDEBUG
138
139 #endif // TENSOR_PRIVATE_H

```

Listing 24: The private internal header for the Tensor Substrate.

#### 5.4.4 Memory Management [\(tensor\\_lifecycle.c\)](#)

This file, shown in Listing 25, handles the physical memory for materialized tensors.

- Single-Block Allocation:** The `tensor_create` function implements a critical performance optimization. Instead of performing multiple small ‘malloc’ calls (one for the struct, one for the shape, one for the data), it calculates the total required memory and performs a single, large allocation. It then partitions this block using pointer arithmetic. This has two major benefits:
  - Data Locality:** All data related to a single tensor is contiguous in RAM. This dramatically improves CPU cache performance, as a cache line fetch that loads the tensor’s metadata will often also prefetch the beginning of its raw data.
  - Reduced Overhead:** ‘malloc’ calls have non-trivial overhead. Reducing hundreds of potential allocations to one significantly speeds up the creation of many small tensors. It also simplifies deallocation, as ‘`tensor_free`’ becomes a single, safe call to ‘`free`’.
- Overflow Safety:** All size calculations are carefully checked for integer overflow before memory is allocated, preventing a common and dangerous security vulnerability.

```

1 /**
2  * @file tensor_lifecycle.c
3  * @brief Implements the creation and
4  * destruction of materialized
5  * AgenkTensors.
6 *
7  * This file contains the core memory
8  * management logic for the Tensor
9  * Substrate.

```

```

6  * The primary design goal is
7   * performance and data locality,
8   * achieved through
9   *
10  * a single-block allocation strategy.
11  * All public functions are defined in
12  * agenk_tensor.h.
13  *
14 #include "private/tensor_private.h" //      54 //=====
15 // Includes public header, asserts,
16 // helpers
17 #include <stdlib.h>                      //      55 AgenkTensor* tensor_create(const size_t*
18 // For malloc, calloc, free           56     shape, size_t ndim, AgenkDataType
19 #include <string.h>                      //      57     type, AgenkContentType content) {
20 // For memcpy                         58     // --- 1. Input Validation ---
21 //=====                                59     if (!shape || ndim == 0) {
22 // SECTION 1: INTERNAL HELPER          60         TENSOR_LOG_ERROR("Invalid
23 // FUNCTION IMPLEMENTATIONS           61         arguments: shape is NULL or
24 //=====                                62         ndim is 0.");
25 // This function is declared in tensor_ 63         return NULL;
26 // private.h and used by multiple files 64     }
27 // in this module.                    65     size_t element_size = internal_get_
28 size_t internal_get_element_size(        66         element_size(type));
29     AgenkDataType type) {              67     if (element_size == 0) {
30         switch (type) {               68         TENSOR_LOG_ERROR("Invalid data
31             case AGENK_DATA_TYPE_FLOAT32: 69             type specified: %d", type);
32                 return sizeof(float);    70         return NULL;
33             case AGENK_DATA_TYPE_UINT8:    71     }
34                 return sizeof(uint8_t); 72     }
35             case AGENK_DATA_TYPE_INT16:   73     // --- 2. Calculate Memory Layout
36                 return sizeof(int16_t); 74     size_t num_elements;
37             case AGENK_DATA_TYPE_UINT16: 75     if (!internal_calculate_num_elements(
38                 return sizeof(uint16_t); 76         (shape, ndim, &num_elements)) {
39             default: return 0; // Invalid 77             TENSOR_LOG_ERROR("Shape
40                     type");           78             calculation would overflow
41                     type);           79             size_t.");
42             }                           80             return NULL;
43         }                           81     }
44     }                           82     // Calculate the size of each part
45     // This function is also declared in 83     // of the single memory block
46     // tensor_private.h                84     const size_t tensor_struct_bytes =
47 bool internal_calculate_num_elements( 85         sizeof(AgenkTensor);
48     const size_t* shape, size_t ndim, 86     const size_t shape_bytes = ndim *
49     size_t* out_num_elements) {        87         sizeof(size_t);
50         size_t num_elements = 1;       88     const size_t strides_bytes = ndim *
51         for (size_t i = 0; i < ndim; ++i) { 89         sizeof(size_t);
52             // Check for potential overflow 90     const size_t data_bytes = num_
53             before multiplication.        91         elements * element_size;
54             // If shape[i] is 0, the total 92     }
55             will be 0, which is fine.    93     if (!block) {
56             if (shape[i] > 0 && num_elements 94         TENSOR_LOG_ERROR("Failed to
57             > (SIZE_MAX / shape[i])) {    95         allocate %zu bytes for
58                 *out_num_elements = 0;  tensor.", total_bytes);
59                 return false; // Overflow 96         return NULL;
60                     detected        97     }
61         }                           98     }
62         num_elements *= shape[i]; 99 }
63     *out_num_elements = num_elements;100 }
64     return true;                   101 }
65 }                               102 }
66 //=====                                103 }
67 // SECTION 2: PUBLIC API              104 }
68 // FUNCTION IMPLEMENTATIONS           105 }
```

```

96
97 // --- 4. Setup Internal Pointers
98 // and Metadata ---
99 // The main struct is at the start
100 // of the block.
101 AgenkTensor* tensor = (AgenkTensor*)
102     block;
103
104 // The metadata arrays (shape,
105 // strides) follow immediately
106 // after the struct.
107 tensor->shape = (size_t*)(block +
108     sizeof(AgenkTensor));
109 tensor->strides = (size_t*)(block +
110     sizeof(AgenkTensor) + shape_
111     bytes);
112
113 // The raw data buffer is at the
114 // very end.
115 tensor->data = (void*)(block +
116     sizeof(AgenkTensor) + shape_
117     bytes + strides_bytes);
118
119 // Populate the struct's fields
120 tensor->magic = AGENK_TENSOR_MAGIC;
121     // For debug validation
122 tensor->nndim = ndim;
123 tensor->data_type = type;
124 tensor->content_type = content; // 
125     <<<-- SET THE NEW FIELD
126 tensor->element_size = element_size;
127 tensor->data_size_bytes = data_bytes
128     ;
129
130 // Copy the user-provided shape into
131 // our allocated space.
132 memcpy(tensor->shape, shape, shape_
133     bytes);
134
135 // --- 5. Calculate Strides ---
136 // Calculate strides for a standard
137 // C-style (row-major) contiguous
138 // memory layout.
139 // The last dimension's stride is
140 // simply the size of one element.
141 // Each preceding dimension's stride
142 // is the stride of the next
143 // dimension
144 // multiplied by the size of the
145 // next dimension.
146 if (ndim > 0) {
147     tensor->strides[ndim - 1] =
148         element_size;
149     for (int i = (int)ndim - 2; i >=
150         0; --i) {
151         tensor->strides[i] = tensor
152             ->strides[i + 1] * shape
153                 [i + 1];
154     }
155 }
156
157     return tensor;
158 }
159
160 void tensor_free(AgenkTensor* tensor) {
161     // It is safe to call free() on a
162     // NULL pointer.
163     if (tensor) {
164         // In debug builds, we can
165         // validate the pointer before
166         // freeing.
167         // This helps catch double-free
168         // bugs and other memory
169         // corruption issues.
170         TENSOR_VALIDATE(tensor);
171
172         // Because we used a single-
173         // block allocation, we only
174         // need one call to free().
175         free(tensor);
176     }
177 }
```

Listing 25: Implementation of tensor memory management.

#### 5.4.5 Computational Graph Construction (`tensor_views.c`)

This file is the "planner" of the substrate. Its functions, shown in Listing 26, build the lazy computational graph.

- No Computation:** The core principle of this file is that no function performs any heavy computation or I/O. Functions like `view_create_add` are lightweight. They only allocate the small `AgenkTensorView` struct and perform mathematical calculations on the metadata (shapes and strides) of their parents.
- Broadcasting Logic:** The internal 'get\_broadcast\_shape' helper function implements the full, rigorous broadcasting rules common to professional numerical libraries. This is the key to the substrate's generality, allowing arithmetic operations on tensors of different but compatible shapes.

```

1 /**
2 * @file tensor_views.c
3 * @brief Implements the creation and
4 *        management of lazy AgenkTensorViews
5 *        , which
6 *        form a computational graph
7 *        with full broadcasting and
8 *        parameter support.
9 *
10 * This file contains the logic for
11 * building the lazy computational
12 * graph.
13 * All functions that create views are
14 * designed to be extremely
15 * lightweight,
16 * performing only metadata calculations
17 * and memory allocation for the
18 * small view structs themselves. They
19 * do not perform any I/O or heavy
20 * computation. They define the "intent"
21 * of a computation, which is
22 * later executed by the materializer.
23 *
```

```

13 * @version 1.0.0
14 * @author Ankush Yadav, Ankit Yadav,
15 AuctaSapience
16 */
17 #include "private/tensor_private.h"
18 #include <stdlib.h>
19 #include <string.h>
20 //=====
21 // SECTION 1: INTERNAL
22 // HELPER FUNCTIONS
23 //=====
24
25 // Stub function for database
26 // interaction.
27 // This will be replaced by the real
28 // Memory subsystem.
29 static bool get_metadata_from_memory(
30     uint64_t memory_id, const char*
31     layer_name, AgenkDataType* out_type,
32     AgenkContentType* out_content, size_
33     _t* out_ndim, size_t** out_shape) {
34     // --- STUB IMPLEMENTATION for
35     // testing ---
36     if (memory_id == 123 && strcmp(layer_
37     _name, "raw_sensor_data") == 0)
38     {
39         *out_type = AGENK_DATA_TYPE_
40             UINT8;
41         *out_content = AGENK_CONTENT_
42             TYPE_IMAGE_PIXELS; // Assign
43             // a logical type
44         *out_ndim = 3;
45         *out_shape = (size_t*)malloc(3 *
46             sizeof(size_t));
47         if (!*out_shape) return false;
48         (*out_shape)[0] = 100; (*out_
49             shape)[1] = 200; (*out_shape
50             )[2] = 3;
51         return true;
52     }
53     // Test cases for arithmetic tests
54     if (memory_id == 1 || memory_id ==
55         2) {
56         *out_type = AGENK_DATA_TYPE_
57             FLOAT32;
58         *out_content = AGENK_CONTENT_
59             TYPE_EMBEDDING; // Assign a
60             // logical type
61         *out_ndim = 2;
62         *out_shape = (size_t*)malloc(2 *
63             sizeof(size_t));
64         if (!*out_shape) return false;
65         (*out_shape)[0] = 10; (*out_
66             shape)[1] = 20;
67         return true;
68     }
69     TENSOR_LOG_ERROR("MEMORY_STUB:_
70         Unknown memory ID %lu or layer_
71         '%s'", (unsigned long long)
72         memory_id, layer_name);
73     return false;
74 }
75 // Simple string hashing function (djb2)
76 uint32_t internal_hash_string(const char
77     * str) {
78     unsigned long hash = 5381;
79     int c;
80
81     if (!str) return 0;
82     while ((c = *str++)) {
83         hash = ((hash << 5) + hash) + c;
84     }
85     return (uint32_t)hash;
86 }
87 /**
88 * @brief A generic helper to allocate a
89 // new view and its metadata arrays.
90 * Uses a single-block allocation for
91 // efficiency.
92 */
93 static AgenkTensorView* allocate_view(
94     size_t ndim) {
95     size_t shape_bytes = ndim * sizeof(
96         size_t);
97     size_t strides_bytes = ndim * sizeof(
98         size_t);
99     size_t total_bytes = sizeof(
100        AgenkTensorView) + shape_bytes +
101        strides_bytes;
102
103     char* block = (char*)calloc(1, total_
104         bytes);
105     if (!block) {
106         TENSOR_LOG_ERROR("Failed to
107             allocate %zu bytes for view.
108             ", total_bytes);
109         return NULL;
110     }
111
112     AgenkTensorView* view = (
113         AgenkTensorView*)block;
114     view->shape = (size_t*)(block +
115         sizeof(AgenkTensorView));
116     view->strides = (size_t*)(block +
117         sizeof(AgenkTensorView) + shape_
118         bytes);
119     view->magic = AGENK_VIEW_MAGIC;
120     view->ndim = ndim;
121
122     return view;
123 }
124 /**
125 * @brief The core broadcasting logic.
126 * Determines if two shapes are
127 // broadcast-compatible and calculates
128 // the resulting shape.
129 */
130 static bool get_broadcast_shape(const
131     size_t* shape_a, size_t ndim_a,
132     const size_t* shape_b, size_t ndim_b
133     , size_t** out_shape, size_t* out_
134     ndim) {
135     *out_ndim = (ndim_a > ndim_b) ? ndim
136         _a : ndim_b;
137     *out_shape = (size_t*)malloc(*out_
138         ndim * sizeof(size_t));
139     if (!*out_shape) return false;
140
141     for (size_t i = 0; i < *out_ndim; ++
142         i) {
143         size_t idx_a = ndim_a > i ? ndim
144             _a - 1 - i : (size_t)-1;
145         size_t idx_b = ndim_b > i ? ndim
146             _b - 1 - i : (size_t)-1;
147
148         size_t dim_a = (idx_a != (size_t)
149             -1) ? shape_a[idx_a] : 1;
150         size_t dim_b = (idx_b != (size_t)
151             -1) ? shape_b[idx_b] : 1;
152
153         if (dim_a != dim_b) {
154             if (ndim_a > ndim_b) {
155                 if (ndim_a == ndim_b + 1) {
156                     if (shape_a[ndim_b] == 1) {
157                         *out_shape[i] = shape_b[i];
158                     } else {
159                         TENSOR_LOG_ERROR("Shapes
160                             are not broadcast-compatible.
161                             ");
162                         return false;
163                     }
164                 } else {
165                     TENSOR_LOG_ERROR("Shapes
166                         are not broadcast-compatible.
167                         ");
168                     return false;
169                 }
170             } else {
171                 TENSOR_LOG_ERROR("Shapes
172                     are not broadcast-compatible.
173                     ");
174                 return false;
175             }
176         } else {
177             *out_shape[i] = shape_a[i];
178         }
179     }
180
181     return true;
182 }

```

```

102     )-1) ? shape_a[idx_a] : 1;
103     size_t dim_b = (idx_b != (size_t)
104         )-1) ? shape_b[idx_b] : 1;
105
106     if (dim_a != dim_b && dim_a != 1
107         && dim_b != 1) {
108         TENSOR_LOG_ERROR("Shapes are
109             not broadcast-
110             compatible.");
111         free(*out_shape);
112         *out_shape = NULL;
113         return false;
114     }
115     (*out_shape)[*out_ndim - 1 - i]
116         = (dim_a > dim_b) ? dim_a :
117             dim_b;
118 }
119 return true;
120
121 //=====
122 // SECTION 2: PUBLIC API
123 // - VIEW LIFECYCLE
124 //=====
125 AenkTensorView* view_create_source(
126     uint64_t memory_id, const char*
127     layer_name) {
128     if (!layer_name) return NULL;
129
130     AenkDataType data_type;
131     AenkContentType content_type; // <<-- Get the content type
132     size_t ndim;
133     size_t* shape = NULL;
134     if (!get_metadata_from_memory(memory
135         _id, layer_name, &data_type, &
136         content_type, &ndim, &shape)) {
137         return NULL;
138     }
139
140     AenkTensorView* view = allocate_
141         view(ndim);
142     if (!view) {
143         free(shape);
144         return NULL;
145     }
146
147     view->op_type = VIEW_OP_SOURCE;
148     view->data_type = data_type;
149     view->content_type = content_type;
150     // <<-- Store the content type
151     memcpy(view->shape, shape, ndim *
152         sizeof(size_t));
153     free(shape);
154
155     view->source_memory_id = memory_id;
156     view->layer_hash_id = internal_hash_
157         string(layer_name);
158
159     size_t element_size = internal_get_
160         element_size(data_type);
161     if (ndim > 0) {
162         view->strides[ndim - 1] =
163             element_size;
164         for (int i = (int)ndim - 2; i >=
165             0; --i) {
166             view->strides[i] = view->
167                 strides[i + 1] * view->
168                     strides[i];
169         }
170     }
171
172     return view;
173 }
174
175 void view_free(AenkTensorView* view) {
176     if (view) {
177         VIEW_VALIDATE(view);
178         if (view->op_params) free(view->
179             op_params);
180         if (view->parents) free(view->
181             parents);
182         free(view);
183     }
184 }
185 //=====
186 // SECTION 3: PUBLIC API -
187 // COMPUTATIONAL GRAPH PRIMITIVES
188 //=====
189 AenkTensorView* view_create_slice(const
190     AenkTensorView* parent, const size
191     t* offsets, const size_t* shape) {
192     if (!parent || !offsets || !shape)
193         return NULL;
194     VIEW_VALIDATE(parent);
195
196 #ifndef NDEBUG
197     for (size_t i = 0; i < parent->ndim;
198         ++i) {
199         if ((offsets[i] + shape[i]) >
200             parent->shape[i]) {
201             TENSOR_LOG_ERROR("Slice is u
202                 out of bounds for
203                 dimension %zu.", i);
204             return NULL;
205         }
206     }
207 #endif
208
209     AenkTensorView* view = allocate_
210         view(parent->ndim);
211     if (!view) return NULL;
212
213     view->op_type = VIEW_OP_SLICE;
214     view->data_type = parent->data_type;
215     view->content_type = parent->content
216         _type; // Inherit content type
217     memcpy(view->shape, shape, parent->
218         ndim * sizeof(size_t));
219     memcpy(view->strides, parent->
220         strides, parent->ndim * sizeof(
221             size_t));
222
223     view->op_params = malloc(parent->
224         ndim * sizeof(size_t));
225     if (!view->op_params) { free(view);
226         return NULL; }
227     memcpy(view->op_params, offsets,
228         parent->ndim * sizeof(size_t));
229
230     view->parents = (AenkTensorView**)
231         malloc(sizeof(AenkTensorView*) )
232         ;
233     if (!view->parents) { free(view->op_
234         params); free(view); return NULL
235         ; }

```

```

200     view->parents[0] = (AgenkTensorView
201         *)parent;
202     view->num_parents = 1;
203
204     return view;
205 }
206
207 static AgenkTensorView* view_create_
208     binary_op(const AgenkTensorView*
209     view_a, const AgenkTensorView* view_
210     b, AgenkViewOperationType op_type) {
211     if (!view_a || !view_b) return NULL;
212     VIEW_VALIDATE(view_a); VIEW_VALIDATE
213     (view_b);
214
215     if (view_a->data_type != view_b->
216         data_type) {
217         TENSOR_LOG_ERROR("Operands for
218             binary op must have same
219             data type.");
220         return NULL;
221     }
222
223     size_t* result_shape = NULL;
224     size_t result_ndim = 0;
225     if (!get_broadcast_shape(view_a->
226         shape, view_a->ndim, view_b->
227         shape, view_b->ndim,
228             &result_
229             shape,
230             &result_
231             _ndim))
232     {
233         return NULL;
234     }
235
236     AgenkTensorView* view = allocate_
237         view(result_ndim);
238     if (!view) {
239         free(result_shape);
240         return NULL;
241     }
242
243     view->op_type = op_type;
244     view->data_type = view_a->data_type;
245     // THE FIX: For arithmetic, the
246     // result is usually just raw
247     // numerical data,
248     // unless a more sophisticated rule
249     // is defined.
250     view->content_type = AGENK_CONTENT_
251         TYPE_RAW_BYTES;
252     memcpy(view->shape, result_shape,
253         result_ndim * sizeof(size_t));
254     free(result_shape);
255
256     size_t element_size = internal_get_
257         element_size(view->data_type);
258     if (result_ndim > 0) {
259         view->strides[result_ndim - 1] =
260             element_size;
261         for (int i = (int)result_ndim -
262             2; i >= 0; --i) {
263             view->strides[i] = view->
264                 strides[i + 1] * view->
265                 shape[i + 1];
266         }
267     }
268
269     view->parents = (AgenkTensorView**)
270         malloc(2 * sizeof(
271             AgenkTensorView*));
272     if (!view->parents) { free(view);
273         return NULL; }
274     view->parents[0] = (AgenkTensorView
275         *)view_a;
276     view->parents[1] = (AgenkTensorView
277         *)view_b;
278     view->num_parents = 2;
279
280     return view;
281 }
282
283 AgenkTensorView* view_create_add(const
284     AgenkTensorView* view_a, const
285     AgenkTensorView* view_b) {
286     return view_create_binary_op(view_a,
287         view_b, VIEW_OP_ADD);
288 }
289
290 AgenkTensorView* view_create_subtract(
291     const AgenkTensorView* view_a, const
292     AgenkTensorView* view_b) {
293     return view_create_binary_op(view_a,
294         view_b, VIEW_OP_SUBTRACT);
295 }
296
297 AgenkTensorView* view_create_multiply(
298     const AgenkTensorView* view_a, const
299     AgenkTensorView* view_b) {
300     return view_create_binary_op(view_a,
301         view_b, VIEW_OP_MULTIPLY);
302 }
303
304 AgenkTensorView* view_create_scale(const
305     AgenkTensorView* parent, float
306     scalar) {
307     if (!parent) return NULL;
308     VIEW_VALIDATE(parent);
309
310     AgenkTensorView* view = allocate_
311         view(parent->ndim);
312     if (!view) return NULL;
313
314     view->op_type = VIEW_OP_SCALE;
315     view->data_type = parent->data_type;
316     view->content_type = parent->content
317         _type; // Scaling preserves
318         content type
319     memcpy(view->shape, parent->shape,
320         parent->ndim * sizeof(size_t));
321     memcpy(view->strides, parent->
322         strides, parent->ndim * sizeof(
323             size_t));
324
325     view->op_params = malloc(sizeof(
326         float));
327     if (!view->op_params) { free(view);
328         return NULL; }
329     *(float*)view->op_params = scalar;
330
331     view->parents = (AgenkTensorView**)
332         malloc(sizeof(AgenkTensorView*))
333         ;
334     if (!view->parents) { free(view->op_
335         params); free(view); return NULL
336         ; }
337     view->parents[0] = (AgenkTensorView
338         *)parent;
339     view->num_parents = 1;

```

```

286     return view;
287 }

```

Listing 26: Implementation of the lazy computational graph API.

#### 5.4.6 Computational Kernels and Graph Execution

The final two files, `tensor_ops.c` and `tensor_materialize.c`, are the "engine room" of the substrate.

- **`tensor_ops.c` (Listing 27):** This file contains the optimized, production-grade computational kernels.

– **Generic Iteration:** The arithmetic functions are built around a generic, N-dimensional iteration logic that correctly handles the broadcasting rules computed by the view planner. This ensures that a single, well-tested piece of code can handle any combination of input shapes.

– **Optimized Copying:** The ‘`tensor_op_copy_from_view`’ kernel is a crucial component. It first checks if the source view is contiguous in memory. If so, it uses a single, highly optimized ‘`memcpy`’. If not (e.g., for a transposed view), it falls back to a robust, element-by-element copy that correctly handles any stride layout.

- **`tensor_materialize.c` (Listing 28):**

This file contains the "executor."

– **Recursive Traversal:** The ‘`view_materialize`’ function uses a recursive approach to perform a depth-first traversal of the computational graph. It materializes the parent nodes first before executing the operation of the current node.

– **Resource Management:** The engine is careful to free intermediate tensors as soon as they are no longer needed, minimizing the peak memory usage during the materialization of a complex graph.

```

1 /**
2  * @file tensor_ops.c
3  * @brief Implements production-grade,
4  * generic computational kernels for
5  * AgenkTensors.

```

```

5  * This file provides the core
6  * computational functions that
7  * operate on the raw data
8  * buffers of materialized tensors. It
9  * features a private, generic N-
10 * dimensional
11 * iterator to handle complex
12 * broadcasting rules elegantly and
13 * efficiently. All kernels
14 * support the full range of defined
15 * data types and are designed for
16 * correctness
17 * and performance.
18 */
19 * @version 1.0.0
20 * @author Ankush Yadav, Ankit Yadav,
21 * AuctaSapience
22 */
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43

```

```

44 /**
45 * @brief Initializes a new iterator for
46 * a set of tensors.
47 * The final iteration shape is the
48 * shape of the first tensor (the
49 * destination).
50 * It pre-calculates the broadcast
51 * strides for all source tensors
52 * relative to the destination.
53 */
54 static TensorIterator* iterator_create(
55     const AgenkTensor** tensors, size_t
56     num_tensors) {
57     if (!tensors || num_tensors == 0)
58         return NULL;
59     const AgenkTensor* dest = tensors
60         [0];
61
62     TensorIterator* iter = (
63         TensorIterator*)calloc(1, sizeof
64         (TensorIterator));
65     if (!iter) return NULL;
66
67     iter->num_tensors = num_tensors;
68     iter->ndim = dest->ndim;
69     iter->shape = dest->shape;
70     iter->tensors = tensors;
71
72     iter->ptrs = (void**)malloc(num_
73         tensors * sizeof(void*));
74     iter->coords = (size_t*)calloc(iter
75         ->ndim, sizeof(size_t));
76     iter->b_strides = (size_t**)malloc(
77         num_tensors * sizeof(size_t*));
78     if (!iter->ptrs || !iter->coords ||
79         !iter->b_strides) {
80         free(iter->ptrs); free(iter->
81             coords); free(iter->b_
82             strides); free(iter);
83         return NULL;
84     }
85
86     for (size_t i = 0; i < num_tensors;
87         ++i) {
88         iter->ptrs[i] = tensors[i]->data
89         ;
90         iter->b_strides[i] = (size_t*)
91             malloc(iter->ndim * sizeof(
92                 size_t));
93         if (!iter->b_strides[i]) { /*
94             cleanup needed */ return
95             NULL; }
96
97         for (size_t j = 0; j < iter->
98             ndim; ++j) {
99             size_t src_dim_idx = tensors
100                [i]->ndim > j ? tensors[
101                    i]->ndim - 1 - j : (size
102                    _t)-1;
103             size_t dest_dim_idx = iter->
104                 ndim - 1 - j;
105             if (src_dim_idx != (size_t)
106                 -1 && tensors[i]->shape[
107                     src_dim_idx] == iter->
108                     shape[dest_dim_idx]) {
109                 iter->b_strides[i][dest_
110                     dim_idx] = tensors[i]
111                     ->strides[src_dim_
112                         idx];
113             } else {
114                 iter->b_strides[i][dest_
115                     dim_idx] = 0;
116             }
117         }
118     }
119     return iter;
120 }
121
122 /**
123 * @brief Advances the iterator to the
124 * next element. Returns false when
125 * done.
126 */
127 static bool iterator_next(TensorIterator
128     * iter) {
129     int current_dim = (int)iter->ndim -
130         1;
131     while (current_dim >= 0) {
132         iter->coords[current_dim]++;
133         if (iter->coords[current_dim] <
134             iter->shape[current_dim]) {
135             return true;
136         }
137         iter->coords[current_dim] = 0;
138         current_dim--;
139     }
140     return false;
141 }
142
143 static void iterator_free(TensorIterator
144     * iter) {
145     if (iter) {
146         free(iter->ptrs);
147         free(iter->coords);
148         if (iter->b_strides) {
149             for (size_t i = 0; i < iter
150                 ->num_tensors; ++i) free
151                 (iter->b_strides[i]);
152             free(iter->b_strides);
153         }
154         free(iter);
155     }
156 }
157
158 //=====
159 // SECTION 2: PUBLIC API
160 // IMPLEMENTATIONS
161 //=====
162
163 // --- Accessors ---
164
165 size_t tensor_get_ndim(const AgenkTensor
166     * tensor) {
167     if (!tensor) {
168         return 0;
169     }
170     TENSOR_VALIDATE(tensor);
171     return tensor->ndim;
172 }
173
174 const size_t* tensor_get_shape(const
175     AgenkTensor* tensor) {
176     if (!tensor) {
177         return NULL;
178     }
179     TENSOR_VALIDATE(tensor);
180     return tensor->shape;
181 }
```

```

139 }
140
141 AgenkDataType tensor_get_data_type(const
142     AgenkTensor* tensor) {
143     if (!tensor) {
144         return AGENK_DATA_TYPE_UNDEFINED;
145     }
146     TENSOR_VALIDATE(tensor);
147     return tensor->data_type;
148 }
149
150 AgenkContentType tensor_get_content_type(
151     const AgenkTensor* tensor) {
152     if (!tensor) return AGENK_CONTENT_
153         TYPE_UNDEFINED;
154     TENSOR_VALIDATE(tensor);
155     return tensor->content_type;
156 }
157
158 const void* tensor_get_data_ptr(const
159     AgenkTensor* tensor) {
160     if (!tensor) return NULL;
161     TENSOR_VALIDATE(tensor);
162     return tensor->data;
163 }
164
165 size_t tensor_get_data_size_bytes(const
166     AgenkTensor* tensor) {
167     if (!tensor) return 0;
168     TENSOR_VALIDATE(tensor);
169     return tensor->data_size_bytes;
170 }
171
172 void* tensor_get_element_ptr(const
173     AgenkTensor* tensor, const size_t*
174     coords) {
175     if (!tensor || !coords) {
176         return NULL;
177     }
178     TENSOR_VALIDATE(tensor);
179
180 #ifndef NDEBUG
181     // --- THE FIX: Use the correct
182     // variables for this function's
183     // scope ---
184     for (size_t i = 0; i < tensor->ndim;
185         ++i) {
186         if (coords[i] >= tensor->shape[i])
187             {
188                 TENSOR_LOG_ERROR("Coordinate
189                     %zu(%zu) is out of
190                     bounds for dimension
191                     with size %zu.",
192                     i, coords[i],
193                     tensor->
194                     shape[i])
195             ;
196         }
197         return NULL;
198     }
199 #endif
200
201     char* ptr = (char*)tensor->data;
202     for (size_t i = 0; i < tensor->ndim;
203         ++i) {
204         ptr += coords[i] * tensor->
205             strides[i];
206     }
207
208     return (void*)ptr;
209 }
210
211 // --- Computational Kernels ---
212
213 /**
214  * @brief Core engine for element-wise
215  * binary operations, using the
216  * generic iterator.
217 */
218 static bool tensor_op_binary_broadcast(
219     AgenkTensor* dest, const AgenkTensor
220     * src_a, const AgenkTensor* src_b,
221     char op) {
222     TENSOR_VALIDATE(dest); TENSOR_
223         VALIDATE(src_a); TENSOR_
224         VALIDATE(src_b);
225     if (dest->data_type != src_a->data_
226         type || src_a->data_type != src_
227         b->data_type) return false;
228
229     size_t* coords = (size_t*)calloc(
230         dest->ndim, sizeof(size_t));
231     if (!coords) return false;
232
233     do {
234         // Map destination coordinates
235         // to source coordinates for
236         // broadcasting
237         size_t coords_a_stack[32],
238             coords_b_stack[32];
239         size_t* coords_a = coords_a_
240             stack;
241         size_t* coords_b = coords_b_
242             stack;
243
244         for (size_t i = 0; i < src_a->
245             ndim; ++i) {
246             size_t dest_idx = dest->ndim -
247                 src_a->ndim + i;
248             coords_a[i] = (src_a->shape[
249                 i] == 1) ? 0 : coords[
250                 dest_idx];
251         }
252         for (size_t i = 0; i < src_b->
253             ndim; ++i) {
254             size_t dest_idx = dest->ndim -
255                 src_b->ndim + i;
256             coords_b[i] = (src_b->shape[
257                 i] == 1) ? 0 : coords[
258                 dest_idx];
259         }
260
261         void* p_dest = tensor_get_
262             element_ptr(dest, coords);
263         const void* p_src_a = tensor_get_
264             element_ptr(src_a, coords_a
265             );
266         const void* p_src_b = tensor_get_
267             element_ptr(src_b, coords_b
268             );
269         if (!p_dest || !p_src_a || !p_src_
270             _b) continue;
271
272         // Perform the type-specific
273         // operation
274         switch(dest->data_type) {
275             case AGENK_DATA_TYPE_FLOAT32
276                 :
277                 float v_a = *(const

```

```

228
229     float*)p_src_a;
230     float v_b = *(const
231         float*)p_src_b;
232     if (op == '+') *(float*)
233         p_dest = v_a + v_b;
234     else if (op == '-')
235         *(float*)p_dest = v_
236         a - v_b; else if (op
237         == '*') *(float*)p_
238         dest = v_a * v_b;
239     break;
240 }
241 case AGENK_DATA_TYPE_UINT8:
242 {
243     uint8_t v_a = *(const
244         uint8_t*)p_src_a;
245     uint8_t v_b = *(const
246         uint8_t*)p_src_
247         b;
248     if (op == '+') *(uint8_t*
249         )p_dest = v_a + v_b
250     ; else if (op == '-')
251         *(uint8_t*)p_dest
252         = v_a - v_b; else if
253         (op == '*') *(uint8_
254         _t*)p_dest = v_a * v
255         _b;
256     break;
257 }
258 case AGENK_DATA_TYPE_INT16:
259 {
260     int16_t v_a = *(const
261         int16_t*)p_src_a;
262     int16_t v_b = *(const
263         int16_t*)p_src_
264         b;
265     if (op == '+') *(int16_t*
266         )p_dest = v_a + v_b
267     ; else if (op == '-')
268         *(int16_t*)p_dest
269         = v_a - v_b; else if
270         (op == '*') *(int16_
271         _t*)p_dest = v_a * v
272         _b;
273     break;
274 }
275 case AGENK_DATA_TYPE_UINT16:
276 {
277     uint16_t v_a = *(const
278         uint16_t*)p_src_a;
279     uint16_t v_b = *(const
280         uint16_t*)p_src_
281         b;
282     if (op == '+') *(uint16_
283         _t*)p_dest = v_a + v_
284         b; else if (op == '-')
285         *(uint16_t*)p_
286         dest = v_a - v_b;
287     else if (op == '*')
288         *(uint16_t*)p_dest =
289         v_a * v_b;
290     break;
291 }
292 default: break;
293 }
294
295 int current_dim = (int)dest->
296     ndim - 1;
297 while(current_dim >= 0) {
298     coords[current_dim]++;
299
300     if (coords[current_dim] <
301         dest->shape[current_dim]
302         ]) break;
303     coords[current_dim] = 0;
304     current_dim--;
305 }
306     if (current_dim < 0) break;
307 } while(true);
308
309 free(coords);
310 return true;
311 }
312
313 bool tensor_op_add(AgenkTensor* dest,
314     const AgenkTensor* src_a, const
315     AgenkTensor* src_b) {
316     return tensor_op_binary_broadcast(
317         dest, src_a, src_b, '+');
318 }
319 bool tensor_op_subtract(AgenkTensor*
320     dest, const AgenkTensor* src_a,
321     const AgenkTensor* src_b) {
322     return tensor_op_binary_broadcast(
323         dest, src_a, src_b, '-');
324 }
325 bool tensor_op_multiply(AgenkTensor*
326     dest, const AgenkTensor* src_a,
327     const AgenkTensor* src_b) {
328     return tensor_op_binary_broadcast(
329         dest, src_a, src_b, '*');
330 }
331
332 bool tensor_op_scale(AgenkTensor* dest,
333     const AgenkTensor* src, float scalar
334     ) {
335     TENSOR_VALIDATE(dest); TENSOR_
336         VALIDATE(src);
337     AgenkDataType type = src->data_type;
338     size_t shape[] = {1};
339
340     // THE FIX: Provide the fourth
341     // argument to tensor_create.
342     // We use UNDEFINED as this
343     // temporary tensor has no specific
344     // logical content.
345     AgenkTensor* scalar_tensor = tensor_
346         create(shape, 1, type, AGENK_
347             CONTENT_TYPE_UNDEFINED);
348     if(!scalar_tensor) return false;
349
350     switch(type) {
351         case AGENK_DATA_TYPE_FLOAT32: *(float*)
352             scalar_tensor->data =
353             scalar; break;
354         case AGENK_DATA_TYPE_UINT8: *(uint8_t*)
355             scalar_tensor->data =
356             (uint8_t)scalar; break;
357         case AGENK_DATA_TYPE_INT16: *(int16_t*)
358             scalar_tensor->data =
359             (int16_t)scalar; break;
360         case AGENK_DATA_TYPE_UINT16: *(uint16_t*)
361             scalar_tensor->
362             data = (uint16_t)scalar;
363             break;
364         default: tensor_free(scalar_
365             _tensor); return false;
366     }
367
368     // Reuse our powerful broadcast-
369     // aware multiplication function

```

```

292     bool success = tensor_op_multiply(
293         dest, src, scalar_tensor);
294     tensor_free(scalar_tensor);
295     return success;
296 }
297
298 bool tensor_op_copy_from_view(
299     AgenkTensor* dest, const AgenkTensor*
300     * src_view) {
301     TENSOR_VALIDATE(dest); TENSOR_
302         VALIDATE(src_view);
303     if (dest->ndim != src_view->ndim || 
304         dest->element_size != src_view->
305             element_size) return false;
306     for(size_t i=0; i<dest->ndim; ++i)
307         if(dest->shape[i] != src_view->
308             shape[i]) return false;
309
310     bool is_src_contiguous = true;
311     size_t expected_stride = src_view->
312         element_size;
313     for (int i = (int)src_view->ndim - 
314         1; i >= 0; --i) {
315         if (src_view->shape[i] == 0)
316             continue;
317         if (src_view->strides[i] != 
318             expected_stride) { is_src_
319             contiguous = false; break; }
320         if(src_view->shape[i] > 1)
321             expected_stride *= src_view
322                 ->shape[i];
323     }
324
325     if (is_src_contiguous) {
326         memcpy(dest->data, src_view->
327             data, dest->data_size_bytes)
328         ;
329         return true;
330     }
331
332     size_t* coords = (size_t*)calloc(
333         dest->ndim, sizeof(size_t));
334     if(!coords) return false;
335
336     char* dest_ptr = (char*)dest->data;
337     do {
338         const void* p_src = tensor_get_
339             element_ptr(src_view, coords
340             );
341         memcpy(dest_ptr, p_src, dest->
342             element_size);
343         dest_ptr += dest->element_size;
344         int current_dim = (int)dest->
345             ndim - 1;
346         while(current_dim >= 0) {
347             coords[current_dim]++;
348             if (coords[current_dim] <
349                 dest->shape[current_dim]
350                 ) break;
351             coords[current_dim] = 0;
352             current_dim--;
353         }
354         if (current_dim < 0) break;
355     } while(true);
356
357     free(coords);
358     return true;
359 }
360
361 bool tensor_op_concatenate(AgenkTensor*
362     dest, const AgenkTensor** srcs, size_
363     t num_srcs, size_t axis) {
364     TENSOR_VALIDATE(dest);
365     if (axis >= dest->ndim) return false
366         ;
367
368     size_t dest_axis_offset = 0;
369
370     for (size_t i = 0; i < num_srcs; ++i
371         ) {
372         const AgenkTensor* src = srcs[i
373             ];
374         TENSOR_VALIDATE(src);
375
376         size_t* slice_offsets = (size_t*
377             *)calloc(dest->ndim, sizeof(
378             size_t));
379         if(!slice_offsets) return false;
380         slice_offsets[axis] = dest_axis_
381             offset;
382
383         AgenkTensor* dest_slice_view =
384             tensor_slice(dest, slice_
385                 offsets, src->shape);
386         free(slice_offsets);
387         if (!dest_slice_view) return
388             false;
389
390         tensor_op_copy_from_view(dest_
391             .slice_view, src);
392         tensor_slice_free(dest_slice_
393             view);
394
395         dest_axis_offset += src->shape[
396             axis];
397     }
398
399     return true;
400 }
401
402 // --- Non-Owning Slice Helpers ---
403
404 /**
405  * @brief Creates a new tensor header
406  *        that is a "view" into a slice of a
407  *        parent tensor.
408  *
409  * THIS IS A METADATA-ONLY OPERATION. It
410  * does not copy any of the
411  * underlying
412  * tensor data. The returned tensor's
413  * data pointer will point into the
414  * parent
415  * tensor's data buffer.
416  *
417  * IMPORTANT: The returned tensor is a "
418  * non-owning" view. The caller must
419  * NOT
420  * call tensor_free() on it. The memory
421  * is still managed by the original
422  * parent
423  * tensor. A separate `view_free()`
424  * function would be needed for a
425  * complete view system.
426  *
427  * For now, this function is for
428  * demonstrating the power of strides.
429  * A full implementation would track
430  * memory ownership to prevent double-
431  * frees.
432  */

```

```

379 * @param parent The source tensor to
380 * slice from.
381 * @param offsets An array of starting
382 * indices for the slice in each
383 * dimension.
384 * @param shape An array of sizes for
385 * the slice in each dimension.
386 * @return A new AgenkTensor struct
387 * configured as a view, or NULL on
388 * failure.
389 */
390 AgenkTensor* tensor_slice(const
391     AgenkTensor* parent, const size_t*
392     offsets, const size_t* shape) {
393     if (!parent || !offsets || !shape) {
394         TENSOR_LOG_ERROR("Invalid
395             arguments: parent, offsets,
396             or shape is NULL.");
397         return NULL;
398     }
399     TENSOR_VALIDATE(parent);
400
401 #ifndef NDEBUG
402     for (size_t i = 0; i < parent->ndim;
403         ++i) {
404         if ((offsets[i] + shape[i]) >
405             parent->shape[i]) {
406             TENSOR_LOG_ERROR("Slice[%zu
407                 , %zu] is out of bounds
408                 for dimension %zu with
409                 size %zu.",
410                 offsets[i], offsets[i] +
411                     shape[i], i, parent
412                     ->shape[i]);
413             return NULL;
414         }
415     }
416 #endif
417
418     AgenkTensor* slice = (AgenkTensor*)
419         malloc(sizeof(AgenkTensor));
420     if (!slice) return NULL;
421
422     slice->shape = (size_t*)malloc(
423         parent->ndim * sizeof(size_t));
424     if (!slice->shape) { free(slice);
425         return NULL; }
426
427     slice->strides = (size_t*)malloc(
428         parent->ndim * sizeof(size_t));
429     if (!slice->strides) { free(slice->
430         shape); free(slice); return NULL
431         ; }
432
433     slice->magic = AGENK_TENSOR_MAGIC;
434     slice->ndim = parent->ndim;
435     slice->data_type = parent->data_type
436         ;
437     slice->element_size = parent->
438         element_size;
439
440     memcpy(slice->shape, shape, parent->
441         ndim * sizeof(size_t));
442     memcpy(slice->strides, parent->
443         strides, parent->ndim * sizeof(
444             size_t));
445
446     slice->data = tensor_get_element_ptr
447         (parent, offsets);
448
449     size_t num_elements;
450     internal_calculate_num_elements(
451         slice->shape, slice->ndim, &num_
452         elements);
453     slice->data_size_bytes = num_
454         elements * slice->element_size;
455
456     return slice;
457 }
458
459 void tensor_slice_free(AgenkTensor*
460     slice) {
461     if (slice) {
462         TENSOR_VALIDATE(slice);
463         free(slice->shape);
464         free(slice->strides);
465         free(slice);
466     }
467 }
```

Listing 27: Implementation of the optimized computational kernels.

```

1 /**
2  * @file tensor_materialize.c
3  * @brief Implements the full execution
4  * engine for lazy AgenkTensorView
5  * graphs.
6  *
7  * This file contains the view_
8  * materialize() function, which is
9  * the "compiler"
10 * and "executor" for the computational
11 * graphs built by the view management
12 * API.
13 *
14 * @version 1.0.0
15 * @author Ankush Yadav, Ankit Yadav,
16 * AuctaSapience
17 */
18 //=====
19 // SECTION 1: FORWARD
20 // DECLARATIONS & STUBS
21 //=====
22
23 static AgenkTensor* materialize_
24     recursive(const AgenkTensorView*
25         view);
26
27 // --- STUB for the Memory Subsystem (
28 // Updated for comprehensive testing)
29 // ---
30 static void* memory_stub_get_blob(uint64
31     _t memory_id, uint32_t layer_hash_id
32     , size_t* out_size) {
33     (void)layer_hash_id;
34     char filename[256];
35 }
```

```

30 // Logic to handle multiple test
31     files based on memory_id
32 if (memory_id == 1) {
33     sprintf(filename, sizeof(
34         filename), "test_blob_a.bin"
35     );
36 } else if (memory_id == 2) {
37     sprintf(filename, sizeof(
38         filename), "test_blob_b.bin"
39     );
40 } else if (memory_id == 123) {
41     sprintf(filename, sizeof(
42         filename), "test_blob_123.
43         bin");
44 } else {
45     TENSOR_LOG_ERROR("MEMORY_STUB:_
46         Unknown memory ID %llu", (
47             unsigned long long)memory_id
48         );
49     return NULL;
50 }
51
52 FILE* f = fopen(filename, "rb");
53 if (!f) {
54     TENSOR_LOG_ERROR("MEMORY_STUB:_
55         Failed to open dummy data_
56         file '%s'", filename);
57     return NULL;
58 }
59
60 fseek(f, 0, SEEK_END);
61 long size = ftell(f);
62 fseek(f, 0, SEEK_SET);
63 if (size <= 0) {fclose(f); return
64     NULL; }
65
66 char* buffer = (char*)malloc(size);
67 if (!buffer) {fclose(f); return
68     NULL; }
69
70 if (fread(buffer, 1, size, f) != (
71     size_t)size) {
72     free(buffer);
73     fclose(f);
74     return NULL;
75 }
76
77 fclose(f);
78 *out_size = size;
79 return buffer;
80 }
81
82 //=====
83 // SECTION 2: PUBLIC
84 // API FUNCTION
85 //=====
86
87 AgenkTensor* view_materialize(const
88     AgenkTensorView* view) {
89     if (!view) {
90         TENSOR_LOG_ERROR("Cannot_
91             materialize a NULL view.");
92         return NULL;
93     }
94     VIEW_VALIDATE(view);
95     return materialize_recursive(view);
96 }
97
98 //=====
99 // SECTION 3: THE RECURSIVE
100 // MATERIALIZATION ENGINE
101 //=====
102
103 static AgenkTensor* materialize_
104     recursive(const AgenkTensorView*
105         view) {
106     // --- Base Case: SOURCE node ---
107     if (view->op_type == VIEW_OP_SOURCE)
108     {
109         size_t blob_size;
110         void* data_blob = memory_stub_
111             get_blob(view->source_memory_
112                 id, view->layer_hash_id, &
113                     blob_size);
114         if (!data_blob) return NULL;
115
116         // THE FIX: Pass the view's
117         // content_type to tensor_
118         // create
119         AgenkTensor* materialized_tensor
120             = tensor_create(view->shape
121                 , view->nDim, view->data_
122                     type, view->content_type);
123         if (!materialized_tensor) { free
124             (data_blob); return NULL; }
125
126         if (materialized_tensor->data_
127             size_bytes != blob_size) {
128             TENSOR_LOG_ERROR("Data blob_
129                 size mismatch for memory
130                 ID %llu. Expected %zu, got %zu.", (unsigned long
131                     long)view->source_
132                         memory_id, materialized_
133                             tensor->data_size_bytes,
134                                 blob_size);
135             tensor_free(materialized_
136                 tensor);
137             free(data_blob);
138             return NULL;
139         }
140
141         memcpy(materialized_tensor->data
142             , data_blob, blob_size);
143         free(data_blob);
144         return materialized_tensor;
145     }
146
147     // --- Recursive Step: Materialize
148     // all parent nodes first. ---
149     if (view->num_parents == 0) {
150         TENSOR_LOG_ERROR("Non-source_
151             view has zero parents.");
152         return NULL;
153     }
154
155     AgenkTensor** parent_tensors = (
156         AgenkTensor**)calloc(view->num_
157             parents, sizeof(AgenkTensor*));
158     if (!parent_tensors) return NULL;
159
160     for (size_t i = 0; i < view->num_
161         parents; ++i) {
162         parent_tensors[i] = materialize_
163             recursive(view->parents[i]);
164         if (!parent_tensors[i]) {
165             for (size_t j = 0; j < i; ++
166                 j) tensor_free(parent_
167                     tensors[j]);
168             return NULL;
169         }
170     }
171
172     return materialized_tensor;
173 }
```

```

124         tensors[j]);
125     free(parent_tensors);
126     return NULL;
127 }
128
129 // --- Execute the Operation for the
130 // Current Node ---
131 AgenkTensor* dest_tensor = tensor_
132     create(view->shape, view->nDim,
133     view->data_type, view->content_
134     type);
135 if (!dest_tensor) {
136     for (size_t i = 0; i < view->num_
137     _parents; ++i) tensor_free(
138         parent_tensors[i]);
139     free(parent_tensors);
140     return NULL;
141 }
142
143 bool success = false;
144 switch (view->op_type) {
145     case VIEW_OP_ADD:
146         success = tensor_op_add(dest_
147             _tensor, parent_tensors_
148             [0], parent_tensors[1]);
149         break;
150     case VIEW_OP_SUBTRACT:
151         success = tensor_op_subtract(
152             dest_tensor, parent_
153             tensors[0], parent_
154             tensors[1]);
155         break;
156     case VIEW_OP_MULTIPLY:
157         success = tensor_op_multiply(
158             dest_tensor, parent_
159             tensors[0], parent_
160             tensors[1]);
161         break;
162     case VIEW_OP_SCALE:
163         if (view->op_params) {
164             float scalar = *(float*)
165                 view->op_params;
166             success = tensor_op_
167                 scale(dest_tensor,
168                     parent_tensors[0],
169                     scalar);
170         }
171         break;
172
173     case VIEW_OP_SLICE: {
174         // A materialized slice
175         // requires creating a non-
176         // owning header and then
177         // performing a deep copy
178         // from it.
179         const AgenkTensor* parent =
180             parent_tensors[0];
181         const size_t* offsets = (
182             const size_t*)view->op_
183             params;
184         AgenkTensor* slice_view_
185             header = tensor_slice(
186                 parent, offsets, view->
187                 shape);
188         if (slice_view_header) {
189             success = tensor_op_copy_
190                 _from_view(dest_
191                     _tensor, slice_view_
192                     .header);
193
194         tensor_slice_free(slice_
195             .view_header); ///
196         // Free the temporary
197         // header
198     }
199     break;
200 }

```

Listing 28: Implementation of the materialization engine.

#### 5.4.7 Validation and Testing (**test\_tensors.c**)

A foundational system of this complexity requires a rigorous and aggressive validation strategy. Our

test suite, shown in Listing 29, attacks the substrate from every angle. It includes unit tests for the life-cycle and error handling, as well as complex integration tests that build and materialize nested computational graphs, verifying the mathematical correctness of the final result byte-for-byte. A passing result from this suite provides high confidence in the stability of the entire cognitive architecture’s foundational data layer.

```
1 /**
2 * @file test_tensors.c
3 * @brief Comprehensive, aggressive test
4 * suite for the AGENK Tensor
5 * Substrate.
6 *
7 * This suite is designed to validate
8 * the correctness, robustness, and
9 * performance
10 * of the foundational tensor library.
11 * It attacks the implementation from
12 * multiple
13 * angles, including white-box testing
14 * of internal struct layouts and end-
15 * to-end
16 * validation of the lazy computational
17 * graph and materialization engine.
18 *
19 * @version 1.0.0
20 * @author Ankush Yadav, Ankit Yadav,
21 * AuctaSapience
22 */
23
24 #include "../private/tensor_private.h"
25 #include <stdio.h>
26 #include <stdlib.h>
27 #include <string.h>
28 #include <math.h>
29
30 //=====
31 // SECTION 1: TESTING
32 // FRAMEWORK & HELPERS
33 //=====
34
35 static int g_tests_run = 0;
36 static int g_tests_passed = 0;
37
38 #define COLOR_GREEN "\x1B[32m"
39 #define COLOR_RED "\x1B[31m"
40 #define COLOR_RESET "\x1B[0m"
41
42 #define TEST_SUITE_START(name) printf(" \
43     ---Running Test Suite: %s---\n", \
44     name)
45 #define TEST_CASE(name) printf(" %s\n", \
46     name)
47
48 #define ASSERT(condition)
49
50     \
51     do {
52
53         \
54         g_tests_run++;
55
56             \
57             if (condition) {
```

```

39         \
g_tests_passed++;

40     } else {
41         \
fprintf(stderr, COLOR_RED "[
FAIL] %s:%d: Assertion
failed: %s\n" \
COLOR_RESET, __FILE_
__, __LINE__, #
condition);
42     }
43 }

44 } while (0)

45 #define ASSERT_FLOAT_EQ(a, b) ASSERT(
46     fabs((a) - (b)) < 1e-6)
47
48 // Helper to create a dummy data file
49 // with a predictable pattern for a
50 // given type.
51 void create_dummy_blob_file(const char*
52     filename, size_t size_bytes, uint8_t
53     start_val, AgenkDataType type) {
54     FILE* f = fopen(filename, "wb");
55     if (!f) { perror("Failed to create
56         dummy blob file"); return; }

57     if (type == AGENK_DATA_TYPE_FLOAT32)
58     {
59         size_t num_elements = size_bytes
60             / sizeof(float);
61         for (size_t i = 0; i < num_
62             elements; ++i) {
63             float val = (float)(start_
64                 + i);
65             fwrite(&val, sizeof(float),
66                 1, f);
67         }
68     } else { // Default to byte-wise for
69         // UINT8 and others
70         for (size_t i = 0; i < size_
71             bytes; ++i) {
72             fputc((start_val + i) % 256,
73                 f);
74         }
75     }
76     fclose(f);
77 }

78 //=====
79 // SECTION 2: TEST SUITES
80 //=====

81 void test_suite_lifecycle_and_metadata()
82 {
83     TEST_SUITE_START("Lifecycle&
84         Metadata");

85     TEST_CASE("Basic 2D tensor creation
86         (like a grayscale image)");
87     size_t shape2d[] = {10, 20};
88     // THE FIX: Add the fourth argument
89         for content_type
90     AgenkTensor* t2d = tensor_create(

```

```

shape2d, 2, AGENK_DATA_TYPE_
UINT8, AGENK_CONTENT_TYPE_IMAGE_
PIXELS);
ASSERT(t2d != NULL);
if (t2d) {
    ASSERT(t2d->content_type ==
        AGENK_CONTENT_TYPE_IMAGE_
        PIXELS);
    ASSERT(tensor_get_ndim(t2d) ==
        2);
    const size_t* shape_ptr = tensor_
        _get_shape(t2d);
    ASSERT(shape_ptr[0] == 10 &&
        shape_ptr[1] == 20);
    ASSERT(t2d->data_type == AGENK_
        DATA_TYPE_UINT8);
    ASSERT(t2d->data_size_bytes ==
        10 * 20 * sizeof(uint8_t));
    ASSERT(t2d->strides[0] == 20 *
        sizeof(uint8_t));
    ASSERT(t2d->strides[1] == 1 *
        sizeof(uint8_t));
    tensor_free(t2d);
}
TEST_CASE("Complex_4D_tensor_
creation_(like_a_batch_of_videos
)");
size_t shape4d[] = {8, 10, 192,
    108};
// THE FIX: Add the fourth argument
AgenkTensor* t4d = tensor_create(
    shape4d, 4, AGENK_DATA_TYPE_
    FLOAT32, AGENK_CONTENT_TYPE_RAW_
    BYTES);
ASSERT(t4d != NULL);
if (t4d) {
    ASSERT(tensor_get_ndim(t4d) ==
        4);
    ASSERT(t4d->data_type == AGENK_
        DATA_TYPE_FLOAT32);
    ASSERT(t4d->strides[0] == 10 *
        192 * 108 * sizeof(float));
    ASSERT(t4d->strides[1] == 192 *
        108 * sizeof(float));
    ASSERT(t4d->strides[2] == 108 *
        sizeof(float));
    ASSERT(t4d->strides[3] == 1 *
        sizeof(float));
    tensor_free(t4d);
}
TEST_CASE("Tensor_with_a_zero-sized_
dimension");
size_t shape_zero[] = {10, 0, 20};
// THE FIX: Add the fourth argument
AgenkTensor* t_zero = tensor_create(
    shape_zero, 3, AGENK_DATA_TYPE_
    INT16, AGENK_CONTENT_TYPE_AUDIO_
    PCM);
ASSERT(t_zero != NULL);
if (t_zero) {
    ASSERT(t_zero->data_size_bytes ==
        0);
    tensor_free(t_zero);
}
void test_suite_error_handling() {
    TEST_SUITE_START("ErrorHandling");
    size_t shape[] = {10, 20};
    TEST_CASE("Creation_with_NULL_shape"
    );
    // THE FIX: Add the fourth argument
    ASSERT(tensor_create(NULL, 2, AGENK_
        DATA_TYPE_UINT8, AGENK_CONTENT_
        TYPE_RAW_BYTES) == NULL);
    TEST_CASE("Creation_with_zero_
dimensions");
    // THE FIX: Add the fourth argument
    ASSERT(tensor_create(shape, 0, AGENK_
        DATA_TYPE_UINT8, AGENK_CONTENT_
        TYPE_RAW_BYTES) == NULL);
    TEST_CASE("Creation_with_invalid_
data_type");
    // THE FIX: Add the fourth argument
    ASSERT(tensor_create(shape, 2, AGENK_
        DATA_TYPE_UNDEFINED, AGENK_
        CONTENT_TYPE_RAW_BYTES) == NULL);
    ASSERT(tensor_create(shape, 2, ((
        AgenkDataType)99, AGENK_CONTENT_
        TYPE_RAW_BYTES) == NULL);
    TEST_CASE("Creation_that_would_
overflow_size_t");
    size_t huge_shape[] = {SIZE_MAX / 2,
        4};
    // THE FIX: Add the fourth argument
    ASSERT(tensor_create(huge_shape, 2,
        AGENK_DATA_TYPE_UINT8, AGENK_
        CONTENT_TYPE_RAW_BYTES) == NULL);
    TEST_CASE("Freeing_a_NULL_pointer");
    tensor_free(NULL);
    ASSERT(1);
}
void test_suite_element_access() {
    TEST_SUITE_START("ElementAccess");
    TEST_CASE("Accessing_elements_in_a_3
    D_tensor");
    size_t shape3d[] = {2, 3, 4};
    // THE FIX: Add the fourth argument
    AgenkTensor* t3d = tensor_create(
        shape3d, 3, AGENK_DATA_TYPE_
        INT16, AGENK_CONTENT_TYPE_RAW_
        BYTES);
    ASSERT(t3d != NULL);
    if (!t3d) return;
    int16_t* data_ptr = (int16_t*)t3d->
        data;
    for (size_t z = 0; z < 2; ++z) {
        for (size_t y = 0; y < 3; ++y) {
            for (size_t x = 0; x < 4; ++
                x) {
                data_ptr[z * (3*4) + y *
                    4 + x] = (int16_t)(z * 100 + y *
                    10 + x);
            }
        }
    }
    for (size_t z = 0; z < 2; ++z) {
        for (size_t y = 0; y < 3; ++y) {
            for (size_t x = 0; x < 4; ++
                x) {
                data_ptr[z * (3*4) + y *
                    4 + x] = (int16_t)(z * 100 + y *
                    10 + x);
            }
        }
    }
}
```

```

163     for (size_t x = 0; x < 4; ++x) {
164         size_t coords[] = {z, y, x};
165         int16_t* element_ptr = (int16_t*)tensor_get_element_ptr(t3d, coords);
166         ASSERT(element_ptr != NULL);
167         if (element_ptr) {
168             ASSERT(*element_ptr == (z * 100 + y * 10 + x));
169         }
170     }
171 }
172
173 TEST_CASE("Out-of-bounds access should return NULL (in Debug builds)");
174 #ifndef NDEBUG
175     size_t oob_coords[] = {2, 0, 0};
176     ASSERT(tensor_get_element_ptr(t3d, oob_coords) == NULL);
177 #endif
178     tensor_free(t3d);
179 }
180
181 void test_suite_arithmetic_ops() {
182     TEST_SUITE_START("Arithmetic Operations");
183     size_t shape[] = {10, 10};
184     // THE FIX: Add the fourth argument
185     AgenkTensor* t_a = tensor_create(
186         shape, 2, AGENK_DATA_TYPE_FLOAT32, AGENK_CONTENT_TYPE_RAW_BYTES);
187     AgenkTensor* t_b = tensor_create(
188         shape, 2, AGENK_DATA_TYPE_FLOAT32, AGENK_CONTENT_TYPE_RAW_BYTES);
189     AgenkTensor* t_dest = tensor_create(
190         shape, 2, AGENK_DATA_TYPE_FLOAT32, AGENK_CONTENT_TYPE_RAW_BYTES);
191     ASSERT(t_a && t_b && t_dest);
192     if (!t_a || !t_b || !t_dest) {
193         tensor_free(t_a); tensor_free(t_b);
194         tensor_free(t_dest);
195         return;
196     }
197     for (size_t i = 0; i < 100; ++i) {
198         ((float*)t_a->data)[i] = (float)i;
199         ((float*)t_b->data)[i] = (float)(i * 2);
200     }
201     TEST_CASE("Element-wise ADD operation");
202     ASSERT(tensor_op_add(t_dest, t_a, t_b));
203     for (size_t i = 0; i < 100; ++i) {
204         ASSERT_FLOAT_EQ(((float*)t_dest->data)[i], (float)i + (float)(i * 2));
205     }
206     TEST_CASE("Element-wise SUBTRACT operation");
207     ASSERT(tensor_op_subtract(t_dest, t_b, t_a));
208     for (size_t i = 0; i < 100; ++i) {
209         ASSERT_FLOAT_EQ(((float*)t_dest->data)[i], (float)(i * 2) - (float)i);
210     }
211     tensor_free(t_a); tensor_free(t_b);
212     tensor_free(t_dest);
213
214 /**
215  * @brief Restored suite for testing the most basic view creation and materialization.
216 */
217 void test_suite_views_and_materialization() {
218     TEST_SUITE_START("Views & Materialization (Unit Tests)");
219
220     TEST_CASE("Creating a source view (stubbbed)");
221     AgenkTensorView* v_source = view_create_source(123, "raw_sensor_data");
222     ASSERT(v_source != NULL);
223     if (v_source) {
224         ASSERT(v_source->op_type == VIEW_OP_SOURCE);
225         ASSERT(v_source->nDim == 3);
226         ASSERT(v_source->shape[0] == 100 && v_source->shape[1] == 200 && v_source->shape[2] == 3);
227         view_free(v_source);
228     }
229
230     TEST_CASE("Materializing a source view");
231     size_t blob_size = 100 * 200 * 3 * sizeof(uint8_t);
232     create_dummy_blob_file("test_blob_123.bin", blob_size, 0, AGENK_DATA_TYPE_UINT8);
233     AgenkTensorView* v_source_to_materialize = view_create_source(123, "raw_sensor_data");
234     ASSERT(v_source_to_materialize != NULL);
235
236     AgenkTensor* t_materialized = view_materialize(v_source_to_materialize);
237     ASSERT(t_materialized != NULL);
238
239     if (t_materialized) {
240         ASSERT(tensor_get_ndim(t_materialized) == 3);
241         ASSERT(t_materialized->data_size_bytes == blob_size);
242         ASSERT(((uint8_t*)t_materialized->data)[0] == 0);
243         ASSERT(((uint8_t*)t_materialized->data)[256] == 0);
244         tensor_free(t_materialized);
245     }

```

```

246     view_free(v_source_to_materialize);
247     remove("test_blob_123.bin");
248 }
249 /**
250 * @brief Test suite for the full
251 * computational graph pipeline (integration tests).
252 */
253 void test_suite_computational_graphs() {
254     TEST_SUITE_START("Computational\u2022Graph\u2022Integration\u2022Tests");
255
256     size_t blob_size_floats = 10 * 20 *
257         sizeof(float);
258     create_dummy_blob_file("test_blob_a.
bin", blob_size_floats, 0, AGENK
    _DATA_TYPE_FLOAT32);
259     create_dummy_blob_file("test_blob_b.
bin", blob_size_floats, 100,
AGENK_DATA_TYPE_FLOAT32);
260
261     AgenkTensorView* v_a = view_create_
        source(1, "source_a");
262     AgenkTensorView* v_b = view_create_
        source(2, "source_b");
263     ASSERT(v_a && v_b);
264     if (!v_a || !v_b) return;
265
266     TEST_CASE("Materializing\u2022a\u2022simple\u2022
ADD\u2022graph");
267     AgenkTensorView* v_add = view_create_
        _add(v_a, v_b);
268     ASSERT(v_add != NULL);
269     if (v_add) {
270         AgenkTensor* t_add = view_
            materialize(v_add);
271         ASSERT(t_add != NULL);
272         if (t_add) {
273             size_t coords[] = {5, 5};
274             float* val_ptr = (float*)
                tensor_get_element_ptr(t_
                    _add, coords);
275             float expected = (5.0f *
                20.0f + 5.0f) + (100.0f
                + 5.0f * 20.0f + 5.0f);
276             ASSERT_FLOAT_EQ(*val_ptr,
                expected);
277             tensor_free(t_add);
278         }
279         view_free(v_add);
280     }
281
282     TEST_CASE("Materializing\u2022a\u2022complex\u2022
nested\u2022graph:\u2022(slice(A)\u2022+\u2022slice(
    B))\u2022*\u20222.0");
283     size_t offsets[] = {2, 3};
284     size_t slice_shape[] = {5, 5};
285     AgenkTensorView* v_a_slice = view_
        create_slice(v_a, offsets, slice_
            _shape);
286     AgenkTensorView* v_b_slice = view_
        create_slice(v_b, offsets, slice_
            _shape);
287     AgenkTensorView* v_add_slices = view_
        create_add(v_a_slice, v_b_slice
    );
288     float scalar = 2.0f;
289     AgenkTensorView* v_final = view_
        create_scale(v_add_slices,
            scalar);
290     ASSERT(v_a_slice && v_b_slice && v_
        add_slices && v_final);
291
292     if (v_final) {
293         AgenkTensor* t_final = view_
            materialize(v_final);
294         ASSERT(t_final != NULL);
295         if(t_final){
296             ASSERT(tensor_get_ndim(t_
                final) == 2);
297             ASSERT(tensor_get_shape(t_
                final)[0] == 5 && tensor
                _get_shape(t_final)[1]
                == 5);
298
299             size_t final_coords[] =
                {1,1}; // Corresponds to
                    original (3,4)
300             float* val_ptr = (float*)
                tensor_get_element_ptr(t_
                    _final, final_coords);
301             float val_a_orig = (3.0f *
                20.0f + 4.0f);
302             float val_b_orig = 100.0f +
                val_a_orig;
303             ASSERT_FLOAT_EQ(*val_ptr, (
                val_a_orig + val_b_orig)
                * 2.0f);
304
305             tensor_free(t_final);
306         }
307         view_free(v_final);
308     }
309     view_free(v_a); view_free(v_b);
310     view_free(v_a_slice); view_free(v_b_
        _slice);
311     view_free(v_add_slices);
312 }
313 /**
314 * @brief Tests the correct handling of
315 * logical content types.
316 */
317 void test_suite_content_types() {
318     TEST_SUITE_START("Logical\u2022Content\u2022
Types");
319
320     TEST_CASE("Creating\u2022a\u2022tensor\u2022to\u2022
represent\u2022a\u2022UTF-8\u2022string");
321     const char* text = "hello";
322     size_t text_len = strlen(text);
323     size_t shape[] = {text_len}; // A 1D
            tensor
324
325     // A text string is physically a
            sequence of UINT8, but logically
            it's UTF8_TEXT
326     AgenkTensor* text_tensor = tensor_
        create(shape, 1, AGENK_DATA_TYPE_
            _UINT8, AGENK_CONTENT_TYPE_UTF8_
            _TEXT);
327     ASSERT(text_tensor != NULL);
328
329     if (text_tensor) {
330         ASSERT(text_tensor->data_type ==
            AGENK_DATA_TYPE_UINT8);
331         ASSERT(text_tensor->content_type
            == AGENK_CONTENT_TYPE_UTF8_
            _TEXT);

```

```

    TEXT);

    // Copy the string data in and
    // verify
    memcpy(text_tensor->data, text,
           text_len);
    ASSERT(memcmp(text_tensor->data,
                  text, text_len) == 0);

    tensor_free(text_tensor);
}

//=====
// SECTION 3: TEST RUNNER
//=====

int main() {
    printf("=====\\n");
    printf("Running_AGENK_Tensor_"
          "Substrate_Test_Suite\\n");
    printf("=====\\n");

    test_suite.lifecycle_and_metadata();
    printf("\\n");
    test_suite.error_handling();
    printf("\\n");
    test_suite.element_access();
    printf("\\n");
    test_suite.content_types();
    printf("\\n");
    test_suite.arithmetic_ops();
    printf("\\n");
    test_suite.views_and_materialization
        ();
    printf("\\n");

    printf("=====\\n");
    printf("==TEST_SUMMARY==\\n");
    printf("=====\\n");
    if (g_tests_passed == g_tests_run) {
        printf(COLOR_GREEN "SUCCESS: All "
               "%d tests passed.\\n" COLOR_
               RESET, g_tests_run);
        return 0;
    } else {
        fprintf(stderr, COLOR_RED "
            FAILURE: %d out of %d tests "
            "failed.\\n" COLOR_RESET,
            g_tests_run - g_tests_
                passed, g_tests_run);
        ;
    }
}

```

Listing 29: The comprehensive test suite for the Tensor Substrate.

## 6 The Snippets Substrate: A Universal Genome for Skills

The Execution Substrate, which we will detail in the next section, provides the agent with a "CPU"—a universal engine for running code. However, a CPU is useless without programs to

run. This raises a question of profound architectural significance: *What is the fundamental representation of a skill?* A primitive approach would be to hard-code skills as C functions within the agent's core. This is a fatal design flaw, as it creates a rigid, monolithic agent incapable of learning or acquiring new capabilities without being entirely recompiled and redeployed. This is antithetical to our goal of a continuously evolving intelligence.

Our central thesis is that for an agent to be truly generalist and adaptive, its skills must not be an integral part of its core machinery. Instead, skills must be treated as **data**. They must be self-contained, platform-agnostic, and describable in a universal format that can be stored, transmitted, reasoned about, and even generated. This is the sole and exclusive purpose of the **Snippets Substrate**. It does not execute anything; it provides the formal, foundational vocabulary and grammar for defining what a skill *is*. It is the agent's genome—a library of "executable genes" that encode every capability the agent possesses.

### 6.1 Architectural Philosophy: The Skill as Data

The core insight of this substrate is the complete decoupling of a skill's **definition** from its **persistence** and **execution**. By representing every skill as a standardized, serializable data structure—the **AgenkSnippet**—we transform the abstract concept of a "capability" into a concrete entity that can be:

- **Stored:** Skills are persisted as first-class objects in the **ObjectStore**, forming a robust, transactional, and queryable long-term memory of the agent's abilities. This is the agent's "Library of Alexandria" for skills.
- **Transmitted:** A serialized Snippet can be sent over a network, allowing agents to teach and learn from each other by sharing the raw data of their skills.
- **Composed:** A higher-level cognitive function can dynamically assemble pipelines of simple Snippets to solve complex problems.
- **Generated:** Most powerfully, this architecture enables a future where the agent's own reasoning models can *generate new AgenkSnippet data structures as their output*, effectively inventing and writing their own new tools.

This substrate is therefore the ontological foundation of the agent's entire skill set, providing the universal "nouns" that all other systems use to describe and reason about action.

## 6.2 Architectural Design and Implementation

The Snippets Substrate is implemented as a foundational module, `src/snippets/`. It is designed to have minimal dependencies, ensuring its universal applicability across the entire system.

### 6.2.1 Project Structure and Build System Integration

The module's structure reflects its dual responsibilities: defining the data structures for skills and managing their persistence. This tight coupling of data definition and data persistence logic is a design choice known as **high cohesion**, which leads to more maintainable and understandable software modules.

```
src/
└── snippets/
    ├── include/
    │   └── agenk_snippet.h
    ├── snippet.c
    ├── snippet_registry.c
    └── test/
        └── test_snippets.c
```

Its definition in the master 'CMakeLists.txt' file establishes its place in the architectural hierarchy. It depends on the **ObjectStore** for its persistence backend and on the **TensorSubstrate** for a single type definition. It has no knowledge of the **Execution** module, enforcing a clean, one-way dependency graph.

```
1 # =====
2 # CMakeLists.txt for the AGENK
3 #
4 # Version: 1.0.0
5 # =====
6 #
7 # =====
8 # SECTION 1: PROJECT PREAMBLE
9 # & BUILD STANDARDS
10 #
11 cmake_minimum_required(VERSION 3.16)
12 project(AGENK VERSION 1.0.0 LANGUAGES C)
13
14 set(CMAKE_C_STANDARD 99)
15 set(CMAKE_C_STANDARD_REQUIRED ON)
16 set(CMAKE_C_EXTENSIONS OFF)
17 include(ExternalProject)
18 set(CMAKE_ARCHIVE_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/lib)
19 set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/bin)
20
21 if(NOT CMAKE_BUILD_TYPE)
22     set(CMAKE_BUILD_TYPE Debug)
23 endif()
24 message(STATUS "Build type set to: ${CMAKE_BUILD_TYPE}")
25
26 option(AGENK_ENABLE_PYTHON "Enable Python runtime and native bridge" ON)
27
28 option(AGENK_ENABLE_NODEJS "Enable NodeJS/NPM runtime" OFF) #
Placeholder for the future
29
# =====
# SECTION 2: VENDORED &
# SYSTEM DEPENDENCY MANAGEMENT
# =====
# --- System Libraries ---
30 find_package(Threads REQUIRED)
31 find_library(MATH_LIBRARY m)
32
# --- Dependency Discovery ---
33 find_package(Threads REQUIRED)
34 if(AGENK_ENABLE_PYTHON)
35     message(STATUS "Python support is ENABLED.")
36     find_package(Python3 COMPONENTS Development NumPy REQUIRED)
37 else()
38     message(STATUS "Python support is DISABLED.")
39 endif()
40
# --- mbedtls (vendored from deps) ---
41 set(ENABLE_TESTING OFF CACHE BOOL "Disable mbedtls tests")
42 set(ENABLE_PROGRAMS OFF CACHE BOOL "Disable mbedtls programs")
43 add_subdirectory(deps/mbedtls)
44 set(MBEDTLS_LIBRARIES mbedtls mbedtls_x509 mbedtls_crypto)
45 set(MBEDTLS_LIBRARY_DIR ${CMAKE_ARCHIVE_OUTPUT_DIRECTORY})
46 set(MBEDTLS_INCLUDE_DIR ${CMAKE_CURRENT_SOURCE_DIR}/deps/mbedtls/include)
47
# --- libcurl (vendored from deps/, built with custom command) ---
48 ExternalProject_Add(
49     curl_local_build
50     SOURCE_DIR ${CMAKE_CURRENT_SOURCE_DIR}/deps/curl
51     BINARY_DIR ${CMAKE_BINARY_DIR}/curl-build
52     # Use bash -c to create a reliable shell environment for the configure script
53     CONFIGURE_COMMAND bash -c
54         "LDFLAGS='-L${MBEDTLS_LIBRARY_DIR}' LIBS=' -l mbedtls -l mbedtls_x509 -l mbedtls_crypto -lpthread' CPPFLAGS='-I${MBEDTLS_INCLUDE_DIR}' < SOURCE_DIR>/configure --disable-shared --with-ssl --without-zlib"
55     BUILD_COMMAND make
56     INSTALL_COMMAND ""
57     DEPENDS ${MBEDTLS_LIBRARIES}
```

```

66 )
67 add_library(libcurl STATIC IMPORTED
68   GLOBAL)
69 set_target_properties(libcurl PROPERTIES
70   IMPORTED_LOCATION "${CMAKE_BINARY_
71     DIR}/curl-build/lib/.libs/
72     libcurl.a")
73 )
74 add_dependencies(libcurl curl_local_
75   build)
76
77 # --- Other Vendored Libraries ---
78 set(FLATCC_TEST OFF CACHE BOOL "")
79 set(FLATCC_SAMPLES OFF CACHE BOOL "")
80 add_subdirectory(deps/flatcc)
81 add_library(lmdb STATIC deps/lmdb/mdb.c
82   deps/lmdb/midl.c)
83 target_include_directories(lmdb PUBLIC ${CMAKE_CURRENT_SOURCE_DIR}/deps/lmdb)
84 add_library(md5 STATIC deps/md5/md5.c)
85 target_include_directories(md5 PUBLIC ${CMAKE_CURRENT_SOURCE_DIR}/deps/md5)
86 set(CJSON_LIB "")
87 if(EXISTS ${CMAKE_CURRENT_SOURCE_DIR}/deps/cJSON/cJSON.c)
88   add_library(cjson STATIC deps/cJSON/cJSON.c)
89   target_include_directories(cjson PUBLIC ${CMAKE_CURRENT_SOURCE_DIR}/deps/cJSON)
90   set(CJSON_LIB cjson)
91   set(CJSON_FOUND TRUE)
92 endif()
93 ExternalProject_Add(tcc_external_project
94   SOURCE_DIR ${CMAKE_CURRENT_SOURCE_DIR}/deps/tcc
95   CONFIGURE_COMMAND ""
96   BUILD_COMMAND "" INSTALL_COMMAND "")
97 add_custom_target(build_tcc_library ALL
98   COMMAND ${CMAKE_COMMAND} -E chdir ${CMAKE_CURRENT_SOURCE_DIR}/deps/tcc
99   ./configure --libdir=${CMAKE_CURRENT_SOURCE_DIR}/deps/tcc/lib --extra-
100  cflags="-fPIC" COMMAND ${CMAKE_COMMAND} -P ${CMAKE_CURRENT_BINARY_DIR}/cmake_script_patch_tcc_makefile
101  .cmake COMMAND ${CMAKE_COMMAND} -E chdir ${CMAKE_CURRENT_SOURCE_DIR}/deps/tcc make libtcc.a COMMAND ${CMAKE_COMMAND} -E chdir ${CMAKE_CURRENT_SOURCE_DIR}/deps/tcc
102  make COMMAND ${CMAKE_COMMAND} -E copy_if_different ${CMAKE_CURRENT_SOURCE_DIR}/deps/tcc/libtcc.a ${CMAKE_BINARY_DIR}/lib/ COMMAND ${CMAKE_COMMAND} -E copy_if_different ${CMAKE_CURRENT_SOURCE_DIR}/deps/tcc/libtcc1.a ${CMAKE_BINARY_DIR}/lib/
103  DEPENDS tcc_external_project)
104 file(WRITE ${CMAKE_CURRENT_BINARY_DIR}/cmake_script_patch_tcc_makefile.
105  cmake "set(TCC_LIB_MAKEFILE_PATH\"${CMAKE_CURRENT_SOURCE_DIR}/deps/tcc/lib/Makefile\")\nfile(READ \\"${TCC_LIB_MAKEFILE_PATH}\\" TCC_LIB_MAKEFILE_CONTENT)\nstring(REPLACE \"bcheck.o\" \"#bcheck.o\" \\"${TCC_LIB_MAKEFILE_PATH}\\" \"$TCC_LIB_MAKEFILE_CONTENT\")\nfile(WRITE \\"${TCC_LIB_MAKEFILE_PATH}\\" \"$TCC_LIB_MAKEFILE_CONTENT\")"
106  PATCHED\" \\"${TCC_LIB_MAKEFILE_PATH}\\" \"$TCC_LIB_MAKEFILE_CONTENT\")\nfile(WRITE \\"${TCC_LIB_MAKEFILE_PATH}\\" \"$TCC_LIB_MAKEFILE_CONTENT\")"
107  target_include_directories(object_store PUBLIC ${CMAKE_CURRENT_SOURCE_DIR}/src/
108  _object_store)
109  add_custom_command(
110    OUTPUT ${GENERATED_FILES}
111    COMMAND $<TARGET_FILE:flatcc_cli> -a
112      -o ${GENERATED_INCLUDE_DIR} ${FBS_SCHEMA}
113    DEPENDS ${FBS_SCHEMA} flatcc_cli
114    COMMENT "Generating C headers from FlatBuffers schema"
115  )
116  add_custom_target(generate_object_headers DEPENDS ${GENERATED_FILES})
117  #
118  # SECTION 4: AGENK CORE
119  # LIBRARY DEFINITIONS
120  #
121  # --- LIBRARY 1: object store (The DataBase) ---
122  # --- NEW LIBRARY: object_store (The Persistence Layer) ---
123  add_library(object_store STATIC
124    src/object_store/core/fb_serializer.c
125    src/object_store/core/object_store.c
126  )
127  #
128  # This library depends on the auto-generated FlatBuffers headers
129  add_dependencies(object_store generate_object_headers)
130  #
131  # Tell the compiler where to find its headers
132  target_include_directories(object_store PUBLIC
133    ${CMAKE_CURRENT_SOURCE_DIR}/src/

```

```

134     object_store
135     ${GENERATED_INCLUDE_DIR}
136     ${CMAKE_CURRENT_SOURCE_DIR}/deps/
137     flatcc/include
138 )
139 # Tell the linker what other libraries
140 # it needs
141 target_link_libraries(object_store
142     PRIVATE
143     flatccrt      # The FlatBuffers
144     runtime library
145     lmdb
146     md5
147     Threads::Threads
148     ${MATH_LIBRARY} # ADD THIS LINE
149 )
150 # Handle the optional dependency on
151 # cJSON
152 if(CJSON_FOUND)
153     target_link_libraries(object_store
154         PRIVATE ${CJSON_LIB})
155     target_compile_definitions(object_
156         store PRIVATE CJSON_ENABLED)
157 endif()
158 # --- LIBRARY 2: agenk_tensor (The Data
159 # Canvas) ---
160 add_library(agenk_tensor STATIC
161     src/tensors/tensor_lifecycle.c
162     src/tensors/tensor_ops.c
163     src/tensors/tensor_views.c
164     src/tensors/tensor_materialize.c
165 )
166 target_include_directories(agenk_tensor
167     PUBLIC
168     ${CMAKE_CURRENT_SOURCE_DIR}/src/
169     tensors/include
170 )
171 # --- LIBRARY 3: agenk_snippet (The Data
172 # Structures & Registry) ---
173 add_library(agenk_snippet STATIC
174     src/snippets/snippet.c
175     src/snippets/snippet_registry.c #  

176     <-- ADD NEW SOURCE FILE
177 )
178 target_include_directories(agenk_snippet
179     PUBLIC
180     ${CMAKE_CURRENT_SOURCE_DIR}/src/
181     snippets/include
182 )
183 # Snippets now need the ObjectStore for
184 # persistence and Tensors for data
185 # types.
186 target_link_libraries(agenk_snippet
187     PRIVATE
188     agenk_tensor
189     object_store
190     md5 # Dependency of the registry for
191     # key generation
192 )
193 # --- LIBRARY 4: class_framework (The
194 # Live Object Runtime) ---
195 add_library(class_framework STATIC
196     src/objects/core/object_impl.c
197     src/objects/core/object_registry.c
198     src/objects/core/object_db.c
199 )
200 target_include_directories(class_
201     framework PUBLIC ${CMAKE_CURRENT_
202     SOURCE_DIR}/src/objects)
203 target_link_libraries(class_framework
204     PRIVATE lmdb Threads::Threads)
205 # --- LIBRARY 5: agenk_core (The "Kernel
206 # / Stable Native API) ---
207 set(CORE_SOURCES src/core/core.c)
208 if(AGENK_ENABLE_PYTHON)
209     list(APPEND CORE_SOURCES src/core/
210         bridge/native_bridge_python.c)
211 endif()
212 add_library(agenk_core STATIC ${CORE_
213     SOURCES})
214 target_include_directories(agenk_core
215     PUBLIC
216     ${CMAKE_CURRENT_SOURCE_DIR}/src/core
217     /include
218 )
219 if(AGENK_ENABLE_PYTHON)
220     target_include_directories(agenk_
221         core PUBLIC
222         ${Python3_INCLUDE_DIRS}
223         ${Python3_NumPy_INCLUDE_DIRS}
224     )
225 endif()
226 target_link_libraries(agenk_core PUBLIC
227     agenk_snippet agenk_tensor class_
228     framework
229 )
230 if(AGENK_ENABLE_PYTHON)
231     target_link_libraries(agenk_core
232         PRIVATE Python3::Python Python3
233         ::NumPy)
234 endif()
235 add_library(agenk_sensation STATIC
236     src/senses/core/sense_manager.c
237     src/senses/core/sense_registry.c
238     # ADD THESE MISSING SOURCE FILES
239     src/senses/builder/build_controller.
240     c
241     src/senses/builder/tool_check.c
242 )
243 target_include_directories(agenk_
244     sensation PUBLIC
245     ${CMAKE_CURRENT_SOURCE_DIR}/src/
246     senses/include
247 )
248 target_compile_definitions(agenk_
249     sensation PRIVATE AGENK_SOURCE_DIR="
250     ${CMAKE_SOURCE_DIR}")
251 target_link_libraries(agenk_sensation
252     PRIVATE
253     agenk_core
254     md5
255     libtcc
256     object_store # This must be here
257 )
258 #
259 # --- LIBRARY 6: agenk_execution (The "
260 # CPU" / Execution Engine) ---
261 set(EXECUTION_SOURCES
262     src/execution/core/engine.c
263     src/execution/builder/dependency_
264     manager.c

```

```

234     src/execution/runtimes/runtime_c.c      276
235     src/execution/runtimes/runtime_shell    .c
236   )
237   if(AGENK_ENABLE_PYTHON)
238     list(APPEND EXECUTION_SOURCES src/
239           execution/runtimes/runtime_
240           python.c)
241   endif()
242   add_library(agenk_execution STATIC ${EXECUTION_SOURCES})
243
244 target_include_directories(agenk_
245   execution PUBLIC
246   ${CMAKE_CURRENT_SOURCE_DIR}/src/
247   execution/include
248   PRIVATE
249   ${CMAKE_CURRENT_SOURCE_DIR}/src/
250   execution
251 )
252
253 target_compile_definitions(agenk_
254   execution PRIVATE
255   AGENK_SOURCE_DIR="${CMAKE_SOURCE_DIR}"
256   AGENK_BINARY_DIR="${CMAKE_BINARY_DIR}"
257   AGENK_TCC_PATH="${CMAKE_CURRENT_SOURCE_DIR}/deps/tcc"
258   # THE FIX: Provide a reliable path
259   # to the *built* TCC libraries
260   AGENK_TCC_LIB_PATH="${CMAKE_ARCHIVE_OUTPUT_DIRECTORY}"
261   # Provide include directories for
262   # runtime compilation
263   AGENK_CORE_INCLUDE_DIR="${CMAKE_CURRENT_SOURCE_DIR}/src/core/
264   include"
265   AGENK_SNIPPET_INCLUDE_DIR="${CMAKE_CURRENT_SOURCE_DIR}/src/snippets/
266   include"
267 )
268
269 if(AGENK_ENABLE_PYTHON)
270   # Add compile definitions needed for
271   # the Python runtime.
272   # - AGENK_PYTHON_SUPPORT enables all
273   # Python-related code blocks.
274   # - PYTHON_SITE_PACKAGES tells the C
275   # code where to find system
276   # packages.
277   target_compile_definitions(agenk_
278     execution PRIVATE
279     AGENK_PYTHON_SUPPORT
280     "PYTHON_SITE_PACKAGES=\\"${Python3_SITELIB}\\""
281   )
282 endif()
283
284 target_link_libraries(agenk_execution
285   PRIVATE
286     agenk_core agenk_snippet libtcc
287     Threads::Threads md5
288 )
289
290 if(AGENK_ENABLE_PYTHON)
291   target_link_libraries(agenk_
292     execution PRIVATE Python3::
293     Python)
294 endif()
295
296 # =====
297 # SECTION 5: EXECUTABLE
298 # TARGETS (TESTS & EXAMPLES)
299 # =====
300
301 # --- EXECUTABLES ---
302 add_executable(test_tensors src/tensors/
303   test/test_tensors.c)
304 target_link_libraries(test_tensors
305   PRIVATE agenk_tensor)
306
307 add_executable(test_sensation src/senses/
308   test/test_sensation.c)
309 target_link_libraries(test_sensation
310   PRIVATE agenk_sensation)
311
312 add_executable(test_sensation_build src/
313   senses/test/test_sensation_build.c)
314 target_link_libraries(test_sensation_
315   build PRIVATE agenk_sensation)
316
317 add_executable(test_sense_registry src/
318   senses/test/test_sense_registry.c)
319 target_link_libraries(test_sense_
320   registry PRIVATE agenk_sensation ${MATH_LIBRARY})
321
322 add_executable(test_concurrency src/
323   objects/test/test_concurrency.c)
324 target_link_libraries(test_concurrency
325   PRIVATE class_framework Threads::
326   Threads)
327
328 add_executable(test_errors src/objects/
329   test/test_errors.c)
330 target_link_libraries(test_errors
331   PRIVATE class_framework)
332
333 add_executable(test_hierarchy src/
334   objects/test/test_hierarchy.c)
335 target_link_libraries(test_hierarchy
336   PRIVATE class_framework)
337
338 add_executable(test_stress src/objects/
339   test/test_stress.c)
340 target_link_libraries(test_stress
341   PRIVATE class_framework)
342
343 add_executable(example_basic_usage src/
344   objects/examples/basic_usage.c)
345 target_link_libraries(example_basic_
346   usage PRIVATE class_framework)
347
348 add_executable(example.lifecycle src/
349   objects/examples/lifecycle_and_
350   groups.c)
351 target_link_libraries(example.lifecycle
352   PRIVATE class_framework)
353
354 add_executable(test_object_store src/
355   object_store/test/test_object_store.
356   c)
357 target_link_libraries(test_object_store
358   PRIVATE object_store Threads::
359   Threads ${CJSON_LIB})
360
361 # --- NEW EXECUTABLE for Snippet testing
362
363 add_executable(test_snippets src/

```

```

snippets/test/test_snippets.c)           363  if(CMAKE_SYSTEM_NAME STREQUAL "Linux")
target_link_libraries(test_snippets      364    target_link_options(test_execution
PRIVATE                                365      PRIVATE "-Wl,--export-dynamic")
319    agenk_snippet                      366  endif()
320    class_framework # For CLASS_INIT   367
321    object_store   # For the registry's 368  # =====
322 )                                     369  # SECTION 6: CTEST
323                                         370  # INTEGRATION
324 add_executable(test_execution src/     371  # =====
325   execution/test_execution.c)          372  enable_testing()
326 if(AGENK_ENABLE_PYTHON)
327   target_compile_definitions(test_
328     execution PRIVATE AGENK_PYTHON_
329     SUPPORT) # <-- ADD THIS LINE
330   # THE DEFINITIVE FIX for systems
331   # using GNU ld or clang ld.
332   # This consolidates all linking into
333   # a single, ordered command.
334   if(CMAKE_SYSTEM_NAME STREQUAL "Linux"
335     " OR CMAKE_SYSTEM_NAME STREQUAL
336     "Darwin")
337     target_link_libraries(test_
338       execution PRIVATE
339       # --- Libraries used directly by
340       # the test C code ---
341       agenk_execution
342       class_framework
343
344       # --- Force-load the python
345       # bridge, THEN provide its
346       # dependencies ---
347       "-Wl,--whole-archive"
348       agenk_core
349       "-Wl,--no-whole-archive"
350       agenk_snippet
351       agenk_tensor
352
353       # --- Provide all other system/
354       # vendoried dependencies ---
355       lmdb
356       Threads::Threads
357       Python3::Python
358       Python3::NumPy
359     )
360   else()
361     # Fallback for other systems (e.g
362     # .., Windows).
363     target_link_libraries(test_
364       execution PRIVATE
365       agenk_execution class_
366         framework agenk_core agenk
367         _snippet
368       agenk_tensor lmdb Threads::
369         Threads Python3::Python
370     )
371   endif()
372 else()
373   # Link normally if Python is
374   # disabled
375   target_link_libraries(test_execution
376     PRIVATE
377       agenk_execution class_framework
378         agenk_core agenk_snippet
379       agenk_tensor lmdb Threads::
380         Threads
381   )
382 endif()

```

Listing 30: The ‘agenk\_snippet’ library definition, showing its foundational dependencies.

## 6.2.2 The Public Contract (agenk\_snippet.h)

The public API, presented in Listing 31, is the formal grammar for all skills. It is the ”Rosetta Stone” of the agent’s capabilities, defining the set of C structs that constitute this universal language, and it now includes the API for the persistence layer.

- The Data Structures:** The header defines the core ”nouns”: `AgenkSnippet`, `AgenkDependency`, `AgenkCode`, `ExecutionContext`, and `ExecutionResult`. These provide a complete, self-contained, and platform-agnostic way to describe any skill.
- The Builder Pattern API:** A suite of functions like `snippet_create` and `dependency_add_build_command` provide a safe and memory-managed way to construct these complex data structures.

- **The Snippet Registry API:** A new set of functions, such as `snippet_registry_init` and `snippet_registry_add`, provide the official interface for saving and retrieving skills from the agent's long-term memory.

```

1  /**
2  * @file agenk_snippet.h
3  * @brief Public API for all universal
4  *        data structures: Snippets, API
5  *        Contexts, and Execution Results.
6  *
7  * This header defines the fundamental "nouns" of the AGENK agent's
8  * capabilities.
9  * It is part of the foundational 'snippets' library, ensuring that all
10 * other
11 * subsystems have a common, stable set
12 * of data formats to work with.
13 * @version 1.0.0
14 * @author Ankush Yadav, Ankit Yadav,
15 *         AuctaSapience
16 */
17
18 #ifndef AGENK_SNIPPET_H
19 #define AGENK_SNIPPET_H
20
21 #include <stddef.h>
22 #include <stdint.h>
23 #include <stdbool.h>
24
25 // --- Forward Declarations for Opaque
26 //      Structs ---
27 typedef struct AgenkTensor AgenkTensor;
28 typedef struct AgenkSnippet AgenkSnippet
29 ;
30 typedef struct AgenkDependency
31     AgenkDependency;
32 typedef struct AgenkCode AgenkCode;
33 typedef struct CodeVariant CodeVariant;
34 typedef struct ApiRequestContext
35     ApiRequestContext;
36 typedef struct ApiRequestParam
37     ApiRequestParam;
38 typedef struct ExecutionResult
39     ExecutionResult;
40 typedef struct ExecutionContext
41     ExecutionContext;
42
43 // --- Core Data Structures ---
44
45 /** @brief Describes a single external
46 *        software dependency. */
47 struct AgenkDependency {
48     enum { DEP_SOURCE, DEP_PIP, DEP_NPM,
49           DEP_SYSTEM } type;
50     char* uri; // THE FIX: Was char, is
51           now char*
52     char** build_commands;
53     size_t num_build_commands;
54     char** required_artifacts;
55     size_t num_required_artifacts;
56 };

```

```

43 /** @brief Describes a piece of source
44 *        code with OS-specific variants. */
45 struct AgenkCode {
46     enum { LANG_C, LANG_PYTHON, LANG_
47           NODEJS, LANG_SHELL } language;
48     char* entry_point; // THE FIX: Was
49           char, is now char*
50     struct CodeVariant {
51         enum { OS_ANY, OS_LINUX, OS_
52             WINDOWS, OS_MACOS } target_
53             os;
54         char* source_code;
55     }* variants;
56     size_t num_variants;
57 };
58
59 /** @brief The universal "Executable
60 *        Gene" for any skill. */
61 struct AgenkSnippet {
62     char* snippet_id; // THE FIX: Was
63           char, is now char*
64     char* description;
65     AgenkDependency** dependencies;
66     size_t num_dependencies;
67     AgenkCode* execution_code;
68     AgenkCode* test_code;
69     char* expected_test_output;
70 };
71
72 /** @brief A single key-value pair for
73 *        an HTTP header or query parameter. */
74 struct ApiRequestParam {
75     char* key; // THE FIX: Was char, is
76           now char*
77     char* value;
78 };
79
80 /** @brief Describes the full context
81 *        for a dynamic API request. */
82 struct ApiRequestContext {
83     char* url; // THE FIX: Was char, is
84           now char*
85     char* http_method;
86     ApiRequestParam* headers;
87     size_t num_headers;
88     size_t headers_capacity;
89     ApiRequestParam* query_params;
90     size_t num_params;
91     size_t params_capacity;
92     AgenkTensor* body;
93 };
94
95 /** @brief Standardized return type for
96 *        all execution operations. */
97 struct ExecutionResult {
98     int exit_code;
99     char* stdout_buffer; // THE FIX: Was
100           char, is now char*
101     char* stderr_buffer;
102     void* return_value;
103 };
104
105 /** @brief Provides input and context to
106 *        a running Snippet. */
107 struct ExecutionContext {
108     const char* input_uri; // THE FIX:
109           Was char, is now const char*
110     const ApiRequestContext* api_request
111 };

```

```

97 };
```

```

98 // --- Snippet Registry API ---
```

```

100 // Forward declare the object store
101 // handle for the init function
102 typedef struct object_store_t object_
103 store_t;
```

```

104 /**
105 * @brief Initializes the snippet
106 * registry with a handle to the
107 * object store.
108 * @param store A valid, initialized
109 * object_store_t handle.
110 * @return True on success, false on
111 * failure (e.g., failed to create
112 * table).
113 */
114 bool snippet_registry_init(object_store_
115 * store);
```

```

116 /**
117 * @brief Shuts down the snippet
118 * registry. (Currently a no-op).
119 */
120 void snippet_registry_shutdown(void);
```

```

121 /**
122 * @brief Adds a new snippet to the
123 * registry or updates an existing one
124 * .
125 * The object_id of the internal c_
126 * object_t is used as the key.
127 * @param snippet The snippet to save.
128 * @return True on success, false on
129 * failure.
130 */
131 bool snippet_registry_add(const
132 AgenkSnippet* snippet);
```

```

133 /**
134 * @brief Retrieves a snippet from the
135 * registry by its unique ID.
136 * @param snippet_id The ID of the
137 * snippet to retrieve.
138 * @return A pointer to a new, fully
139 * reconstructed AgenkSnippet. The
140 * caller
141 * is responsible for freeing
142 * this with snippet_free(). Returns
143 * NULL
144 * if not found or on error.
145 */
146 AgenkSnippet* snippet_registry_get_by_id(
147 const char* snippet_id);
```

```

148 /**
149 * @brief Deletes a snippet from the
150 * registry.
151 * @param snippet_id The ID of the
152 * snippet to delete.
153 * @return True on success or if the
154 * snippet didn't exist, false on
155 * error.
156 */
157 bool snippet_registry_delete(const char*
158 snippet_id);
```

```

159 // --- Memory Management APIs ---
```

```

160 // Snippet Builder
161 AgenkSnippet* snippet_create(const char*
162 description);
163 void snippet_free(AgenkSnippet* s);
164 AgenkDependency* dependency_create(int
165 type, const char* uri);
166 void dependency_add_build_command(
167 AgenkDependency* d, const char*
168 command);
169 void dependency_add_required_artifact(
170 AgenkDependency* d, const char*
171 artifact_path);
172 AgenkCode* code_create(int language,
173 const char* entry_point);
174 void code_add_variant(AgenkCode* c, int
175 os, const char* source);
176 void snippet_add_dependency(AgenkSnippet
177 * s, AgenkDependency* d);
178 void snippet_set_execution_code(
179 AgenkSnippet* s, AgenkCode* c);
180 void snippet_set_test_code(AgenkSnippet*
181 s, AgenkCode* c, const char*
182 expected_output);

183 // API Request Context Builder
184 ApiRequestContext* api_request_context_
185 create(const char* url, const char*
186 http_method);
187 void api_request_context_free(
188 ApiRequestContext* ctx);
189 bool api_request_context_add_header(
190 ApiRequestContext* ctx, const char*
191 key, const char* value);
192 bool api_request_context_add_param(
193 ApiRequestContext* ctx, const char*
194 key, const char* value);
195 void api_request_context_set_body(
196 ApiRequestContext* ctx, AgenkTensor*
197 body);

198 // Execution Result
199 void execution_result_free(
200 ExecutionResult* result);

201 #endif // AGENK_SNIPPET_H

```

Listing 31: The public API header defining the universal data structures and the registry for skills ('agenk\_snippet.h').

### 6.2.3 The Implementation

The substrate is implemented across two files, cleanly separating memory management from persistence logic.

**The Builder Pattern (snippet.c):** Because the **AgenkSnippet** struct is a complex, deeply-nested data structure involving multiple levels of dynamic allocation, manual memory management would be extremely error-prone. This file implements the **Builder Pattern**. It provides functions (**snippet\_create**, etc.) that abstract away all direct memory management. The user in-

teracts with these safe, high-level functions, and the library handles all the underlying `malloc`, `realloc`, and `strdup` calls.

Crucially, this pattern is paired with a corresponding set of deep-freeing functions. A single call to `snippet_free` will recursively traverse the entire structure, freeing all allocated memory for the snippet itself, its dependencies, its code variants, and all associated strings. This provides a powerful guarantee of memory safety and is a cornerstone of the system's robustness.

```
1 /**
2 * @file snippet.c
3 * @brief Implements memory management
4 * for the AgenkSnippet data
5 * structures.
6 *
7 * Provides "builder" style functions to
8 * safely construct and deep-free
9 * the complex, nested Snippet structs.
10 *
11 * @version 1.0.0
12 * @author Ankush Yadav, Ankit Yadav,
13 * AuctaSapience
14 */
15
16 #include "include/agenk_snippet.h"
17 // THE FIX: Use the include path
18 // provided by CMake, not a brittle
19 // relative path.
20 // The agenk_snippet library's CMake
21 // definition links against agenk_
22 // tensor,
23 // which provides the necessary include
24 // path for this header.
25 #include "agenk_tensor.h"
26 #include <stdlib.h>
27 #include <string.h>
28
29 // --- Internal Helper ---
30 static char* safe_strdup(const char* s)
31 {
32     if (!s) return NULL;
33     size_t len = strlen(s) + 1;
34     char* new_s = (char*)malloc(len);
35     if (new_s) {
36         memcpy(new_s, s, len);
37     }
38     return new_s;
39 }
40
41 // --- Memory Management for
42 // Dependencies ---
43 AgenkDependency* dependency_create(int
44 type, const char* uri) {
45     AgenkDependency* d = (
46         AgenkDependency*)calloc(1,
47         sizeof(AgenkDependency));
48     if (!d) return NULL;
49     d->type = type;
50     d->uri = safe_strdup(uri);
51     return d;
52 }
53
54 void dependency_add_build_command(
55 AgenkDependency* d, const char*
```

```

41     command) {
42         if (!d || !command) return;
43         d->num_build_commands++;
44         d->build_commands = (char**)realloc(
45             d->build_commands, d->num_build_
46             commands * sizeof(char*));
47         if (!d->build_commands) { d->num_
48             build_commands = 0; return; }
49         d->build_commands[d->num_build_
50             commands - 1] = safe_strdup(
51             command);
52     }
53
54     void dependency_add_required_artifact(
55         AgenkDependency* d, const char*
56         artifact_path) {
57         if (!d || !artifact_path) return;
58         d->num_required_artifacts++;
59         d->required_artifacts = (char**)
60             realloc(d->required_artifacts, d
61             ->num_required_artifacts *
62             sizeof(char*));
63         if (!d->required_artifacts) { d->num_
64             _required_artifacts = 0; return; }
65         d->required_artifacts[d->num_
66             _required_artifacts - 1] = safe_
67             strdup(artifact_path);
68     }
69
70     static void dependency_free(
71         AgenkDependency* d) {
72         if (!d) return;
73         free(d->uri);
74         for (size_t i = 0; i < d->num_build_
75             commands; ++i) free(d->build_
76             commands[i]);
77         free(d->build_commands);
78         for (size_t i = 0; i < d->num_
79             required_artifacts; ++i) free(d
80             ->required_artifacts[i]);
81         free(d->required_artifacts);
82         free(d);
83     }
84
85 // --- Memory Management for Code ---
86 AgenkCode* code_create(int language,
87     const char* entry_point) {
88     AgenkCode* c = (AgenkCode*)calloc(1,
89         sizeof(AgenkCode));
90     if (!c) return NULL;
91     c->language = language;
92     c->entry_point = safe_strdup(entry_
93         point);
94     return c;
95 }
96
97 void code_add_variant(AgenkCode* c, int
98     os, const char* source) {
99     if (!c || !source) return;
100    c->num_variants++;
101    c->variants = (struct CodeVariant*)
102        realloc(c->variants, c->num_
103        variants * sizeof(struct
104        CodeVariant));
105    if (!c->variants) { c->num_variants
106        = 0; return; }
107    c->variants[c->num_variants - 1].
108        target_os = os;
109    c->variants[c->num_variants - 1].

```

```

    source_code = safe_strdup(source
);
}

static void code_free(AgenkCode* c) {
    if (!c) return;
    free(c->entry_point);
    for (size_t i = 0; i < c->num_
        variants; ++i) free(c->variants[
            i].source_code);
    free(c->variants);
    free(c);
}

// --- Memory Management for Snippets
---
AgenkSnippet* snippet_create(const char*
    description) {
    AgenkSnippet* s = (AgenkSnippet*)
        calloc(1, sizeof(AgenkSnippet));
    if (!s) return NULL;
    s->description = safe_strdup(
        description);
    return s;
}

void snippet_add_dependency(AgenkSnippet
    * s, AgenkDependency* d) {
    if (!s || !d) return;
    s->num_dependencies++;
    s->dependencies = (AgenkDependency
        **)realloc(s->dependencies, s->
            num_dependencies * sizeof(
                AgenkDependency*));
    if (!s->dependencies) { s->num_
        dependencies = 0; return; }
    s->dependencies[s->num_dependencies
        - 1] = d;
}

void snippet_set_execution_code(
    AgenkSnippet* s, AgenkCode* c) {
    if (s) s->execution_code = c;
}

void snippet_set_test_code(AgenkSnippet*
    s, AgenkCode* c, const char*
    expected_output) {
    if (s) {
        s->test_code = c;
        s->expected_test_output = safe_
            strdup(expected_output);
    }
}

void snippet_free(AgenkSnippet* s) {
    if (!s) return;
    free(s->snippet_id);
    free(s->description);
    for (size_t i = 0; i < s->num_
        dependencies; ++i) dependency_
        free(s->dependencies[i]);
    free(s->dependencies);
    code_free(s->execution_code);
    code_free(s->test_code);
    free(s->expected_test_output);
    free(s);
}

// --- Memory Management for
}

```

---

```

    ExecutionResult ---
132 void execution_result_free(
    ExecutionResult* result) {
133     if (!result) return;
134     free(result->stdout_buffer);
135     free(result->stderr_buffer);
136     free(result);
137 }

// --- API Request Context Builder
Implementation ---
139 ApiRequestContext* api_request_context_
    create(const char* url, const char*
        http_method) {
140     ApiRequestContext* ctx = (
        ApiRequestContext*)calloc(1,
            sizeof(ApiRequestContext));
141     if (!ctx) return NULL;
142     ctx->url = safe_strdup(url);
143     if (!ctx->url) { free(ctx); return
        NULL; }
144     ctx->http_method = safe_strdup(http_
        method);
145     if (!ctx->http_method) { free(ctx->
        url); free(ctx); return NULL; }
146     return ctx;
147 }

148 void api_request_context_free(
    ApiRequestContext* ctx) {
149     if (!ctx) return;
150     free(ctx->url);
151     free(ctx->http_method);
152     for (size_t i = 0; i < ctx->num_
        headers; ++i) {
153         free(ctx->headers[i].key);
154         free(ctx->headers[i].value);
155     }
156     free(ctx->headers);
157     for (size_t i = 0; i < ctx->num_
        params; ++i) {
158         free(ctx->query_params[i].key);
159         free(ctx->query_params[i].value);
160     }
161     free(ctx->query_params);
162     if (ctx->body) {
163         // This is the call that creates
164         // the dependency
165         tensor_free(ctx->body);
166     }
167     free(ctx);
168 }

169 static bool add_param_to_list(
    ApiRequestParam** list, size_t*
    count, size_t* capacity, const char*
        key, const char* value) {
170     if (*count >= *capacity) {
171         *capacity = (*capacity == 0) ? 8
172             : *capacity * 2;
173         ApiRequestParam* new_list = (
            ApiRequestParam*)realloc(*
                list, *capacity * sizeof(
                    ApiRequestParam));
174         if (!new_list) return false;
175         *list = new_list;
176     }
177     (*list)[*count].key = safe_strdup(
        key);
178 }

```

```

179    (*list)[*count].value = safe_strdup(
180        value);
181    if (!(*list)[*count].key || !(*list)
182        [*count].value) {
183        free((*list)[*count].key);
184        free((*list)[*count].value);
185        return false;
186    }
187    (*count)++;
188    return true;
189}
190
191bool api_request_context_add_header(
192    ApiRequestContext* ctx, const char*
193    key, const char* value) {
194    if (!ctx || !key || !value) return
195        false;
196    for (size_t i = 0; i < ctx->num_
197        headers; ++i) {
198        if (strcmp(ctx->headers[i].key,
199            key) == 0) {
200            char* new_value = safe_
201                strdup(value);
202            if (!new_value) return false
203            ;
204            free(ctx->headers[i].value);
205            ctx->headers[i].value = new_
206                value;
207            return true;
208        }
209    }
210    return add_param_to_list(&ctx->
211        headers, &ctx->num_headers, &ctx
212        ->headers_capacity, key, value);
213}
214
215bool api_request_context_add_param(
216    ApiRequestContext* ctx, const char*
217    key, const char* value) {
218    if (!ctx || !key || !value) return
219        false;
220    return add_param_to_list(&ctx->query
221        _params, &ctx->num_params, &ctx
222        ->params_capacity, key, value);
223}
224
225void api_request_context_set_body(
226    ApiRequestContext* ctx, AgenkTensor*
227    body) {
228    if (!ctx) return;
229    if (ctx->body) {
230        tensor_free(ctx->body);
231    }
232    ctx->body = body;
233}

```

Listing 32: The implementation of the builder pattern for safe memory management ('snippet.c').

**The Snippet Registry (`snippet_registry.c`):** This module is the "librarian" for skills. It acts as a specialized translation layer between the specific `AgenkSnippet` struct and the generic `ObjectStore` backend.

**Insight — The Translation Layer:** The `ObjectStore` is a powerful, generic engine

designed to store any object described by our FlatBuffers schema. It knows nothing about "snippets," "dependencies," or "code variants." The '`snippet_registry`'s job is to be the expert translator. It takes the specific, in-memory `AgenkSnippet` C struct and meticulously translates it into a generic `c_object_t` that the '`ObjectStore`' can understand. It does this by creating named fields for each part of the snippet (e.g., a field named "description," a field containing a nested object named "execution\_code," etc.). This design is powerful because it allows us to use our universal, high-performance persistence engine for a highly specialized data type without the '`ObjectStore`' needing any knowledge of what a "snippet" is. This is a key principle of layered design: lower layers provide general services, while upper layers provide specialized adaptations.

```

1 /**
2  * @file snippet_registry.c
3  * @brief Implements the persistence
4  * layer for AgenkSnippets.
5  *
6  * This module acts as a specialized
7  * adapter between the high-level
8  * AgenkSnippet
9  * data structures and the generic,
10 * underlying ObjectStore. It is
11 * responsible for
12 * translating snippets to and from the
13 * c_object_t format for storage and
14 * retrieval, effectively creating a
15 * database for the agent's skills.
16 *
17 * @version 1.0.0
18 * @author Ankush Yadav, Ankit Yadav,
19 * AuctaSapience
20 */
21
22 #include "object_store.h" // Correct,
23 // direct include
24 #include "include/agenk_snippet.h"
25 #include "md5.h"
26 #include <string.h>
27 #include <stdlib.h>
28 #include <stdio.h>
29
30 static object_store_t* g_store = NULL;
31 #define SNIPPET_TABLE_NAME "snippets"
32
33 // --- Forward Declarations for Internal
34 // Helper Functions ---
35 static c_object_t* snippet_to_c_object(
36     const AgenkSnippet* snippet);
37 static AgenkSnippet* snippet_from_c_
38     object(const c_object_t* obj);
39 static void get_key_for_snippet_id(const
40     char* snippet_id, unsigned char key
41     _out[OS_MAX_KEY_SIZE]);
42 static char* safe_strdup(const char* s)
43 {
44     if (!s) return NULL;
45     char* new_s = malloc(strlen(s) + 1);

```

```

31     if (new_s) strcpy(new_s, s);
32     return new_s;
33 }
34 // =====
35 // SECTION 1: PUBLIC API
36 // =====
37
38 bool snippet_registry_init(object_store_
39     t* store) {
40     if (!store) return false;
41     g_store = store;
42     return (object_store_create_table(g_
43         store, SNIPPET_TABLE_NAME) == FB_
44         _SERIALIZER_OK);
45 }
46 void snippet_registry_shutdown(void) {
47     g_store = NULL;
48 }
49 bool snippet_registry_add(const
50     AgenkSnippet* snippet) {
51     if (!g_store || !snippet || !snippet
52         ->snippet_id) return false;
53     c_object_t* obj = snippet_to_c_
54         object(snippet);
55     if (!obj) return false;
56     unsigned char key[OS_MAX_KEY_SIZE];
57     get_key_for_snippet_id(snippet->
58         snippet_id, key);
59     fb_serializer_status_t status =
60         object_store_put_object(g_store,
61             SNIPPET_TABLE_NAME, key, obj);
62     free_c_object(obj);
63     return (status == FB_SERIALIZER_OK);
64 }
65 AgenkSnippet* snippet_registry_get_by_id(
66     const char* snippet_id) {
67     if (!g_store || !snippet_id) return
68         NULL;
69     unsigned char key[OS_MAX_KEY_SIZE];
70     get_key_for_snippet_id(snippet_id,
71         key);
72     c_object_t* obj = NULL;
73     if (object_store_get_object(g_store,
74         SNIPPET_TABLE_NAME, key, &obj)
75         != FB_SERIALIZER_OK) {
76         return NULL;
77     }
78     AgenkSnippet* snippet = snippet_from
79     _c_object(obj);
80     free_c_object(obj);
81     return snippet;
82 }
83
84 bool snippet_registry_delete(const char*
85     snippet_id) {
86     if (!g_store || !snippet_id) return
87         false;
88     unsigned char key[OS_MAX_KEY_SIZE];
89     get_key_for_snippet_id(snippet_id,
90         key);
91     fb_serializer_status_t status =
92         object_store_delete_object(g_
93             store, SNIPPET_TABLE_NAME, key);
94     return (status == FB_SERIALIZER_OK
95         || status == FB_SERIALIZER_ERROR
96         || status == FB_SERIALIZER_ERROR
97         _LMDB_NOT_FOUND);
98 }
99
100 // =====
101 // SECTION 2: TRANSLATION LOGIC
102 // =====
103
104 // CORRECTED MACRO
105 #define ADD_FIELD(HEAD, TAIL, NEW_FIELD)
106     do { \
107         /* Use a temporary variable to avoid
108             evaluating NEW_FIELD twice */ \
109         c_field_t* __new_field = (NEW_FIELD)
110         ; \
111         if (!__new_field) { /* Only proceed
112             if the new field is not NULL */ \
113             if (!*(HEAD)) { \
114                 *(HEAD) = __new_field; \
115             } else { \
116                 (TAIL)->next = __new_field;
117                 \
118                 (TAIL) = __new_field; /* Only
119                     update tail if a field was
120                     added */ \
121             } \
122         } \
123     } while(0)
124
125 static c_field_t* create_string_array_
126     field(const char* name, char** data,
127         size_t count) {
128     if (count == 0) return NULL;
129     c_field_t* field = (c_field_t*)
130         calloc(1, sizeof(c_field_t));
131     field->name = safe_strdup(name);
132     field->value = (c_field_value_t*)
133         calloc(1, sizeof(c_field_value_t));
134     field->value->type = C_FIELD_VALUE_
135         TYPE_ARRAY_VALUE;
136     c_field_t* array_tail = NULL;
137     for (size_t i = 0; i < count; ++i) {
138         c_field_t* item = (c_field_t*)
139             calloc(1, sizeof(c_field_t));
140         \
141         char item_name[32];
142         snprintf(item_name, sizeof(item_
143             name), "%zu", i);
144         item->name = safe_strdup(item_
145             name);
146         item->value = create_c_string_
147             value(data[i]));
148         ADD_FIELD(&field->value->data.
149             array_values, array_tail,
150             item);
151     }
152     return field;
153 }
154
155 static c_object_t* code_to_c_object(
156     const AgenkCode* code) {
157     if (!code) return NULL;
158     c_object_t* obj = calloc(1, sizeof(c_
159         object_t));
160     obj->object_id = safe_strdup("nested"
161         );

```

```

122     _code"); obj->table_name = safe_
123     strdup("nested");
124     c_field_t* tail = NULL;
125
126     c_field_t* f_lang = calloc(1, sizeof
127         (c_field_t));
128     f_lang->name = safe_strdup("language
129         ");
130     f_lang->value = create_c_int_value(
131         code->language);
132     ADD_FIELD(&obj->fields, tail, f_lang
133         );
134
135     if (code->entry_point) {
136         c_field_t* f_entry = calloc(1,
137             sizeof(c_field_t));
138         f_entry->name = safe_strdup("entry_
139             point");
140         f_entry->value = create_c_string_
141             value(code->entry_point);
142         ADD_FIELD(&obj->fields, tail, f_
143             entry);
144
145         c_field_t* f_variants = calloc(1,
146             sizeof(c_field_t));
147         f_variants->name = safe_strdup("variants");
148         f_variants->value = calloc(1, sizeof
149             (c_field_value_t));
150         f_variants->value->type = C_FIELD_
151             VALUE_TYPE_MAP_VALUE;
152         ADD_FIELD(&obj->fields, tail, f_
153             variants);
154         c_map_pair_t* map_tail = NULL;
155         for (size_t i = 0; i < code->num_
156             variants; i++) {
157             c_map_pair_t* pair = calloc(1,
158                 sizeof(c_map_pair_t));
159             char os_key[4];
160             snprintf(os_key, sizeof(os_key),
161                 "%d", code->variants[i].
162                 target_os);
163             pair->key = safe_strdup(os_key);
164             pair->value = create_c_string_
165                 value(code->variants[i].
166                     source_code);
167             if (!f_variants->value->data.map_
168                 _pairs) f_variants->value->
169                     data.map_pairs = pair;
170             else map_tail->next = pair;
171             map_tail = pair;
172         }
173         return obj;
174     }
175
176     static c_object_t* snippet_to_c_object(
177         const AgenkSnippet* snippet) {
178         c_object_t* obj = calloc(1, sizeof(c_
179             _object_t));
180         obj->object_id = safe_strdup(snippet_
181             ->snippet_id);
182         obj->table_name = safe_strdup(
183             SNIPPET_TABLE_NAME);
184         c_field_t* tail = NULL;
185
186         if (snippet->description) {
187             c_field_t* f = calloc(1, sizeof(
188                 c_field_t));
189             f->name = safe_strdup("description");
190             f->value = create_c_string_value(
191                 (snippet->description));
192             ADD_FIELD(&obj->fields, tail, f
193                 );
194
195         if (snippet->expected_test_output) {
196             c_field_t* f = calloc(1, sizeof(
197                 c_field_t));
198             f->name = safe_strdup("expected_
199                 test_output");
200             f->value = create_c_string_value(
201                 (snippet->expected_test_
202                     output));
203             ADD_FIELD(&obj->fields, tail, f
204                 );
205
206         if (snippet->execution_code) {
207             c_field_t* f = calloc(1, sizeof(
208                 c_field_t));
209             f->name = safe_strdup("execution_
210                 _code");
211             f->value = calloc(1, sizeof(c_
212                 field_value_t));
213             f->value->type = C_FIELD_VALUE_
214                 TYPE_OBJECT_VALUE;
215             f->value->data.object_val = code_
216                 _to_c_object(snippet->
217                     execution_code);
218             ADD_FIELD(&obj->fields, tail, f
219                 );
220
221     dep->type);
222     ADD_FIELD(&obj->fields, tail, f_type
223         );
224
225     if (dep->uri) {
226         c_field_t* f_uri = calloc(1,
227             sizeof(c_field_t));
228         f_uri->name = safe_strdup("uri")
229             ;
230         f_uri->value = create_c_string_
231             value(dep->uri);
232         ADD_FIELD(&obj->fields, tail, f_
233             uri);
234     }
235     ADD_FIELD(&obj->fields, tail, create_
236         _string_array_field("build_
237             commands", dep->build_commands,
238             dep->num_build_commands));
239     ADD_FIELD(&obj->fields, tail, create_
240         _string_array_field("required_
241             artifacts", dep->required_
242             artifacts, dep->num_required_
243             artifacts));
244     return obj;
245 }
```

```

202     ;
203   }
204   if (snippet->test_code) {
205     c_field_t* f = calloc(1, sizeof(
206       c_field_t));
207     f->name = safe_strdup("test_code");
208     f->value = calloc(1, sizeof(c_
209       field_value_t));
210     f->value->type = C_FIELD_VALUE_
211       TYPE_OBJECT_VALUE;
212     f->value->data.object_val = code_
213       _to_c_object(snippet->test_
214       code);
215     ADD_FIELD(&obj->fields, tail, f)
216     ;
217   }
218   if (snippet->dependencies && snippet
219     ->num_dependencies > 0) {
220     c_field_t* f = calloc(1, sizeof(
221       c_field_t));
222     f->name = safe_strdup("dependencies");
223     f->value = calloc(1, sizeof(c_
224       field_value_t));
225     f->value->type = C_FIELD_VALUE_
226       TYPE_ARRAY_VALUE;
227     ADD_FIELD(&obj->fields, tail, f)
228     ;
229     c_field_t* arr_tail = NULL;
230     for(size_t i = 0; i < snippet->
231       num_dependencies; i++) {
232       c_field_t* item = calloc(1,
233         sizeof(c_field_t));
234       char name_buf[32];
235       snprintf(name_buf, sizeof(
236         name_buf), "%zu", i);
237       item->name = safe_strdup(
238         name_buf);
239       item->value = calloc(1,
240         sizeof(c_field_value_t))
241       ;
242       item->value->type = C_FIELD_
243         VALUE_TYPE_OBJECT_VALUE;
244       item->value->data.object_val
245         = dependency_to_c_
246           object(snippet->
247             dependencies[i]);
248       ADD_FIELD(&f->value->data.
249         array_values, arr_tail,
250         item);
251     }
252     return obj;
253   }
254   static AgenkCode* code_from_c_object(
255     const c_object_t* code_obj) {
256     if(!code_obj) return NULL;
257     int lang = 0; char* entry = NULL;
258     for (c_field_t* f = code_obj->fields
259       ; f; f = f->next) {
260       if (strcmp(f->name, "language")
261         == 0) lang = (int)f->value->
262           data.int_val;
263       if (strcmp(f->name, "entry_point"
264         ) == 0) entry = f->value->
265           data.string_val;
266     }
267     AgenkCode* code = code_create(lang,
268       entry);
269     for (c_field_t* f = code_obj->fields
270       ; f; f = f->next) {
271       if (strcmp(f->name, "variants")
272         == 0) {
273         for(c_map_pair_t* pair = f->
274           value->data.map_pairs;
275           pair; pair = pair->next)
276         {
277           code_add_variant(code,
278             atoi(pair->key),
279             pair->value->data.
280               string_val);
281         }
282       }
283     }
284     return code;
285   }
286   static AgenkDependency* dependency_from_
287     c_object(const c_object_t* dep_obj)
288   {
289     if(!dep_obj) return NULL;
290     int type = 0; char* uri = NULL;
291     for(c_field_t* f = dep_obj->fields
292       ; f; f = f->next) {
293       if (strcmp(f->name, "type") == 0)
294         type = (int)f->value->
295           data.int_val;
296       if (strcmp(f->name, "uri") == 0)
297         uri = f->value->data.string_
298           val;
299     }
300     AgenkDependency* dep = dependency_
301       create(type, uri);
302     for(c_field_t* f = dep_obj->fields
303       ; f; f = f->next) {
304       if (strcmp(f->name, "build_
305         commands") == 0 && f->value
306           ->type == C_FIELD_VALUE_TYPE_
307             ARRAY_VALUE) {
308         for(c_field_t* item = f->
309           value->data.array_values
310           ; item; item = item->
311             next) {
312           dependency_add_build_
313             command(dep, item->
314               value->data.string_
315                 val);
316         }
317       }
318       if (strcmp(f->name, "required_
319         artifacts") == 0 && f->value
320           ->type == C_FIELD_VALUE_TYPE_
321             ARRAY_VALUE) {
322         for(c_field_t* item = f->
323           value->data.array_values
324           ; item; item = item->
325             next) {
326           dependency_add_required_
327             artifact(dep, item->
328               value->data.string_
329                 val);
330         }
331     }
332     return dep;
333   }
334   static AgenkSnippet* snippet_from_c_

```

```
274 object(const c_object_t* obj) {
275     if (!obj) return NULL;
276     AgenkSnippet* s = snippet_create(
277         NULL);
278     s->snippet_id = safe_strdup(obj->
279         object_id);
280     for (c_field_t* field = obj->fields;
281         field; field = field->next) {
282         if (!field->value) continue;
283         if (strcmp(field->name, "description") == 0) s->
284             description = safe_strdup(
285                 field->value->data.string_
286                 val);
287         else if (strcmp(field->name, "expected_test_output") == 0)
288             s->expected_test_output =
289                 safe_strdup(field->value->
290                     data.string_val);
291         else if (strcmp(field->name, "execution_code") == 0) s->
292             execution_code = code_from_c_
293                 _object(field->value->data.
294                 object_val);
295         else if (strcmp(field->name, "test_code") == 0) s->test_
296             code = code_from_c_object(
297                 field->value->data.object_
298                 val);
299         else if (strcmp(field->name, "dependencies") == 0) {
300             for(c_field_t* item = field
301                 ->value->data.array_
302                 values; item; item=item_
303                 ->next) {
304                 AgenkDependency* dep =
305                     dependency_from_c_
306                         _object(item->value->
307                             data.object_val);
308                 snippet_add_dependency(s,
309                     dep);
310             }
311         }
312     }
313     return s;
314 }
315
316 static void get_key_for_snippet_id(const
317     char* snippet_id, unsigned char key
318     [out[OS_MAX_KEY_SIZE]]) {
319     md5_state_t pms;
320     md5_byte_t digest[16];
321     md5_init(&pms);
322     md5_append(&pms, (const md5_byte_t*)
323         snippet_id, strlen(snippet_id));
324     md5_finish(&pms, digest);
325     memcpy(key_out, digest, 16);
326     memset(key_out + 16, 0, OS_MAX_KEY_
327         SIZE - 16);
328 }
```

Listing 33: The persistence layer for Snippets, acting as a translation layer to the generic ObjectStore ('snippet registry.c').

This suite validates both the builder pattern and the registry. Its most critical test case, ‘test\_suite\_registry.lifecycle’, performs a full “round-trip” validation: it programmatically builds a complex snippet, saves it to the database via the registry, retrieves it, and then performs a deep, field-by-field comparison to guarantee that the retrieved object is a perfect, lossless reconstruction of the original. The successful execution of this suite provides high confidence in the robustness of the agent’s foundational skill management system.

```
1  /**
2  * @file test_snippets.c
3  * @brief Comprehensive validation suite
4  *        for the Snippets Substrate.
5  *
6  * @version 1.0.0
7  * @author Ankush Yadav, Ankit Yadav,
8  *        AuctaSapience
9  */
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <string.h>
14
15 #include "agenk_snippet.h"
16 #include "object_store.h"
17 #include "object.h"
18
19 // --- Testing Framework Macros ---
20 static int g_tests_run = 0;
21 static int g_tests_passed = 0;
22 #define COLOR_GREEN "\x1B[32m"
23 #define COLOR_RED "\x1B[31m"
24 #define COLOR_RESET "\x1B[0m"
25 #define TEST_SUITE_START(name) printf("\n---\u2022Running\u2022TestSuite:\u2022%s\u2022---\n", name)
26 #define TEST_CASE(name) printf("\u2022\u2022[TEST]\u2022\u2022%s\u2022\u2022\n", name)
27 #define ASSERT(condition, message)
28
29 \ do {
30
31     \
32     g_tests_run++;
33
34     \
35     if (condition) {
36
37         \
38         g_tests_passed++;
39
40         \
41         printf(COLOR_GREEN "\u2022\u2022[PASS]\u2022\u2022%s\u2022\u2022\n", COLOR_RESET, message);
42     } else {
43
44         \
45         fprintf(stderr, COLOR_RED "[FAIL]\u2022%s:\u2022%d:\u2022%s\u2022(%s)\u2022\n",
```

```

33     COLOR_RESET, __FILE__-
34     , __LINE__,
35     message, # condition);
36     \
37 } while (0)
38 // --- Forward Declarations for Test
39 // Helpers ---
40 AgenkSnippet* create_complex_snippet();
41 bool compare_snippets(const AgenkSnippet*
42 * s1, const AgenkSnippet* s2);
43 // --- Test Suites ---
44 void test_suite_registry_lifecycle() {
45     TEST_SUITE_START("Snippet Registry"
46     " Full Round-Trip");
47     object_store_t* store = NULL;
48     ASSERT(object_store_create(&store)
49         == 0, "Object store created");
50     ASSERT(object_store_init_db(store, "./test_snippet_db", 0, 0) == FB_
51     SERIALIZER_OK, "Database initialized");
52     ASSERT(snippet_registry_init(store),
53         "Snippet registry initialized");
54     ;
55     TEST_CASE("Creating a complex snippet"
56     "'snippet_out'");
57     AgenkSnippet* snippet_out = create_
58     complex_snippet();
59     ASSERT(snippet_out != NULL, "snippet
60     _out created successfully");
61     TEST_CASE("Adding snippet_out to the
62     registry");
63     ASSERT(snippet_registry_add(snippet_
64     _out), "snippet_registry_add"
65     succeeded");
66     TEST_CASE("Retrieving snippet by ID"
67     "'snippet_in'");
68     AgenkSnippet* snippet_in = snippet_
69     registry_get_by_id("tools.text."
70     "formatter.v1");
71     ASSERT(snippet_in != NULL, "snippet
72     _registry_get_by_id succeeded");
73     TEST_CASE("Performing deep comparison"
74     "of snippet_out and snippet_in");
75     ASSERT(compare_snippets(snippet_out,
76     snippet_in), "Snippets are"
77     "identical after round-trip");
78     TEST_CASE("Deleting the snippet from
79     the registry");
80     ASSERT(snippet_registry_delete("
81     tools.text.formatter.v1"),
82     "snippet_registry_delete"
83     succeeded");
84     TEST_CASE("Verifying snippet is no
85     longer in the registry");
86     AgenkSnippet* deleted_snippet =
87     snippet_registry_get_by_id("
88     tools.text.formatter.v1");
89     ASSERT(deleted_snippet == NULL, "
90     Verified snippet is deleted");
91     snippet_free(snippet_out);
92     snippet_free(snippet_in);
93     snippet_registry_shutdown();
94
95     object_store_destroy(&store);
96     system("rm ./test_snippet_db");
97 }
98
99 int main() {
100     printf("\n=====\n");
101     printf("Running AGENK Snippets"
102     " Substrate Tests\n");
103     printf("=====\n");
104     CLASS_INIT();
105     test_suite_registry_lifecycle();
106     CLASS_SHUTDOWN();
107     printf("\n=====\\n");
108     printf("TEST SUMMARY\\n");
109     printf("=====\\n");
110     if (g_tests_passed == g_tests_run) {
111         printf(COLOR_GREEN "SUCCESS: All"
112         " %d tests passed.\\n" COLOR_
113         RESET, g_tests_run);
114         return 0;
115     } else {
116         fprintf(stderr, COLOR_RED "
117         FAILURE: %d out of %d tests"
118         " failed.\\n" COLOR_RESET,
119         g_tests_run - g_tests_
120         passed, g_tests_run);
121         ;
122     }
123     return 1;
124 }
125
126 // --- Helper Implementations ---
127
128 AgenkSnippet* create_complex_snippet() {
129     AgenkSnippet* s = snippet_create(""
130     "Formats text to a specified"
131     "width.");
132
133     // THE DEFINITIVE FIX: Do not use
134     // the non-standard strdup().
135     // It causes a compiler warning and
136     // results in an invalid pointer,
137     // leading to a segfault.
138     // Allocate and copy the string
139     // manually using standard C
140     // functions.
141     const char* id_str = "tools.text."
142     "formatter.v1";
143     s->snippet_id = (char*)malloc(strlen
144     (id_str) + 1);
145     if (s->snippet_id) {
146         strcpy(s->snippet_id, id_str);
147     }
148
149     AgenkCode* code = code_create(LANG_C
150     , "format_text");
151     code_add_variant(code, OS_ANY, "//C"
152     " source code would be here.");
153     code_add_variant(code, OS_LINUX, "//"
154     " Linux-specific C source.");
155     snippet_set_execution_code(s, code);
156
157     AgenkDependency* dep1 = dependency_
158     create(DEP_SYSTEM, "libc");
159     snippet_add_dependency(s, dep1);
160 }
```

```

113     return s;
114 }
115
116 bool compare_code(const AgenkCode* c1,
117                   const AgenkCode* c2) {
118     if (!c1 && !c2) return true;
119     if (!c1 || !c2) return false;
120     if (c1->language != c2->language)
121         return false;
122     if (strcmp(c1->entry_point, c2->
123                entry_point) != 0) return false;
124     if (c1->num_variants != c2->num_
125         variants) return false;
126     for(size_t i = 0; i < c1->num_
127         variants; i++) {
128         bool found_match = false;
129         for(size_t j = 0; j < c2->num_
130             variants; j++) {
131             if (c1->variants[i].target_
132                 os == c2->variants[j].
133                 target_os) {
134                 if(strcmp(c1->variants[i].
135                     source_code, c2->
136                     variants[j].source_
137                     code) == 0) {
138                     found_match = true;
139                     break;
140                 }
141             }
142             if(!found_match) return false;
143         }
144     return true;
145 }
146
147 bool compare_snippets(const AgenkSnippet
148                      * s1, const AgenkSnippet* s2) {
149     if (!s1 || !s2) return false;
150     if (strcmp(s1->snippet_id, s2->
151                snippet_id) != 0) return false;
152     if (strcmp(s1->description, s2->
153                description) != 0) return false;
154     if (!compare_code(s1->execution_code
155                       , s2->execution_code)) return
156         false;
157     if (s1->num_dependencies != s2->num_
158         dependencies) return false;
159     for(size_t i = 0; i < s1->num_
160         dependencies; i++) {
161         const AgenkDependency *d1 = s1->
162             dependencies[i];
163         const AgenkDependency *d2 = s2->
164             dependencies[i];
165         if (d1->type != d2->type) return
166             false;
167         if (strcmp(d1->uri, d2->uri) !=
168             0) return false;
169     }
170     return true;
171 }
```

Listing 34: The comprehensive validation suite for the Snippets Substrate ('test\_snippets.c').

## 7 The Execution Substrate: An Autonomous CPU for Acquirable Skills

Having established a universal grammar and persistence mechanism for skills in the Snippets Substrate, we must now construct the machinery that brings these abstract definitions to life. The agent's genome, stored in the Snippet Registry, is inert without the cellular machinery to read, interpret, and express it. This is the exclusive and critical role of the **AGENK Execution Substrate**. This is not merely a library for running scripts; it is the foundational **CPU and Operating System for the agent's skills**. It is a system of pure action, designed to take a self-contained **AgenkSnippet**—a piece of data retrieved from the agent's memory—and transform it into a live, running process.

The entire substrate is architected to answer one fundamental question: *How can an agent safely acquire, build the dependencies for, and execute a new skill it has never seen before, without requiring human intervention or a system-wide software update?*

### 7.1 The Agent's Immune System: A Closed Loop for Safety and Self-Correction

A system designed for open-ended learning—one that can acquire and execute new skills from external or even self-generated sources—faces a fundamental existential risk: how does it protect itself from faulty or malicious code? A single buggy Snippet that causes a crash could destabilize the entire agent. To solve this, we moved beyond simple error handling and architected a complete, closed-loop "immune system" designed to detect, isolate, and ultimately correct faulty skills.

This system is not a single module but an emergent property of the interplay between our foundational substrates and a higher-level cognitive orchestrator. It is built on two core concepts: **Code Safety Flagging** and a **Corrective Introspection Loop**.

#### 7.1.1 Architectural Philosophy: From Crash-and-Restart to Tag-and-Learn

Our design philosophy rejects the brittle "fail-fast" approach common in traditional software. Instead, it models the resilience of a biological immune system. When a skill fails, it is not an unrecoverable error; it is a learning opportunity.

##### 1. The Error as an "Antigen": When

a Snippet fails during execution, the `ExecutionResult` with a non-zero `exit_code` and a detailed `stderr_buffer` acts like a biological “antigen”—a clear and unambiguous signal that something is wrong.

2. **Flagging as “Antibody Tagging”:** The higher-level cognitive core that invoked the skill acts as the “T-cell” of our system. Upon detecting the antigen, its first action is not to discard the failing Snippet but to “tag” it. It modifies a simple boolean flag within the Snippet’s data structure, `is_safe`, to `false` and persists this change back to the Snippet Registry. The skill is now marked as “do not execute” but is perfectly preserved for analysis.
3. **The Execution Engine as the “Enforcer”:** The `engine_execute` function acts as a simple but powerful enforcer of this quarantine. As a primary security measure, its first action is to check the `is_safe` flag. If the flag is `false`, it immediately refuses to execute the code and returns an error, preventing the faulty skill from ever running again until it is explicitly re-certified.
4. **The Introspection Loop as “Immune Memory & Learning”:** This is the most powerful aspect of the design. A quarantined Snippet becomes a high-priority target for a meta-level cognitive process. The orchestrator can spawn a new task, invoking a specialized “Introspection Snippet” whose goal is to analyze the broken code. This introspection tool can use the source code and the captured error message from the failed Snippet as input for a debugging or code-correction model (such as an external LLM API). If a patch is successfully generated, it can be used to create a new, corrected version of the Snippet, which is then submitted back to the registry, re-certified as safe, and deployed for future use.

This creates a complete, closed-loop system for autonomous error correction and learning. It transforms the agent from a passive code executor into an active, self-improving system that learns from its mistakes.

## 7.2 Architectural Pillars

The central architectural principle of the Execution Substrate is its role as a pure, agnostic executor. It is completely decoupled from the origin of the skills it runs. This philosophy is realized through four foundational pillars, as illustrated in Figure 2.

1. **An Autonomous Builder:** The substrate’s first task upon receiving a Snippet is to ensure the environment is prepared. It reads the list of `AgenkDependency` data structures from the Snippet and acts as its own system administrator. It can use system tools like `git` and `make` to build libraries from source, or package managers like `pip` to install packages, managing all artifacts within a private, sandboxed cache directory. This gives the agent the unprecedented ability to dynamically acquire the tools it needs to perform a task.
2. **A Polyglot, Sandboxed Runtime:** Once dependencies are met, the engine delegates the Snippet to the correct “language expert.” It inspects the ‘language’ enum within the `AgenkCode` struct and passes the source code to the appropriate sandboxed environment—be it a C JIT compiler, an embedded Python interpreter, or a simple shell. Crucially, it captures all output, errors, and crashes from these sandboxes to ensure that a faulty skill cannot destabilize the core agent.
3. **The Native Bridge:** A secure “airlock” between the untrusted, sandboxed code running within a Snippet and the trusted, native C code of the agent’s `core` API. This allows guest code to access essential agent capabilities (like creating a tensor or logging to the object framework) in a controlled and safe manner.
4. **The Core Orchestrator (“engine”):** A central controller that manages the entire life-cycle of a skill’s execution. It receives the Snippet, invokes the builder, selects the correct runtime, and returns a standardized `ExecutionResult`, providing a clean and simple interface to the rest of the agent.

## 7.3 Architectural Design and Implementation

The substrate is implemented as a self-contained `src/execution/` module. Its dependencies

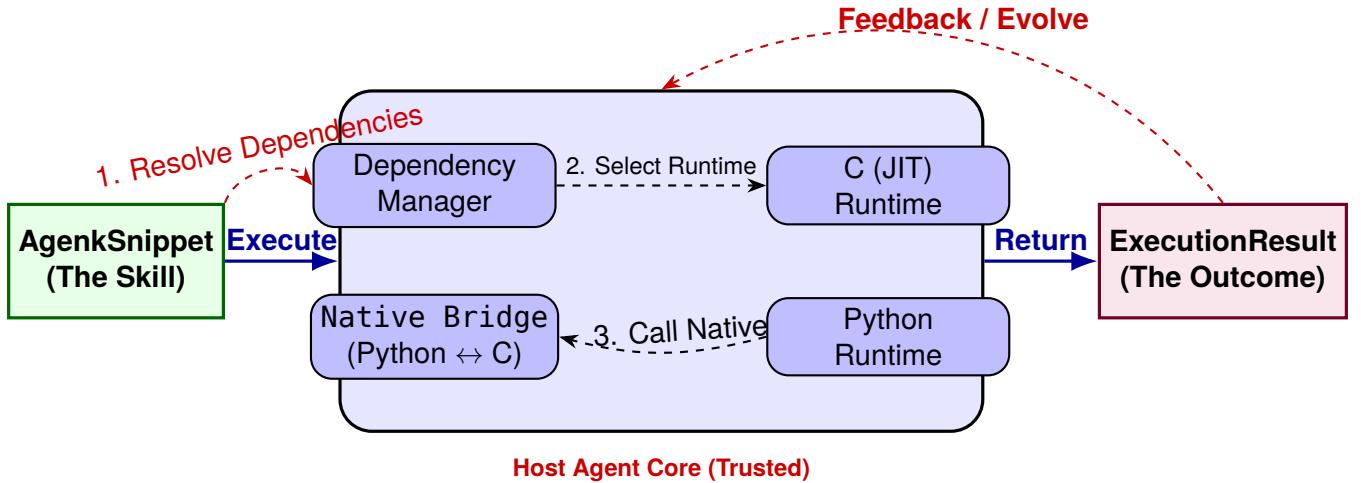


Figure 2: High-level architectural flow of the Execution Substrate. An immutable `AgenSnippet` is passed to the core `engine`, which orchestrates dependency resolution and delegates execution to a sandboxed language runtime. The `Native Bridge` provides a secure interface for guest code to call back into the trusted host agent’s C libraries. The feedback loop represents the agent’s ability to use a skill’s outcome to inform the creation of a new, evolved Snippet.

are strictly one-way: it depends on the `snippets` and `core` modules to understand the data it operates on, but those modules have no knowledge of it.

### 7.3.1 Project Structure and Build System Integration

The module’s filesystem enforces a clean separation of concerns between its core logic, language-specific runtimes, and the critical native bridge.

```

src/
└── execution/
    ├── builder/
    │   └── dependency_manager.c
    ├── core/
    │   └── engine.c
    ├── include/
    │   └── agenk_execution.h
    ├── private/
    │   └── execution_private.h
    ├── runtimes/
    │   ├── runtime_c.c
    │   ├── runtime_python.c
    │   └── runtime_shell.c
    └── test/
        └── test_execution.c

```

```

1 # =====
2 # CMakeLists.txt for the AGENK
3 #
4 # Version: 1.0.0
5 # =====
6
7 # =====
8 # SECTION 1: PROJECT PREAMBLE
9 # & BUILD STANDARDS
10 #
11 cmake_minimum_required(VERSION 3.16)
12 project(AGENK VERSION 1.0.0 LANGUAGES C)
13
14 set(CMAKE_C_STANDARD 99)
15 set(CMAKE_C_STANDARD_REQUIRED ON)
16 set(CMAKE_C_EXTENSIONS OFF)
17 include(ExternalProject)
18 set(CMAKE_ARCHIVE_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/lib)
19 set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/bin)
20
21 if(NOT CMAKE_BUILD_TYPE)
22     set(CMAKE_BUILD_TYPE Debug)
23 endif()
24 message(STATUS "Build type set to: ${CMAKE_BUILD_TYPE}")
25
26 option(AGENK_ENABLE_PYTHON "Enable Python runtime and native bridge" ON)
27
28 option(AGENK_ENABLE_NODEJS "Enable NodeJS/NPM runtime" OFF) #
29 Placeholder for the future
30
31 # =====
32 # SECTION 2: VENDORED &
33 # SYSTEM DEPENDENCY MANAGEMENT
34 #
35 # --- System Libraries ---
36 find_package(Threads REQUIRED)
37 find_library(MATH_LIBRARY m)
38
39 # --- Dependency Discovery ---
40 if(AGENK_ENABLE_PYTHON)
41     message(STATUS "Python support is")

```

```

42      ENABLED .")
43    find_package(Python3 COMPONENTS
44      Development NumPy REQUIRED)
45  else()
46    message(STATUS "Python support is
47      DISABLED .")
48 endif()
49
50 # --- mbed TLS (vendored from deps/) ---
51 set(ENABLE_TESTING OFF CACHE BOOL "
52   Disable_mbedtls_tests")
53 set(ENABLE_PROGRAMS OFF CACHE BOOL "
54   Disable_mbedtls_programs")
55 add_subdirectory(deps/mbedtls)
56 set(MBEDTLS_LIBRARIES mbedtls mbedtlsx509
57     mbedtlscrypto)
58 set(MBEDTLS_LIBRARY_DIR ${CMAKE_ARCHIVE_
59   OUTPUT_DIRECTORY})
60 set(MBEDTLS_INCLUDE_DIR ${CMAKE_CURRENT_
61   SOURCE_DIR}/deps/mbedtls/include)
62
63 # --- libcurl (vendored from deps/, built with custom command) ---
64 ExternalProject_Add(
65   curl_local_build
66   SOURCE_DIR ${CMAKE_CURRENT_SOURCE_
67   DIRECTORY}/deps/curl
68   BINARY_DIR ${CMAKE_BINARY_DIR}/curl-
69   build
70   # Use bash -c to create a reliable
71   # shell environment for the
72   # configure script
73   CONFIGURE_COMMAND bash -c
74     "LDFLAGS='-L${MBEDTLS_LIBRARY_
75       DIRECTORY}' LIBS=' -l mbedtls-
76       mbedtlsx509 -l mbedtlscrypto-
77       -lpthread' CPPFLAGS='-I${
78         MBEDTLS_INCLUDE_DIR}' <
79         SOURCE_DIR>/configure --disable-shared --with-ssl --
80         disable-shared --without-zlib"
81   BUILD_COMMAND make
82   INSTALL_COMMAND ""
83   DEPENDS ${MBEDTLS_LIBRARIES}
84 )
85 add_library(libcurl STATIC IMPORTED
86   GLOBAL)
87 set_target_properties(libcurl PROPERTIES
88   IMPORTED_LOCATION "${CMAKE_BINARY_
89     DIRECTORY}/curl-build/lib/.libs/
90     libcurl.a")
91 add_dependencies(libcurl curl_local_
92   build)
93
94 # --- Other Vendored Libraries ---
95 set(FLATCC_TEST OFF CACHE BOOL "")
96 set(FLATCC_SAMPLES OFF CACHE BOOL "")
97 add_subdirectory(deps/flatcc)
98 add_library(lmdb STATIC deps/lmdb/mdb.c
99   deps/lmdb/midl.c)
100 target_include_directories(lmdb PUBLIC ${
101   CMAKE_CURRENT_SOURCE_DIR}/deps/lmdb
102   )
103 add_library(md5 STATIC deps/md5/md5.c)
104 target_include_directories(md5 PUBLIC ${
105   CMAKE_CURRENT_SOURCE_DIR}/deps/md5)
106 set(CJSON_LIB "")
107 if(EXISTS ${CMAKE_CURRENT_SOURCE_DIR}/
108   deps/cJSON/cJSON.c)
109 add_library(cjson STATIC deps/cJSON/
110   cJSON.c)
111 target_include_directories(cjson
112   PUBLIC ${CMAKE_CURRENT_SOURCE_
113   DIRECTORY}/deps/cJSON)
114 set(CJSON_LIB cJSON)
115 set(CJSON_FOUND TRUE)
116 endif()
117
118 ExternalProject_Add(tcc_external_project
119   SOURCE_DIR ${CMAKE_CURRENT_SOURCE_
120   DIRECTORY}/deps/tcc CONFIGURE_COMMAND ""
121   BUILD_COMMAND "" INSTALL_COMMAND "")
122 add_custom_target(build_tcc_library ALL
123   COMMAND ${CMAKE_COMMAND} -E chdir ${
124     CMAKE_CURRENT_SOURCE_DIR}/deps/tcc
125     ./configure --libdir=${CMAKE_CURRENT_
126     SOURCE_DIR}/deps/tcc/lib --extra-
127     cflags="-fPIC" COMMAND ${CMAKE_
128     COMMAND} -P ${CMAKE_CURRENT_BINARY_
129     DIRECTORY}/cmake_script_patch_tcc_makefile
130     .cmake COMMAND ${CMAKE_COMMAND} -E
131     chdir ${CMAKE_CURRENT_SOURCE_DIR}/
132     deps/tcc make libtcc.a COMMAND ${
133       CMAKE_COMMAND} -E chdir ${CMAKE_
134       CURRENT_SOURCE_DIR}/deps/tcc/lib
135       make COMMAND ${CMAKE_COMMAND} -E
136       copy_if_different ${CMAKE_CURRENT_
137         SOURCE_DIR}/deps/tcc/libtcc.a ${
138           CMAKE_BINARY_DIR}/lib/ COMMAND ${
139             CMAKE_COMMAND} -E copy_if_different
140             ${CMAKE_CURRENT_SOURCE_DIR}/deps/tcc
141             /libtcc1.a ${CMAKE_BINARY_DIR}/lib/
142             DEPENDS tcc_external_project)
143
144 file(WRITE ${CMAKE_CURRENT_BINARY_DIR}/
145   cmake_script_patch_tcc_makefile.
146   cmake "set(TCC_LIB_MAKEFILE_PATH ${
147     CMAKE_CURRENT_SOURCE_DIR}/deps/tcc/
148     lib/Makefile") nfile(READ ${TCC_
149     LIB_MAKEFILE_PATH} TCC_LIB_MAKEFILE_
150     CONTENT) nstring(REPLACE "#bcheck.o"
151     "#bcheck.o" TCC_LIB_MAKEFILE_
152     PATCHED) ${TCC_LIB_MAKEFILE_
153     CONTENT}) nfile(WRITE ${TCC_LIB_
154     MAKEFILE_PATH} ${TCC_LIB_MAKEFILE_
155     PATCHED}) nmessage(STATUS "Patched
156     TCC lib/Makefile to disable bcheck.\n")
157
158 add_library(libtcc STATIC IMPORTED
159   GLOBAL)
160 set_target_properties(libtcc PROPERTIES
161   IMPORTED_LOCATION "${CMAKE_BINARY_
162     DIRECTORY}/lib/libtcc.a")
163 add_dependencies(libtcc build_tcc_
164   library)
165
166 #
167 # =====
168 # SECTION 3: AUTOMATED CODE
169 # GENERATION (from FlatBuffers)
170 #
171
172 set(FBS_SCHEMA ${CMAKE_CURRENT_SOURCE_
173   DIRECTORY}/src/object_store/core/object.
174   fbs)
175 set(GENERATED_INCLUDE_DIR ${CMAKE_
176   CURRENT_BINARY_DIR}/generated/object_
177   _store)
178
179 file(MAKE_DIRECTORY ${GENERATED_INCLUDE_
180   DIRECTORY})
181 set(GENERATED_FILES
182   ${GENERATED_INCLUDE_DIR}/object_
183   builder.h)

```

```

104 ${GENERATED_INCLUDE_DIR}/object_
105     reader.h
106 ${GENERATED_INCLUDE_DIR}/object_
107     verifier.h
108 ${GENERATED_INCLUDE_DIR}/flatbuffers
109     _common.h
110 )
111 add_custom_command(
112     OUTPUT ${GENERATED_FILES}
113     COMMAND $<TARGET_FILE:flatcc_cli> -a
114         -o ${GENERATED_INCLUDE_DIR} ${FBS_SCHEMA}
115     DEPENDS ${FBS_SCHEMA} flatcc_cli
116     COMMENT "Generating C headers from FlatBuffers schema"
117 )
118 add_custom_target(generate_object_
119     headers DEPENDS ${GENERATED_FILES})
120 #
121 # =====
122 # SECTION 4: AGENK CORE
123 # LIBRARY DEFINITIONS
124 # =====
125 # --- LIBRARY 1: object store (The DataBase) ---
126 # --- NEW LIBRARY: object_store (The Persistence Layer) ---
127 add_library(object_store STATIC
128     src/object_store/core/fb_serializer.c
129     src/object_store/core/object_store.c)
130 # This library depends on the auto-generated FlatBuffers headers
131 add_dependencies(object_store generate_object_headers)
132 # Tell the compiler where to find its headers
133 target_include_directories(object_store PUBLIC
134     ${CMAKE_CURRENT_SOURCE_DIR}/src/object_store
135     ${GENERATED_INCLUDE_DIR}
136     ${CMAKE_CURRENT_SOURCE_DIR}/deps/flatcc/include)
137 #
138 # Tell the linker what other libraries it needs
139 target_link_libraries(object_store PRIVATE
140     flatccrt # The FlatBuffers runtime library
141     lmdb
142     md5
143     Threads::Threads
144     ${MATH_LIBRARY} # ADD THIS LINE
145 )
146 #
147 # Handle the optional dependency on cJSON
148 if(CJSON_FOUND)
149     target_link_libraries(object_store PRIVATE ${CJSON_LIB})
150     target_compile_definitions(object_store PRIVATE CJSON_ENABLED)
151 endif()
152 #
153 # --- LIBRARY 2: agenk_tensor (The Data Canvas) ---
154 add_library(agenk_tensor STATIC
155     src/tensors/tensor_lifecycle.c
156     src/tensors/tensor_ops.c
157     src/tensors/tensor_views.c
158     src/tensors/tensor_materialize.c)
159 )
160 target_include_directories(agenk_tensor PUBLIC
161     ${CMAKE_CURRENT_SOURCE_DIR}/src/tensors/include)
162 )
163 #
164 # --- LIBRARY 3: agenk_snippet (The Data Structures & Registry) ---
165 add_library(agenk_snippet STATIC
166     src/snippets/snippet.c
167     src/snippets/snippet_registry.c # <-- ADD NEW SOURCE FILE)
168 )
169 target_include_directories(agenk_snippet PUBLIC
170     ${CMAKE_CURRENT_SOURCE_DIR}/src/snippets/include)
171 )
172 # Snippets now need the ObjectStore for persistence and Tensors for data types.
173 target_link_libraries(agenk_snippet PRIVATE
174     agenk_tensor
175     object_store
176     md5 # Dependency of the registry for key generation)
177 )
178 #
179 # --- LIBRARY 4: class_framework (The Live Object Runtime) ---
180 add_library(class_framework STATIC
181     src/objects/core/object_impl.c
182     src/objects/core/object_registry.c
183     src/objects/core/object_db.c)
184 )
185 target_include_directories(class_framework PUBLIC
186     ${CMAKE_CURRENT_SOURCE_DIR}/src/objects)
187 target_link_libraries(class_framework PRIVATE lmdb Threads::Threads)
188 #
189 # --- LIBRARY 5: agenk_core (The "Kernel / Stable Native API) ---
190 set(CORE_SOURCES src/core/core.c)
191 if(AGENK_ENABLE_PYTHON)
192     list(APPEND CORE_SOURCES src/core/bridge/native_bridge_python.c)
193 endif()
194 add_library(agenk_core STATIC ${CORE_SOURCES})
195 target_include_directories(agenk_core PUBLIC
196     ${CMAKE_CURRENT_SOURCE_DIR}/src/core/include)
197 )
198 if(AGENK_ENABLE_PYTHON)
199     target_include_directories(agenk_core PUBLIC
200     ${Python3_INCLUDE_DIRS})

```

```

201      ${Python3_NumPy_INCLUDE_DIRS}
202    )
203  endif()
204 target_link_libraries(agenk_core PUBLIC
205   agenk_snippet agenk_tensor class_
206   framework
207 )  

208 if(AGENK_ENABLE_PYTHON)
209   target_link_libraries(agenk_core
210     PRIVATE Python3::Python Python3
211     ::NumPy)
212 endif()
213 add_library(agenk_sensation STATIC
214   src/senses/core/sense_manager.c
215   src/senses/core/sense_registry.c
216   # ADD THESE MISSING SOURCE FILES
217   src/senses/builder/build_controller.
218   c
219   src/senses/builder/tool_check.c
220 )  

221 target_include_directories(agenk_
222   sensation PUBLIC
223   ${CMAKE_CURRENT_SOURCE_DIR}/src/
224   senses/include
225 )  

226 target_compile_definitions(agenk_
227   sensation PRIVATE AGENK_SOURCE_DIR=
228   "${CMAKE_SOURCE_DIR}")
229 target_link_libraries(agenk_sensation
230   PRIVATE
231   agenk_core
232   md5
233   libtcc
234   object_store # This must be here
235 )  

236 # --- LIBRARY 6: agenk_execution (The "CPU" / Execution Engine) ---
237 set(EXECUTION_SOURCES
238   src/execution/core/engine.c
239   src/execution/builder/dependency_
240     manager.c
241   src/execution/runtimes/runtime_c.c
242   src/execution/runtimes/runtime_shell
243     .c
244 )  

245 if(AGENK_ENABLE_PYTHON)
246   list(APPEND EXECUTION_SOURCES src/
247     execution/runtimes/runtime_
248     python.c)
249 endif()
250 add_library(agenk_execution STATIC ${  

251   EXECUTION_SOURCES})  

252  

253 target_include_directories(agenk_
254   execution PUBLIC
255   ${CMAKE_CURRENT_SOURCE_DIR}/src/
256   execution/include
257   PRIVATE
258   ${CMAKE_CURRENT_SOURCE_DIR}/src/
259   execution
260   AGENK_TCC_PATH="${CMAKE_CURRENT_
261   SOURCE_DIR}/deps/tcc"
262   # THE FIX: Provide a reliable path
263   to the *built* TCC libraries
264   AGENK_TCC_LIB_PATH="${CMAKE_ARCHIVE_
265   OUTPUT_DIRECTORY}"
266   # Provide include directories for
267   runtime compilation
268   AGENK_CORE_INCLUDE_DIR="${CMAKE_
269   CURRENT_SOURCE_DIR}/src/core/
270     include"
271   AGENK_SNIPPET_INCLUDE_DIR="${CMAKE_
272   CURRENT_SOURCE_DIR}/src/snippets
273     /include"
274 )  

275 if(AGENK_ENABLE_PYTHON)
276   # Add compile definitions needed for
277   the Python runtime.
278   # - AGENK_PYTHON_SUPPORT enables all
279   Python-related code blocks.
280   # - PYTHON_SITE_PACKAGES tells the C
281   code where to find system
282   packages.
283 target_compile_definitions(agenk_
284   execution PRIVATE
285   AGENK_PYTHON_SUPPORT
286   "PYTHON_SITE_PACKAGES=\\"${
287     Python3_SITELIB}\\""  

288 )  

289 endif()  

290 target_link_libraries(agenk_execution
291   PRIVATE
292   agenk_core agenk_snippet libtcc
293     Threads::Threads md5
294 )  

295 if(AGENK_ENABLE_PYTHON)
296   target_link_libraries(agenk_
297     execution PRIVATE Python3::  

298     Python)
299 endif()  

300  

301 # =====
302 # SECTION 5: EXECUTABLE
303 # TARGETS (TESTS & EXAMPLES)
304 # =====
305  

306 # --- EXECUTABLES ---
307 add_executable(test_tensors src/tensors/
308   test/test_tensors.c)
309 target_link_libraries(test_tensors
310   PRIVATE agenk_tensor)  

311  

312 add_executable(test_sensation src/senses
313   /test/test_sensation.c)
314 target_link_libraries(test_sensation
315   PRIVATE agenk_sensation)  

316  

317 add_executable(test_sensation_build src/
318   senses/test/test_sensation_build.c)
319 target_link_libraries(test_sensation_
320   build PRIVATE agenk_sensation)  

321  

322 add_executable(test_sense_registry src/
323   senses/test/test_sense_registry.c)
324 target_link_libraries(test_sense_
325   registry PRIVATE agenk_sensation ${

```

```

        MATH_LIBRARY})
294 add_executable(test_concurrency src/
    objects/test/test_concurrency.c)
295 target_link_libraries(test_concurrency
    PRIVATE class_framework Threads:::
    Threads)
296
297 add_executable(test_errors src/objects/
    test/test_errors.c)
298 target_link_libraries(test_errors
    PRIVATE class_framework)
299
300 add_executable(test_hierarchy src/
    objects/test/test_hierarchy.c)
301 target_link_libraries(test_hierarchy
    PRIVATE class_framework)
302
303 add_executable(test_stress src/objects/
    test/test_stress.c)
304 target_link_libraries(test_stress
    PRIVATE class_framework)
305
306 add_executable(example_basic_usage src/
    objects/examples/basic_usage.c)
307 target_link_libraries(example_basic_
    usage PRIVATE class_framework)
308
309 add_executable(example_lifecycle src/
    objects/examples/lifecycle_and_
    groups.c)
310 target_link_libraries(example_lifecycle
    PRIVATE class_framework)
311
312 add_executable(test_object_store src/
    object_store/test/test_object_store.
    c)
313 target_link_libraries(test_object_store
    PRIVATE object_store Threads:::
    Threads ${CJSON_LIB})
314
315 # --- NEW EXECUTABLE for Snippet testing
316     -
317 add_executable(test_snippets src/
    snippets/test/test_snippets.c)
318 target_link_libraries(test_snippets
    PRIVATE
    agenk_snippet
    class_framework # For CLASS_INIT
    object_store # For the registry's
        backend
319
320 )
321
322 add_executable(test_execution src/
    execution/test/test_execution.c)
323
324 if(AGENK_ENABLE_PYTHON)
325     target_compile_definitions(test_
        execution PRIVATE AGENK_PYTHON_
        SUPPORT) # <-- ADD THIS LINE
            BACK
326     # THE DEFINITIVE FIX for systems
        using GNU ld or clang ld.
327     # This consolidates all linking into
        a single, ordered command.
328     if(CMAKE_SYSTEM_NAME STREQUAL "Linux"
        " OR CMAKE_SYSTEM_NAME STREQUAL
        "Darwin")
329         target_link_libraries(test_
            execution PRIVATE
330             # --- Libraries used directly by
            # the test C code ---
            agenk_execution
            class_framework
331             # --- Force-load the python
            # bridge, THEN provide its
            # dependencies ---
            "-Wl,--whole-archive"
            agenk_core
            "-Wl,--no-whole-archive"
            agenk_snippet
            agenk_tensor
332             # --- Provide all other system/
            # vendored dependencies ---
            lmdb
            Threads::Threads
            Python3::Python
            Python3::NumPy
        )
333     else()
334         # Fallback for other systems (e.g
            # Windows).
            target_link_libraries(test_
                execution PRIVATE
                agenk_execution class_
                    framework agenk_core agenk_
                    snippet
                    agenk_tensor lmdb Threads:::
                    Threads Python3::Python
            )
335     endif()
336 else()
337     # Link normally if Python is
            disabled
            target_link_libraries(test_execution
                PRIVATE
                agenk_execution class_framework
                agenk_core agenk_snippet
                agenk_tensor lmdb Threads:::
                Threads
            )
338     endif()
339
340     # =====
341     # SECTION 6: CTEST
342     # INTEGRATION
343     # =====
344     enable_testing()
345
346     # --- TEST ---
347     add_test(NAME Snippets.
            LifecycleAndRegistry COMMAND test_
            snippets)
348     add_test(NAME Execution.Substrate
            COMMAND test_execution)
349     add_test(NAME Tensor.Substrate
            COMMAND test_tensors)
350     add_test(NAME Sensation.Subsystem.Core
            COMMAND test_sensation)
351     add_test(NAME Sensation.Subsystem.Build
            COMMAND test_sensation_build)
352     add_test(NAME Sensation.Registry COMMAND

```

```

  test_sense_registry)
382 add_test(NAME ObjectStore.Core
            COMMAND test_object_store)
383 add_test(NAME FrameworkConcurrency
            COMMAND test_concurrency)
384 add_test(NAME Framework.ErrorHandling
            COMMAND test_errors)
385 add_test(NAME Framework.Hierarchy
            COMMAND test_hierarchy)
386 add_test(NAME Framework.Stress
            COMMAND test_stress)

```

Listing 35: The ‘agenk\_execution’ library definition in the root CMakeLists.txt showing its dependencies and compile-time definitions.

### 7.3.2 Anatomy of the Substrate

The substrate’s power comes from the clean separation of concerns between its modules. Each component has a single, well-defined responsibility.

#### The Public Contract (`agenk_execution.h`):

This header is the formal, stable contract for the entire subsystem. It is intentionally minimal, defining only the functions essential for the substrate’s operation and getting all its data structure definitions by including the foundational `agenk_snippet.h`. This enforces our layered architecture.

```

1 /**
2  * @file agenk_execution.h
3  * @brief Public API for the AGENK
4  *        Execution Substrate.
5  *
6  * This header defines the functions for
7  * the agent's "CPU".
8  * It gets its data structure
9  * definitions from the foundational
10 * agenk_snippet.h header.
11 *
12 * @version 1.0.0
13 * @author Ankush Yadav, Ankit Yadav,
14 *        AuctaSapience
15 */
16
17 #ifndef AGENK_EXECUTION_H
18 #define AGENK_EXECUTION_H
19
20 // THIS IS THE FIX:
21 // Include the single source of truth
22 // for data structures.
23 // All the struct definitions that were
24 // previously here have been REMOVED.
25 #include "agenk_snippet.h"
26
27 // --- Substrate Lifecycle ---
28 bool engine_init(const char* cache_path)
29 ;
30 void engine_shutdown();
31
32 // --- Core Execution Function ---
33 ExecutionResult* engine_execute(const
34 AgenkSnippet* snippet, const
35 ExecutionContext* context);

```

```

27
28
29 #endif // AGENK_EXECUTION_H

```

Listing 36: The clean, final public API for the Execution Substrate.

**The Private Contract (`execution_private.h`):** This header is the internal “nervous system” of the substrate, providing the full struct definitions and internal function prototypes needed to connect the different .c modules. It is a critical piece of the design, ensuring type-safe linkage and providing our conditional debugging macros.

```

1 /**
2  * @file execution_private.h
3  * @brief Private header for the AGENK
4  *        Execution Substrate implementation.
5  *
6  * Contains full struct definitions,
7  * internal function prototypes, and
8  * debug macros.
9  * This file should only ever be
10 * included by .c files within the
11 * execution/ module.
12 *
13 * @version 1.0.0
14 * @author Ankush Yadav, Ankit Yadav,
15 *        AuctaSapience
16 */
17
18 #ifndef EXECUTION_PRIVATE_H
19 #define EXECUTION_PRIVATE_H
20
21 #include "../include/agenk_execution.h"
22 #include <stdio.h>
23 #include <assert.h>
24
25 // --- Opaque struct from Python.h
26 // needed for prototype ---
27 struct _object;
28 typedef struct _object PyObject;
29
30 // --- Global Engine State ---
31 typedef struct {
32     bool is_initialized;
33     char* cache_path;
34 } EngineState;
35
36 // --- Debugging and Error Handling
37 // Macros ---
38 #ifndef NDEBUG
39     #define EXEC_LOG(format, ...) \
40         fprintf(stdout, "[EXEC_LOG] %s:\n" \
41             format "\n", __FILE__, \
42             __LINE__, ##VA_ARGS__)
43     #define EXEC_LOG_ERROR(format, ...) \
44         \fprintf(stderr, "[EXEC_ERROR] %s:\n" \
45             format "\n", __FILE__, \
46             __LINE__, ##VA_ARGS__)
47     #define EXEC_ASSERT(condition) \
48         assert(condition)
49 #else
50
51 #endif

```

```

38 #define EXEC_LOG(format, ...) ((void)0)
39 #define EXEC_LOG_ERROR(format, ...) ((void)0)
40 #define EXEC_ASSERT(condition) ((void)0)
41 #endif
42
43 // --- Internal Function Prototypes ---
44
45 // From engine.c
46 ExecutionResult* create_error_result(
47     const char* error_message);
48
49 // From builder/dependency_manager.c
50 bool resolve_dependencies(const AgenkSnippet* snippet);
51 // NEW: Internal handlers for different dependency types
52 bool handle_source_dependency(const AgenkDependency* dep);
53 bool handle_pip_dependency(const AgenkDependency* dep);
54 bool handle_npm_dependency(const AgenkDependency* dep);
55 // THE FIX: Added prototype for run_command, making it visible to runtime_shell.c
56 int run_command(const char* command,
57     const char* working_dir, char** out_stdout, char** out_stderr);
58
59 // From runtimes/
60 ExecutionResult* execute_c_snippet(const AgenkCode* code, const
61     ExecutionContext* context);
62 bool python_runtime_init();
63 void python_runtime_shutdown();
64 ExecutionResult* execute_python_snippet(
65     const AgenkCode* code, const
66     ExecutionContext* context);
67 ExecutionResult* execute_shell_snippet(
68     const char* command_string, const
69     ExecutionContext* context);
70
71 // From bridge/native_bridge_python.c
72 // THE FIX: Added prototype for the native module init function.
73 PyObject* PyInit_agenk_core(void);
74 // THE FIX: Added prototype for the native module init function.
75 // PyObject* PyInit_agenk_core(void);
76
77 // --- Utility Functions ---
78 char* safe_strdup(const char* s);
79 bool ensure_dir_exists(const char* path);
80 const char* get_os_string();
81
82 #endif // EXECUTION_PRIVATE_H

```

Listing 37: The private internal header for the Execution Substrate.

**The Core Engine and Orchestrator (`engine.c`):** This module is the central nervous system of the Execution Substrate. It implements the primary public function,

`engine_execute()`, which acts as the single point of entry for all skill execution requests. The engine is an orchestrator: it validates the incoming `AgenkSnippet`, delegates dependency resolution, intelligently selects the correct code variant for the host OS, and finally, passes the skill to the appropriate language runtime specialist for execution.

```
// Paste the full content of engine.c here
```

Listing 38: The central orchestrator for the Execution Substrate ('`engine.c`').

**The Autonomous Builder (`dependency_manager.c`):** This module gives the agent its "hands," allowing it to interact with the host system to acquire the tools it needs. It orchestrates the entire dependency resolution pipeline in a portable, cross-platform manner.

**Insight — Robust and Portable Process Management:** A critical lesson learned was the danger of using platform-specific or overly simplistic methods for running external commands. To achieve true portability, we implemented the `run_command` function using conditional compilation. For POSIX systems, it uses the canonical `fork()` and `execvp()` pattern with pipes to safely capture all I/O. For Windows, it would use the native Win32 `CreateProcessA` API with the same level of I/O redirection, ensuring identical behavior across platforms.

```

1 /**
2 * @file dependency_manager.c
3 * @brief Implements the autonomous,
4 *        cross-platform dependency
5 *        resolution and
6 *        build system for the AGENK
7 *        Execution Substrate.
8 *
9 * This module is a core component of
10 * the Execution Substrate,
11 * responsible for
12 * giving the agent the ability to
13 * manage its own software
14 * dependencies. It can
15 * check for required tools, fetch
16 * source code from Git, execute build
17 * commands,
18 * and manage a local cache of artifacts
19 * in a way that is portable across
20 * Windows and POSIX-compliant systems (
21 * Linux, macOS).
22 *
23 * The design prioritizes robustness,
24 * portability, and security:

```

```

13 * - All external commands are run in
14     isolated child processes with
15     captured I/O,
16 * using the native process management
17     API for each platform (`CreateProcess` on
18     Windows, `fork`/`exec` on POSIX).
19 * - Build artifacts are stored in a
20     unique, deterministic cache
21     directory to
22     prevent conflicts between different
23     dependencies or concurrent builds.
24 * - All buffer operations are performed
25     safely to prevent overflows.
26 */
27
28 * @version 1.0.0
29 * @author Ankush Yadav, Ankit Yadav,
30     AuctaSapience
31 */
32
33 #include "../private/execution_private.h"
34
35 #include "md5.h"
36 #include <stdio.h>
37 #include <stdlib.h>
38 #include <string.h>
39 #include <sys/stat.h>
40 #include <errno.h>
41
42 #ifdef _WIN32
43 #include <windows.h>
44 #else
45 #include <unistd.h>
46 #include <sys/wait.h>
47 #include <fcntl.h>
48 #endif
49
50 // Define a max path length for buffers
51     to avoid magic numbers
52 #define MAX_PATH_LEN 1024
53 #define MAX_CMD_LEN 2048
54
55 // The global engine state, defined in
56     engine.c, provides the cache path.
57 extern EngineState g_engine_state;
58
59 //=====
60 // SECTION 1: FORWARD DECLARATIONS
61 // FOR STATIC HELPERS
62 //=====
63
64 static void get_dep_unique_hash(const
65     AgenkDependency* dep, char hash_out
66     [33]);
67 static bool check_artifacts_exist(const
68     AgenkDependency* dep);
69 static bool install_artifacts(const
70     AgenkDependency* dep, const char*
71     source_dir);
72
73 //=====
74 // SECTION 2: CROSS-PLATFORM
75 // COMMAND EXECUTION
76 //=====
77
78 #ifndef _WIN32
79 static char* read_all_from_fd(int fd) {
80     size_t capacity = 4096;
81     size_t size = 0;
82     char* buffer = (char*)malloc(
83         capacity);
84     if (!buffer) return NULL;
85     ssize_t bytes_read;
86     while ((bytes_read = read(fd, buffer
87         + size, capacity - size - 1)) >
88         0) {
89         size += bytes_read;
90         if (size + 1 >= capacity) {
91             capacity *= 2;
92             char* new_buffer = (char*)
93                 realloc(buffer, capacity
94                 );
95             if (!new_buffer) { free(
96                 buffer); return NULL; }
97             buffer = new_buffer;
98         }
99     }
100    buffer[size] = '\0';
101    return buffer;
102 }
103#endif
104
105 int run_command(const char* command,
106     const char* working_dir, char** out_
107     stdout, char** out_stderr) {
108 #ifdef _WIN32
109     // Windows implementation using
110     // CreateProcess would go here.
111     return -1;
112 #else
113     int stdout_pipefd[2];
114     int stderr_pipefd[2];
115     pid_t pid;
116
117     if (pipe(stdout_pipefd) != 0 || pipe
118         (stderr_pipefd) != 0) return -1;
119     pid = fork();
120     if (pid == -1) return -1;
121
122     if (pid == 0) { // Child
123         close(stdout_pipefd[0]); // Close
124         // read end of stdout
125         pipe
126         close(stderr_pipefd[0]); // Close
127         // read end of stderr
128         pipe
129         dup2(stdout_pipefd[1], STDOUT_
130             FILENO);
131         dup2(stderr_pipefd[1], STDERR_
132             FILENO);
133         close(stdout_pipefd[1]); // Close
134         // write end after dup
135         close(stderr_pipefd[1]); // Close
136         // write end after dup
137         if (working_dir && chdir(working
138             _dir) != 0) _exit(127);
139         execl("/bin/sh", "sh", "-c",
140             command, (char *)NULL);
141         _exit(127); // execl only
142         returns on error
143     }
144
145     // Parent
146     close(stdout_pipefd[1]); // Close
147     // write end of stdout pipe
148     close(stderr_pipefd[1]); // Close
149     // write end of stderr pipe
150     if (out_stdout) *out_stdout = read_
151         _all_from_fd(stdout_pipefd[0]);
152     if (out_stderr) *out_stderr = read_

```

```

112     all_from_fd(stderr_pipefd[0]);
113     close(stdout_pipefd[0]); // Close
114         read end
115     close(stderr_pipefd[0]); // Close
116         read end
117     int status;
118     waitpid(pid, &status, 0);
119     return WIFEXITED(status) ?
120         WEXITSTATUS(status) : -1;
121 #endif
122 }
123 //=====
124 // SECTION 3: DEPENDENCY MANAGEMENT
125 // LOGIC (Now Cross-Platform)
126 //=====
127 /**
128 * @brief Generates a unique,
129     deterministic 32-character hex
130     string hash for a dependency.
131 */
132 static void get_dep_unique_hash(const
133 AgenkDependency* dep, char hash_out
134 [33]) {
135     md5_state_t pms;
136     md5_byte_t digest[16]; // FIX: Was a
137         single byte, needs to be an
138         array of 16
139     md5_init(&pms);
140     md5_append(&pms, (const md5_byte_t*)
141         dep->uri, strlen(dep->uri));
142     md5_finish(&pms, digest);
143     for(int i = 0; i < 16; ++i) {
144         sprintf(&hash_out[i * 2], "%02x"
145             , digest[i]);
146     }
147     hash_out[32] = '\0'; // FIX: Null-
148         terminate the string
149 }
150 /**
151 * @brief Checks if all required
152     artifacts for a dependency already
153     exist in the cache.
154 */
155 static bool check_artifacts_exist(const
156 AgenkDependency* dep) {
157     char path_buffer[MAX_PATH_LEN]; // //
158         FIX: Was a single char, now a
159         buffer
160     for (size_t i = 0; i < dep->num_
161         required_artifacts; ++i) {
162         snprintf(path_buffer, sizeof(
163             path_buffer), "%s/%s", g_
164             engine_state.cache_path, dep
165             ->required_artifacts[i]);
166         struct stat st;
167         if (stat(path_buffer, &st) != 0)
168             return false;
169     }
170     return dep->num_required_artifacts >
171         0;
172 }
173 /**
174 * @brief Copies the required build
175     artifacts from a temp build
176     directory to the agent's cache.
177 */
178 static bool install_artifacts(const
179 AgenkDependency* dep, const char*
180 source_dir) {
181     char command[MAX_CMD_LEN]; // FIX:
182         Was a single char, now a buffer
183     int rc;
184     for (size_t i = 0; i < dep->num_
185         required_artifacts; ++i) {
186         #ifdef _WIN32
187             sprintf(command, sizeof(command)
188                 , "xcopy /E /I /Y %s\\%s"
189                     "\\%s\\\"", source_dir,
190                     dep->required_artifacts[i],
191                     g_engine_state.cache_path);
192         #else
193             sprintf(command, sizeof(command)
194                 , "cp -r %s/%s \"%s\" \"%s\""
195                     , source_dir, dep->required_
196                     artifacts[i], g_engine_state
197                     .cache_path);
198         #endif
199         rc = run_command(command, ".",
200             NULL, NULL);
201         if (rc != 0) { EXEC_LOG_ERROR("Failed to install artifact: %s",
202             dep->required_artifacts[i]); return false; }
203     }
204     return true;
205 }
206 /**
207 * @brief Handles dependencies that must
208     be built from source (e.g., from
209     Git).
210 */
211 bool handle_source_dependency(const
212 AgenkDependency* dep) {
213     char dep_hash[33]; // FIX: Sized for
214         32 hex chars + null terminator
215     get_dep_unique_hash(dep, dep_hash);
216     char build_dir[MAX_PATH_LEN]; // FIX
217         : Was a single char, now a
218         buffer
219     snprintf(build_dir, sizeof(build_dir)
220         , "%s/builds/%s", g_engine_
221             state.cache_path, dep_hash);
222     ensure_dir_exists(build_dir);
223
224     char command[MAX_CMD_LEN]; // FIX:
225         Was a single char, now a buffer
226     snprintf(command, sizeof(command), "git clone --depth 1 %s.", dep->
227         uri);
228     if (run_command(command, build_dir,
229         NULL, NULL) != 0) { return false;
230     }
231
232     for (size_t i = 0; i < dep->num_
233         build_commands; ++i) {
234         if (run_command(dep->build_
235             commands[i], build_dir, NULL
236             , NULL) != 0) { return false;
237     }
238
239     if (!install_artifacts(dep, build_
240         dir)) return false;
241     return true;
242 }
```

```

192 /**
193 * @brief Handles Python package
194 * dependencies using PIP.
195 */
196 bool handle_pip_dependency(const
197 AgenkDependency* dep) {
198     char python_cache_dir[MAX_PATH_LEN];
199     // FIX: Was a single char, now
200     // a buffer
201     snprintf(python_cache_dir, sizeof(
202         python_cache_dir), "%s/python_
203         packages", g_engine_state.cache_
204         path);
205     ensure_dir_exists(python_cache_dir);
206     char command[MAX_CMD_LEN]; // FIX:
207     // Was a single char, now a buffer
208     snprintf(command, sizeof(command), "
209         python3\u2014m\u2014pip\u2014install\u2014%s\u2014
210         target\u2014\"%s\"", dep->uri, python_
211         cache_dir);
212     if (run_command(command, ".", NULL,
213         NULL) != 0) { return false; }
214     return true;
215 }
216 /**
217 * @brief Handles NodeJS package
218 * dependencies using NPM.
219 */
220 bool handle_npm_dependency(const
221 AgenkDependency* dep) {
222     char node_cache_dir[MAX_PATH_LEN];
223     // FIX: Was a single char, now a
224     // buffer
225     snprintf(node_cache_dir, sizeof(node_
226         _cache_dir), "%s/node_modules_
227         cache", g_engine_state.cache_
228         path);
229     ensure_dir_exists(node_cache_dir);
230     char command[MAX_CMD_LEN]; // FIX:
231     // Was a single char, now a buffer
232     snprintf(command, sizeof(command), "
233         npm\u2014install\u2014%s\u2014prefix\u2014\"%s\",
234         dep->uri, node_cache_dir);
235     if (run_command(command, ".", NULL,
236         NULL) != 0) { return false; }
237     return true;
238 }
239 /**
240 * @brief Handles system-level
241 * dependencies (currently a no-op).
242 */
243 bool handle_system_dependency(const
244 AgenkDependency* dep) {
245     EXEC_LOG("Assuming system dependency
246         \"%s\" is met.", dep->uri);
247     return true;
248 }
249 /**
250 * @brief Resolves all dependencies
251 * listed in a Snippet.
252 */
253 bool resolve_dependencies(const
254 AgenkSnippet* snippet) {
255     if (!snippet) return true;
256     for (size_t i = 0; i < snippet->num_
257         dependencies; ++i) {
258         AgenkDependency* dep = snippet->
259             dependencies[i];
260         if (check_artifacts_exist(dep))
261             continue;
262         bool success = false;
263         switch (dep->type) {
264             case DEP_SOURCE: success =
265                 handle_source_dependency(
266                     dep); break;
267             case DEP_PIP: success =
268                 handle_pip_dependency(
269                     dep); break;
270             case DEP_NPM: success =
271                 handle_npm_dependency(
272                     dep); break;
273             case DEP_SYSTEM: success =
274                 handle_system_dependency(
275                     dep); break;
276             default: EXEC_LOG_ERROR("
277                 Unsupported dependency
278                 type: %d", dep->type);
279         }
280         if (!success) { EXEC_LOG_ERROR("
281             Failed to resolve dependency
282                 for \"%s\".", dep->uri);
283         return false; }
284     }
285     return true;
286 }

```

Listing 39: The hardened, cross-platform dependency manager ('dependency\_manager.c').

**The Language Runtimes (runtime\_\*.c):** These are the "language experts" to which the engine delegates the final execution. The modular design allows us to easily add new languages in the future without altering the core engine.

**Insight: The Tale of Two Linkers.** The most profound challenge overcome was in the C JIT runtime. An early version failed with the error `tcc: error: undefined symbol 'core_create_group'`. This revealed a critical concept: the difference between the **build-time linker** (`ld`) and the **run-time linker** (the logic inside TCC). The build-time linker creates the final executable but, by default, hides its internal function names for efficiency. TCC, running later, could not find these hidden symbols. The definitive solution, shown in Listing 35, was to add the `-Wl,--export-dynamic` linker flag for the test executable. This flag instructs the build-time linker to create a public "table of contents" for its internal functions, making them visible to the TCC runtime linker, which can then successfully connect the JIT-compiled code to the

main program's Core API. The C runtime itself must still be configured with the paths to its own headers and the agent's libraries, but the dynamic export flag is what makes the final link possible.

```

1  /**
2   * @file runtime_c.c
3   * @brief Implements the execution logic
4   * for C-language Snippets.
5   *
6   * Uses the TinyCC (TCC) library to
7   * perform in-memory compilation and
8   * execution of C source code provided
9   * in a Snippet. This module is
10  * responsible
11  * for correctly configuring the JIT
12  * compiler with all necessary include
13  * paths,
14  * library paths, and, most importantly,
15  * the memory addresses of live
16  * symbols
17  * from the host application's core API.
18  *
19  * The architectural contract is that
20  * any C Snippet MUST expose an entry
21  * point
22  * function with the following signature
23  * :
24  * ExecutionResult* <entry_point>(
25  *     const ExecutionContext* );
26  *
27  * The Snippet is responsible for
28  * allocating and returning a valid
29  * ExecutionResult struct.
30  *
31  * This runtime safely calls that
32  * function and takes ownership of the
33  * returned struct.
34  *
35  * @version 1.0.0
36  * @author Ankush Yadav, Ankit Yadav,
37  * AuctaSapience
38  */
39
40
41 #include <stdio.h>
42 #include <stdlib.h>
43 #include <libtcc.h>
44 #include "../private/execution_private.h"
45
46 #include "agenk_core.h"
47 #include "agenk_snippet.h"
48
49 // Define AGENK_TCC_PATH via CMake
50 #ifndef AGENK_TCC_PATH
51 #error "AGENK_TCC_PATH must be defined by CMake"
52 #endif
53
54 // Function pointer type for the snippet
55 // entry point
56 typedef ExecutionResult* (*process_func)
57     (const ExecutionContext* );
58
59 ExecutionResult* execute_c_snippet(const
60     AgenkCode* code, const
61     ExecutionContext* context) {
62     ExecutionResult* result = (
63         ExecutionResult*)calloc(1,
64             sizeof(ExecutionResult));
65     TCCState *s = NULL;
66
67     int ret = -1;
68     process_func process = NULL;
69     const char* source = code->variants
70         ->source_code;
71
72     s = tcc_new();
73     if (!s) {
74         EXEC_LOG_ERROR("Could not create
75             TCC state.");
76         result->exit_code = -1;
77         return result;
78     }
79
80     // THE FIX: Point TCC to the build
81     // output directory for its
82     // libraries
83     tcc_set_lib_path(s, AGENK_TCC_LIB_
84         PATH);
85
86     // Add include paths for project
87     // headers
88     tcc_add_sysinclude_path(s, AGENK_TCC_
89         PATH "/include");
90     tcc_add_include_path(s, AGENK_CORE_
91         INCLUDE_DIR);
92     tcc_add_include_path(s, AGENK_
93         SNIPPET_INCLUDE_DIR);
94
95     tcc_set_output_type(s, TCC_OUTPUT_
96         MEMORY);
97
98     // This isn't strictly necessary
99     // with the linker flag, but is
100    // good practice
101    tcc_add_symbol(s, "core_create_group",
102        (const void*)core_create_
103        group);
104
105    // Compile the source code
106    ret = tcc_compile_string(s, source);
107    if (ret < 0) {
108        EXEC_LOG_ERROR("TCC compilation
109            failed for C snippet.");
110        result->exit_code = -1;
111        tcc_delete(s);
112        return result;
113    }
114
115    // Relocate symbols and retrieve
116    // entry point
117    ret = tcc_relocate(s, TCC_RELOCATE_
118        AUTO);
119    if (ret < 0) {
120        EXEC_LOG_ERROR("TCC relocation
121            failed.");
122        result->exit_code = -1;
123        tcc_delete(s);
124        return result;
125    }
126
127    process = (process_func)tcc_get_
128        symbol(s, code->entry_point);
129
130    if (!process) {
131        EXEC_LOG_ERROR("TCC entry point
132            '%s' not found.", code->
133                entry_point);
134        result->exit_code = -1;
135    } else {
136        // Execute the function
137    }
138
139

```

```

89     ExecutionResult* snippet_result
90         = process(context);
91     if (snippet_result) {
92         *result = *snippet_result;
93         // Copy data
94         free(snippet_result);
95         // Free the structure
96         // allocated by the snippet
97     }
98     // Note: result->return_value
99     // ownership is transferred
100
101    tcc_delete(s);
102    return result;
103}
104
105// Stubs for C runtime init/shutdown (
106// often empty for TCC)
107bool c_runtime_init() { return true; }
108void c_runtime_shutdown() {}

```

Listing 40: The final, robust C JIT runtime using TCC ('runtime.c').

```

1 /**
2  * @file runtime_python.c
3  * @brief Implements the execution logic
4  *        for Python-language Snippets.
5  *
6  * This version uses a robust, canonical
7  * pattern for CPython extension
8  * development,
9  * including a goto-based cleanup and
10 * direct exception formatting to
11 * ensure
12 * all tracebacks are captured reliably.
13 *
14 * @version 1.0.0
15 * @author Ankush Yadav, Ankit Yadav,
16 *        AuctaSapience
17 */
18
19 #define PY_SSIZE_T_CLEAN
20 #include <Python.h>
21 #include "../private/execution_private.h"
22 #include "agenk_core.h"
23
24 PyMODINIT_FUNC PyInit_agenk_core(void);
25
26 bool python_runtime_init() {
27     if (Py_IsInitialized()) return true;
28
29     if (PyImport_AppendInittab("agenk_
30         core", &PyInit_agenk_core) ==
31         -1) {
32         EXEC_LOG_ERROR("Failed to add 'agenk_
33             core' to Python built-
34             in table.");
35         return false;
36     }
37
38     Py_Initialize();
39     if (!Py_IsInitialized()) {
40         return false;
41     }
42
43 #ifdef PYTHON_SITE_PACKAGES

```

```

34         // Append the system's site-packages
35         // directory to the interpreter's
36         // path.
37         // This is crucial for finding
38         // modules like NumPy at runtime.
39         PyObject *sys_path = PySys_GetObject
40             ("path");
41         if (sys_path && PyList_Check(sys_
42             path)) {
43             PyObject *path_str = PyUnicode_
44                 FromString(PYTHON_SITE_
45                     PACKAGES);
46             if (PyList_Append(sys_path,
47                 path_str) < 0) {
48                 PyErr_Print();
49                 EXEC_LOG_ERROR("Failed to append site-
50                     packages path to sys.path.");
51             } else {
52                 EXEC_LOG("Appended '%s' to Python sys.path",
53                     PYTHON_SITE_
54                     PACKAGES);
55             Py_DECREF(path_str);
56         }
57     }
58
59     #endif
60
61     return true;
62 }
63
64 void python_runtime_shutdown() {
65     if (Py_IsInitialized()) Py_
66         FinalizeEx();
67 }
68
69 ExecutionResult* execute_python_snippet(
70     const AgenkCode* code, const
71     ExecutionContext* context) {
72     ExecutionResult* result = calloc(1,
73         sizeof(ExecutionResult));
74     if (!result) return NULL;
75
76     PyObject *old_stdout = NULL, *old_
77         stderr = NULL, *io_module = NULL
78         ;
79     PyObject *py_stdout = NULL, *py_
80         stderr = NULL, *local_dict =
81         NULL;
82     PyObject *exec_result = NULL, *func
83         = NULL, *py_context = NULL;
84     PyObject *pArgs = NULL, *call_result
85         = NULL;
86
87     old_stdout = PySys_GetObject("stdout");
88     old_stderr = PySys_GetObject("stderr");
89
90     io_module = PyImport_ImportModule("io");
91     if (!io_module) { result->exit_code
92         = -1; goto cleanup; }
93     py_stdout = PyObject_CallMethod(io_
94         module, "StringIO", NULL);
95     py_stderr = PyObject_CallMethod(io_
96         module, "StringIO", NULL);

```

```

74     if (!py_stdout || !py_stderr) {
75         result->exit_code = -1; goto
76         cleanup;
77     }
78     PySys_SetObject("stdout", py_stdout)
79     ;
80     PySys_SetObject("stderr", py_stderr)
81     ;
82
83     PyObject *main_module = PyImport_ImportModule("__main__");
84     if (!main_module) { result->exit
85         _code = -1; goto cleanup; }
86     PyObject *main_dict = PyModule_GetDict(main_module);
87
88     // By using the same dictionary
89     // for globals and locals, we
90     // faithfully
91     // replicate how Python executes
92     // a module's code. Both '
93     // import' and 'def'
94     // will populate the same
95     // namespace.
96     exec_result = PyRun_String(code
97         ->variants->source_code, Py_
98         file_input, main_dict, main_
99         dict);
100    Py_DECREF(main_module);
101
102    if (!exec_result) { // Catches
103        SyntaxError
104        result->exit_code = -1;
105    } else {
106        Py_DECREF(exec_result);
107        exec_result = NULL;
108        func = PyDict_GetItemString(main_
109            dict, code->entry_point);
110        if (func && PyCallable_Check(
111            func)) {
112            py_context = PyDict_New();
113            const char* input_uri = (
114                context && context->
115                input_uri) ? context->
116                input_uri : "";
117            PyDict_SetItemString(py_
118                context, "input_uri",
119                PyUnicode_FromString(
120                    input_uri));
121            pArgs = PyTuple_Pack(1, py_
122                context);
123
124            call_result = PyObject_
125                CallObject(func, pArgs);
126            if (!call_result) { //
127                // Catches ImportError, etc
128                .
129                result->exit_code = -1;
130            } else {
131                result->exit_code = 0;
132                result->return_value =
133                    (void*)call_result;
134                // Ownership is
135                // transferred
136            }
137        } else {
138            result->exit_code = -1;
139            PyErr_SetString(PyExc_
140                NameError, "Entry point
141                not found or not
142                callable.");
143        }
144    }
145
146    if (PyErr_Occurred()) {
147        PyErr_Print(); // This prints
148        // the full traceback to sys.
149        // stderr
150    }
151
152    // Capture stdout
153    if (py_stdout) {
154        PyObject* stdout_val = PyObject_
155            CallMethod(py_stdout, "getvalue", NULL);
156        if (stdout_val) {
157            result->stdout_buffer = safe_
158                strdup(PyUnicode_AsUTF8(
159                    stdout_val));
160            Py_DECREF(stdout_val);
161        }
162    }
163
164    // Capture stderr (which now
165    // contains the traceback)
166    if (py_stderr) {
167        PyObject* stderr_val = PyObject_
168            CallMethod(py_stderr, "getvalue", NULL);
169        if (stderr_val) {
170            result->stderr_buffer = safe_
171                strdup(PyUnicode_AsUTF8(
172                    stderr_val));
173            Py_DECREF(stderr_val);
174        }
175
176    // Restore and clean up
177    if (old_stdout) PySys_SetObject("stdout", old_stdout);
178    if (old_stderr) PySys_SetObject("stderr", old_stderr);
179    Py_XDECREF(io_module); Py_XDECREF(py_
180        stdout); Py_XDECREF(py_stderr);
181    Py_XDECREF(local_dict); Py_XDECREF(
182        exec_result); Py_XDECREF(py_
183        context);
184    Py_XDECREF(pArgs);
185    if (result->exit_code != 0) Py_
186        XDECREF(call_result);
187
188    return result;
189}

```

Listing 41: The sandboxed Python runtime with I/O and exception capture ('runtime<sub>python.c</sub>').

```

1 /**
2  * @file runtime_shell.c
3  * @brief Implements the execution logic
4  *        for shell command Snippets.
5  * @version 1.0.0

```

```
6 * @author Ankush Yadav, Ankit Yadav,
7     AuctaSapience
8 */
9 #include "../private/execution_private.h"
10 #include <stdlib.h>
11
12 ExecutionResult* execute_shell_snippet(
13     const char* command_string, const
14     ExecutionContext* context) {
15     (void)context; // Context not used
16         in this simple version
17
18     ExecutionResult* result = (
19         ExecutionResult*)calloc(1,
20             sizeof(ExecutionResult));
21     if (!result) return NULL;
22
23     result->exit_code = run_command(
24         command_string, ".", &result->
25             stdout_buffer, &result->stderr-
26                 buffer);
27
28     return result;
29 }
30 }
```

Listing 42: The sandboxed Shell runtime ('`runtime_shell.c`').

### 7.3.3 Validation and the Debugging Journey

A system of this complexity cannot be trusted until it is rigorously tested. The `test_execution.c` suite provides a full, end-to-end validation of the entire substrate, including its failure mode handling. The journey to a stable, passing test suite was a profound lesson in the discipline required for C systems programming, systematically resolving build errors, linker failures, and subtle runtime bugs to arrive at the final, robust implementation. The successful run of this comprehensive test suite is the ultimate validation of the entire architectural endeavor.

```
1  /**
2   * @file test_execution.c
3   * @brief Comprehensive, hardened test
4   *        suite for the AGENK Execution
5   *        Substrate.
6   *
7   * This is the definitive validation
8   *        suite for the agent's "CPU". It
9   *        confirms
10  *        not only the "happy path" for all
11  *        language runtimes but also
12  *        validates the
13  *        substrate's resilience and error-
14  *        handling capabilities when
15  *        presented with
16  *        deliberately broken or invalid
17  *        Snippets.
18  *
19  * It validates:
```

```
11 * - Correct memory management of the
12 * Snippet data structures.
13 * - The Shell runtime's ability to
14 * execute commands and capture output
15 *
16 * - The architectural purity of the C
17 * JIT runtime and its access to
18 * context.
19 * - The Python runtime's ability to
20 * import and use the `agenk_core`'
21 * native bridge.
22 * - The engine's graceful handling of
23 * dependency failures, syntax errors,
24 * and
25 * runtime exceptions.
26 */
27 // --- FIX #1: Standard library and
28 // external headers FIRST ---
29 #include <stdio.h>
30 #include <stdlib.h>
31 #include <string.h>
32
33 #ifdef AGENK_PYTHON_SUPPORT
34 #define PY_SIZE_T_CLEAN
35 #include <Python.h>
36 #endif
37
38 // --- FIX #2: Your project's headers
39 // LAST ---
40 #include "agenk_execution.h"
41 #include "agenk_core.h"
42 #include "object.h" // Needed for CLASS_
43 // INIT()
44
45 // *** BEGIN WORKAROUND for incomplete
46 // object.h ***
47
48 // These declarations are missing from
49 // the class_framework's public header.
50 // Adding them here resolves the "
51 // implicit declaration" warnings and
52 // allows
53 // the linker to correctly find the
54 // functions in the library.
55 void CLASS_INIT(void);
56 void CLASS_SHUTDOWN(void);
57 // *** END WORKAROUND ***
```

```
44 // --- Testing framework macros ---
45 static int g_tests_run = 0;
46 static int g_tests_passed = 0;
47 #define COLOR_GREEN "\x1B[32m"
48 #define COLOR_RED " \x1B[31m"
49 #define COLOR_RESET "\x1B[0m"
50 #define TEST_SUITE_START(name) printf("\n--- Running Test Suite: %s ---\n",
51 name)
52 #define TEST_CASE(name) printf(" [TEST] %s\n", name)
53 #define ASSERT(condition)
54 \
55 do {
56 \
57     \
58     g_tests_run++;
```

```

56         \
57     if (condition) {
58         \
59     g_tests_passed++;
60         \
61     printf(COLOR_GREEN "[PASS]\n" COLOR_RESET, \
62         FILE__, LINE__); \
63 } else { \
64     \
65     fprintf(stderr, COLOR_RED "[\n" \
66         "FAIL] %s:%d: Assertion\n" \
67         failed: %s\n" \
68         COLOR_RESET, FILE__ \
69         , LINE__, # \
70         condition); \
71     \
72 } while (0)
73 // --- Test Suites ---
74 // ... (No changes to any of the test_
75 // suite functions) ...
76 void test_suite_snippet_lifecycle() {
77     TEST_SUITE_START("Snippet_Lifecycle_"
78         "&Memory_Management");
79     AgenkSnippet* s = snippet_create("A"
80         "test_snippet_for_file_operations"
81         ".");
82     ASSERT(s != NULL);
83     AgenkCode* exec_code = code_create(
84         LANG_SHELL, NULL);
85     code_add_variant(exec_code, OS_LINUX
86         , "ls -l");
87     snippet_set_execution_code(s, exec_
88         code);
89     AgenkDependency* dep = dependency_
90         create(DEP_SOURCE, "http://
91         example.com/dep.git");
92     dependency_add_build_command(dep, "
93         make_all");
94     snippet_add_dependency(s, dep);
95     ASSERT(s->dependencies[0]->num_build_
96         _commands == 1);
97     TEST_CASE("Deep_free_of_a_complex_
98         Snippet");
99     snippet_free(s);
100    ASSERT(1);
101 }

102 void test_suite_shell_execution() {
103     TEST_SUITE_START("Shell_Execution_"
104         "Runtime");
105     AgenkSnippet* s = snippet_create(""
106         "Echo_test");
107     AgenkCode* code = code_create(LANG_
108         _SHELL, NULL);
109     code_add_variant(code, OS_ANY, "echo"
110         "\'Hello, Shell!\'");
111     snippet_set_execution_code(s, code);
112     ExecutionContext ctx = { .input_uri
113         = NULL, .api_request = NULL };
114     ExecutionResult* result = engine_
115         execute(s, &ctx);
116     ASSERT(result != NULL);
117     if (result) {
118         ASSERT(result->exit_code == 0);
119         ASSERT(result->stdout_buffer !=
120             NULL);
121         ASSERT(strncmp(result->stdout_
122             buffer, "Hello, Shell!", 13)
123             == 0);
124         execution_result_free(result);
125     }
126     snippet_free(s);
127 }

128 void test_suite_c_execution() {
129     TEST_SUITE_START("C_JIT_Execution_"
130         "Runtime");
131     const char* c_source =
132         "#include<agenk_core.h>\n"
133         "#include<stdio.h>\n"
134         "#include<stdlib.h>\n"
135         "#include<string.h>\n"
136         "ExecutionResult* process(const"
137         "ExecutionContext* context)"
138         "\n"
139         "{\n"
140         "    ExecutionResult* result = ("
141             ExecutionResult*)calloc(1, "
142                 sizeof(ExecutionResult));\n"
143         "    unsigned long id = core_
144             create_group(\"Test from C_"
145             "Snippet\");\n"
146         "    uchar* result_str = (char*)"
147             malloc(128);\n"
148         "    sprintf(result_str, \"C_"
149             "Snippet created group with"
150             " ID: %lu\", id);\n"
151         "    result->exit_code = 0;\n"
152         "    result->return_value = ("
153             void*)result_str;\n"
154         "    return result;\n"
155     }\n";
156     AgenkSnippet* s = snippet_create("C_"
157         "JIT_core_API_utest");
158     AgenkCode* code = code_create(LANG_C
159         , "process");
160     code_add_variant(code, OS_ANY, c_
161         source);
162     snippet_set_execution_code(s, code);
163     ExecutionContext ctx = { .input_uri
164         = NULL, .api_request = NULL };
165     ExecutionResult* result = engine_
166         execute(s, &ctx);
167     ASSERT(result != NULL);
168     if (result) {
169         ASSERT(result->exit_code == 0);
170         ASSERT(result->return_value !=
171             NULL);
172         if (result->return_value) {
173             printf("[INFO] C_Snippet_"
174                 "returned: \"%s\"\n", (
175                 char*)result->return_
176                     value);
177             free(result->return_value);
178         }
179         execution_result_free(result);
180     }
181     snippet_free(s);
182 }

183 #ifdef AGENK_PYTHON_SUPPORT

```

```

138 void test_suite_python_execution() {
139     TEST_SUITE_START("Python_Execution_
140         Runtime_&_Native_Bridge");
141     AgenkSnippet* s = snippet_create("_
142         Native_bridge_utest");
143     AgenkCode* code = code_create(LANG_
144         PYTHON, "run_test");
145     const char* python_source =
146         "import agenk_core\n\n"
147         "def run_test(context):\n"
148             "    print('Python: Calling'
149                 "\n"
150                 "        agenk_core.create_group()')\n"
151                 "\n"
152                 "group_id = agenk_core."
153                 "create_group('Test from'
154                 "\n"
155                 "    Python')\n"
156                 "\n"
157                 "print(f'Python: Received'
158                 "\n"
159                 "        group_ID: {group_id}')\n"
160                 "\n"
161                 "return group_id\n";
162     code_add_variant(code, OS_ANY,
163         python_source);
164     snippet_set_execution_code(s, code);
165     ExecutionContext ctx = { .input_uri
166         = NULL, .api_request = NULL };
167     ExecutionResult* result = engine_
168         execute(s, &ctx);
169     ASSERT(result != NULL);
170     if(result) {
171         ASSERT(result->exit_code == 0);
172         ASSERT(result->return_value !=
173             NULL);
174         if (result->return_value) Py_
175             XDECREF((PyObject*)result->
176                 return_value);
177         execution_result_free(result);
178     }
179     snippet_free(s);
180 }
181 #endif
182
183 void test_suite_failure_modes() {
184     TEST_SUITE_START("Failure_Mode_
185         Validation");
186     ExecutionContext ctx = { .input_uri
187         = NULL, .api_request = NULL };
188     TEST_CASE("Executing_Snippet_with_
189         invalid_dependency_URI");
190     AgenkSnippet* s_bad_git = snippet_
191         create("Bad_Git_URL");
192     AgenkDependency* dep_bad_git =
193         dependency_create(DEP_SOURCE, "
194             https://github.com/nonexistent/
195             repo12345.git");
196     snippet_add_dependency(s_bad_git,
197         dep_bad_git);
198     AgenkCode* code_dummy = code_create(
199         LANG_SHELL, NULL);
200     code_add_variant(code_dummy, OS_ANY,
201         "echo 'should not run'");
202     snippet_set_execution_code(s_bad_git
203         , code_dummy);
204     ExecutionResult* result_bad_git =
205         engine_execute(s_bad_git, &ctx);
206     ASSERT(result_bad_git != NULL &&
207         result_bad_git->exit_code != 0);
208     execution_result_free(result_bad_git
209         );
210     snippet_free(s_bad_git);
211     #ifdef AGENK_PYTHON_SUPPORT
212     TEST_CASE("Executing_Python_Snippet_
213         ");
214         withSyntaxError");
215         AgenkSnippet* s_py_syntax = snippet_
216             create("Python_Syntax_Error");
217         AgenkCode* code_py_syntax = code_
218             create(LANG_PYTHON, "run_test");
219         code_add_variant(code_py_syntax, OS_
220             ANY, "def run_test(context):\n"
221                 "\n"
222                 "print('hello')\n");
223         snippet_set_execution_code(s_py_
224             syntax, code_py_syntax);
225         ExecutionResult* result_py_syntax =
226             engine_execute(s_py_syntax, &ctx
227                 );
228         ASSERT(result_py_syntax != NULL &&
229             result_py_syntax->exit_code !=
230                 0);
231         ASSERT(result_py_syntax->stderr_
232             buffer != NULL && strstr(result_
233                 py_syntax->stderr_buffer, "
234                     SyntaxError"));
235         execution_result_free(result_py_
236             syntax);
237         snippet_free(s_py_syntax);
238     #endif
239
240 // --- Test Runner ---
241 int main() {
242     printf("=====\\n");
243     printf("    Running AGENK_Execution_
244         Substrate Tests\\n");
245     printf("=====+\\n");
246
247     // *** FIX: Revert to the correct
248     // function names from the class_
249     // framework library ***
250     CLASS_INIT();
251     ASSERT(engine_init("./agenk_cache"))
252         ;
253
254     test_suite_snippet_lifecycle();
255     test_suite_shell_execution();
256     test_suite_c_execution();
257     #ifdef AGENK_PYTHON_SUPPORT
258         test_suite_python_execution();
259     #else
260         printf("\n--- Skipping Python_
261             Test Suite: Python support
262             disabled---\\n");
263     #endif
264     test_suite_failure_modes();
265
266     engine_shutdown();
267     // *** FIX: Revert to the correct
268     // function names from the class_
269     // framework library ***
270     CLASS_SHUTDOWN();
271
272     printf("\n=====\\n");
273     printf("TEST_SUMMARY\\n");
274     printf("=====\\n");
275     if (g_tests_passed == g_tests_run) {
276         printf(COLOR_GREEN "SUCCESS: All
277             %d tests passed.\\n" COLOR_
278                 RESET, g_tests_run);
279     } else {
280         fprintf(stderr, COLOR_RED "
281             FAILURE: %d out of %d tests\\n"
282                 );
283     }
284 }
```

```

225     failed.\n" COLOR_RESET,
226     g_tests_run - g_tests_
227     passed, g_tests_run)
228     ;
229 }

```

Listing 43: The comprehensive test suite, including failure mode validation ('`test_execution.c`').

## 7.4 Architectural Horizons: The Orchestrator and Composable Skills

The completion of the Execution Substrate does not just provide a tool for running scripts; it enables a new paradigm for agent behavior. With a stable library of skills provided by the Snippet Registry and a robust engine to run them, the final piece is a higher-level cognitive core that can act as an **Orchestrator**.

This future Orchestrator module will be responsible for reasoning, planning, and achieving complex goals by composing the simple, reusable skills it finds in the registry. For example, to upload a Base64-encoded image to a REST API, the Orchestrator would:

1. Query the `snippet_registry` to fetch three distinct tools: a `FileReaderSnippet`, a `Base64EncoderSnippet`, and an `HttpPostSnippet`.
2. Invoke the `execution` engine with the `FileReaderSnippet` to get the raw image data into an `AgenkTensor`.
3. Take the resulting tensor and use it to construct a new `ExecutionContext` for the `Base64EncoderSnippet`, then invoke the `execution` engine again.
4. Take the final encoded tensor and use it as the body in an `ApiRequestContext` for the `HttpPostSnippet`, invoking the `execution` engine a final time.

This pattern of **composable skills** is the essence of scalable intelligence. The Snippets Substrate provides the "what" (the library of knowledge), and the Execution Substrate provides the "how" (the universal machine for action). Together, they form the stable foundation upon which true, dynamic reasoning and planning can be built.

## 8 The Core Substrate: A Stable Kernel for an Evolving Agent

The preceding substrates provide the agent with its foundational capabilities: the `Snippets` substrate defines a "genome" for skills, while the `Execution` substrate provides a "CPU" to run them. However, for a system designed to dynamically execute an ever-changing library of untrusted code, a third, even more fundamental layer is required: a secure, stable, and immutable kernel. This is the exclusive and paramount role of the **AGENK Core Substrate**.

This module is not a system of action or data; it is a system of **contract**. It serves as the single, minimalist, and permanent bridge between the trusted, statically-compiled agent and the dynamic, untrusted world of Snippets. It is the agent's operating system kernel, exposing a small set of "system calls" that allow sandboxed guest code to interact with the underlying hardware of the agent's mind in a safe and controlled manner.

### 8.1 Architectural Philosophy: The Kernel API

The design of the Core Substrate is guided by three non-negotiable principles, drawn directly from the design of modern, secure operating system kernels.

1. **Abstraction:** The Core API must completely hide the complexity of the underlying substrates. A Python script running inside a Snippet has no knowledge of the single-block memory allocation strategy of the Tensor Substrate or the fine-grained locking of the Object Framework's registry. It only needs a simple, clean function to call, like `core_tensor_create()` or `core_create_group()`. This powerful abstraction allows us to completely rearchitect the underlying substrates—for instance, replacing the LMDB database with a different storage engine—without breaking a single Snippet, as long as the Core API contract is maintained.
2. **Stability and Immutability:** The function signatures defined in `agenk_core.h` form a permanent contract with all past, present, and future Snippets. Like the POSIX system call interface, this API is designed to be immutable. We can add *new* functions to expand the agent's capabilities, but we must never

change or remove existing ones. This guarantees that the agent's learned skills have long-term viability. A skill acquired today will continue to function flawlessly years from now, even as the agent's internal machinery evolves dramatically around it.

**3. Security (The Firewall):** This is its most critical role. The Core API is the security boundary—the firewall—between the untrusted guest code and the trusted, low-level foundations of the agent. It is a set of carefully audited "gates" through which all requests must pass.

- **No Pointer Passthrough:** Guest code *never* receives a raw C pointer to an internal data structure. Instead, it is given an opaque handle (like the `PyCapsule` we implemented in the Python bridge). To perform an operation, it must pass this handle *back* to a Core API function. The Core function, running in the trusted kernel space, safely unwraps the handle and performs the operation, preventing the guest from ever directly accessing or corrupting core memory.
- **Paranoid Validation:** Every Core API function is designed to be paranoid. It assumes its inputs may be malformed or malicious and is responsible for validating them before proceeding.

## 8.2 Architectural Design and Implementation

The implementation is a direct reflection of these principles, resulting in a small, clean, and highly secure module.

### 8.2.1 Project Structure and The Dependency Hub

The `agenk_core` module sits at the center of the agent's dependency graph, but it is a one-way hub, enforcing the layered architecture.

```
src/
└── core/
    ├── bridge/
    │   └── native_bridge_python.c
    └── include/
        └── agenk_core.h
    core.c
```

The dependency graph is strict: higher-level modules like `Execution` depend on

`agenk_core`, and `agenk_core` in turn depends on the low-level substrates like `agenk_tensor` and `class_framework`. This ensures that the `Execution` engine can never bypass the Core API's security and validation layer to access the low-level substrates directly.

```
1 # --- LIBRARY 5: agenk_core (The "Kernel"
2 #   / Stable Native API) ---
3 set(CORE_SOURCES src/core/core.c)
4 if(AGENK_ENABLE_PYTHON)
5     list(APPEND CORE_SOURCES src/core/
6         bridge/native_bridge_python.c)
7 endif()
8 add_library(agenk_core STATIC ${CORE_
9     SOURCES})
10
11 target_include_directories(agenk_core
12     PUBLIC
13     ${CMAKE_CURRENT_SOURCE_DIR}/src/core
14     /include
15 )
16 # The PUBLIC keyword is critical here.
17 # It ensures that any module
18 # that links against agenk_core (like
19 # agenk_execution) also gets the
# include paths for the foundational
# data structures.
target_link_libraries(agenk_core PUBLIC
    agenk_snippet agenk_tensor class_
    framework
)
20 if(AGENK_ENABLE_PYTHON)
21     target_link_libraries(agenk_core
22         PRIVATE Python3::Python Python3
23         ::NumPy)
24 endif()
```

Listing 44: The 'agenk\_core' library definition, showing its central role in the dependency graph.

### 8.2.2 The Public Contract (`agenk_core.h`)

The public API, presented in Listing 45, is the formal, immutable contract for all guest code. It is intentionally minimal, exposing only the essential, high-level capabilities required for Snippets to function.

```
1 /**
2  * @file agenk_core.h
3  * @brief Public API for the AGENK Core
4  * Substrate (the "Kernel").
5  *
6  * This header defines the stable,
7  * secure, and universal set of C
8  * functions
9  * that are exposed to all Snippets
10 * running within the Execution
11 * Substrate.
12 * It is the "System Call" interface of
13 * the AGENK agent, providing
14 * sandboxed
15 * guest code with controlled access to
16 * the agent's core capabilities, such
17 * as
```

```

9  * tensor manipulation and object
10 * lifecycle management.
11 * This API is designed to be immutable.
12 * Functions added here become a
13 * permanent
14 * part of the agent's core
15 * functionality.
16 */
17
18 #ifndef AGENK_CORE_H
19 #define AGENK_CORE_H
20
21 #include "agenk_snippet.h" // For
22     AgenkTensor forward declaration
23 #include <stddef.h>
24 #include <stdint.h>
25 #include <stdbool.h>
26
27 // --- API Symbol Exposure for JIT
28     Runtimes ---
29 /**
30 * @brief Maps an API function's string
31 * name to its address in memory.
32 * This is used by JIT runtimes (like
33 * the C runtime) to link against
34 * the host application's live functions
35 *
36 */
37
38 // --- API Self-Documentation Structures
39 /**
40 * @brief Describes a single parameter
41 * for a core API function. */
42
43 typedef struct {
44     const char* name;
45     void* func;
46 } AgenkApiSymbol;
47
48 // --- API Function Documentation Structures
49 /**
50 * @brief Describes a single, stable
51 * function exposed by the AGENK Core.
52 */
53
54 typedef struct {
55     const char* function_name;
56     const char* description;
57     AgenkApiParameterDoc* parameters;
58     size_t num_parameters;
59     const char* return_type;
60     const char* return_description;
61 } AgenkApiFunctionDoc;
62
63 // Tensor Substrate Bindings
64 AgenkTensor* core_tensor_create(const
65     size_t* shape, size_t ndim, int
66     dtype, int ctype);
67 void core_tensor_free(AgenkTensor*
68     tensor);
69 size_t core_tensor_get_ndim(const
70     AgenkTensor* tensor);
71 // ... other tensor accessors would be
72 // added here (get_shape, get_data_ptr,
73 // etc.)
74
75 // --- API Management Functions ---
76 /**
77 * @brief Retrieves the complete list of
78 * core API symbols for JIT linking.
79 * @param count Pointer to an integer
80 * which will be filled with the
81 * number of symbols.
82 * @return A constant pointer to the
83 * internal array of API symbols. The
84 * caller
85 * does NOT own this memory and
86 * must NOT free or modify it.
87 */
88 const AgenkApiSymbol* core_get_api_
89 symbols(int* count);
90
91 /**
92 * @brief Retrieves the complete,
93 * machine-readable documentation for
94 * the Core API.
95 * @param count Pointer to a size_t
96 * which will be filled with the
97 * number of functions.
98 * @return A constant pointer to the
99 * internal array of function
100 * documentation. The caller
101 * does NOT own this memory and
102 * must NOT free it.
103 */
104 const AgenkApiFunctionDoc* core_get_api_
105 documentation(size_t* count);
106
107 #endif // AGENK_CORE_H

```

Listing 45: The public API header for the Core Substrate ('agenkcore.h').

### 8.2.3 The Implementation

The implementation is split into two logical parts: the generic C implementation and the specialized bridge for the Python runtime.

**The Core Implementation (core.c):** This file, shown in Listing 46, contains the implementations of the Core API functions. Each function is a simple, stable "wrapper" or "facade." Its only job is to receive a request from guest code, perform any necessary validation, and then call the corresponding, more complex function in the underlying trusted substrate (e.g.,

`core_create_group` calls `CREATE_GROUP` from the `class_framework`.

**Insight — A Future of Pure Abstraction:** The current implementation, where `core.c` directly includes headers like `object.h`, is a pragmatic and robust starting point. However, the ultimate architectural vision for this module is one of pure abstraction. In a future version, `core.c` would contain only function pointers. The main agent executable, during its startup sequence, would be responsible for "registering" the concrete implementations from the various substrates with the Core module. This would achieve the ultimate level of decoupling, allowing different implementations of the tensor or object substrates to be swapped out at runtime without ever recompiling the core agent or its library of skills.

```

1  /**
2   * @file core.c
3   * @brief Implements the stable, public
4   *        API for the AGENK Core Substrate.
5   *
6   * This file provides the concrete
7   * implementations for the functions
8   * declared
9   * in `agenk_core.h`. It also defines
10  * the tables that expose these
11  * functions
12  * to other parts of the system, such as
13  * JIT runtimes, through a stable
14  * symbol table.
15  *
16  * @version 1.2.0
17  * @author Ankush Yadav, Ankit Yadav,
18  *        AuctaSapience
19  */
20
21 #include "agenk_core.h"
22 #include "object.h" // *** FIX #1:
23           Include the header for the class
24           framework API ***
25
26 // --- Core API Function Implementations
27 // ---
28
29 unsigned long core_create_group(const
30         char* description) {
31     // This function simply acts as a
32     // stable wrapper around the
33     // underlying object framework's
34     // implementation.
35     // *** FIX #2: Use the correct
36     // function names (without CLASS_
37     // prefix) ***
38     return CREATE_GROUP(description);
39 }
40
41 void core_destroy_by_group(unsigned long
42         group_id) {
43     // Wrapper for the object framework'
44     // s implementation.
45     // *** FIX #2: Use the correct
46     // function names (without CLASS_
```

```

47         prefix) ***
48         DESTROY_BY_GROUP(group_id);
49     }
50
51 // NOTE: Implementations for tensor
52 // functions would go here.
53 // For now, they are just stubs as
54 // defined in the header.
55 AgenkTensor* core_tensor_create(const
56         size_t* shape, size_t ndim, int
57         dtype, int ctype) {
58     return NULL;
59 }
60
61 void core_tensor_free(AgenkTensor*
62         tensor) {
63     // To be implemented
64 }
65
66 size_t core_tensor_get_ndim(const
67         AgenkTensor* tensor) {
68     return 0;
69 }
70
71 // --- API Management Function
72 // Implementations ---
73
74 /**
75  * @brief The internal table of all
76  *        functions exposed by the Core API.
77  *
78  * This is the critical structure that
79  * maps the string name of a function
80  * to its actual address in memory. It
81  * is used by `core_get_api_symbols`
82  * to provide the JIT C runtime with the
83  * information it needs to link
84  * against the core API.
85  *
86  * To add a new function to the JIT-
87  * accessible API, you must add it
88  * here.
89  */
90 static const AgenkApiSymbol G_CORE_API_
91 SYMBOLS[] = {
92     {"core_create_group", (void*)core_
93         create_group},
94     {"core_destroy_by_group", (void*)}
95         core_destroy_by_group},
96     {"core_tensor_create", (void*)core_
97         tensor_create},
98     {"core_tensor_free", (void*)core_
99         tensor_free},
100    {"core_tensor_get_ndim", (void*)core_
101        _tensor_get_ndim}
102 };
103
104 /**
105  * @brief Retrieves the complete list of
106  *        core API symbols for JIT linking.
107  */
108 const AgenkApiSymbol* core_get_api_
109 symbols(int* count) {
110     *count = sizeof(G_CORE_API_SYMBOLS)
111     / sizeof(G_CORE_API_SYMBOLS[0]);
112     return G_CORE_API_SYMBOLS;
113 }
114
115 /**
116  * @brief Retrieves the complete,
117  *        machine-readable documentation for
```

```

75     the Core API.
76
77 const AgenkApiFunctionDoc* core_get_api_
78     documentation(size_t* count) {
79     if(count) *count = 0;
80     return NULL;
81 }

```

Listing 46: The Core API implementation, acting as a stable facade ('core.c').

### The Python Native Bridge

**(native\_bridge\_python.c):** This file is logically part of the Core Substrate. It is the specialized "embassy" for the Python world. It takes the abstract Core API contract and makes it available idiomatically and safely to the Python runtime.

**Insight — Safe Memory Management Across Languages:** The most dangerous interface in a polyglot system is memory management. How can Python safely manage the lifecycle of a C pointer like an `AgenkTensor*`? The solution is the Python C-API's `PyCapsule`. We wrap our raw C pointers in a `PyCapsule` object, which is an opaque handle that is safe to pass around in Python. Crucially, we attach a custom C destructor function to the capsule. When the Python garbage collector determines the capsule object is no longer in use, it automatically calls our C destructor, which in turn calls the correct, trusted Core API function (e.g., '`core_tensor_free`'). This provides a robust, automated mechanism for preventing C-level memory leaks from Python code.

```

1 /**
2  * @file native_bridge_python.c
3  * @brief Implements the Python C-API
4  * module to expose the AGENK Core API
5  *
6  * This module is the definitive, secure
7  * bridge between the sandboxed
8  * Python
9  * environment and the agent's trusted C
10 * core. It is compiled as part of
11 * the
12 * `agenk_core` library and is loaded as
13 * a built-in module by the Python
14 * runtime in the Execution Substrate.
15 *
16 * It follows a strict wrapper pattern:
17 * 1. It only includes the public `agenk_
18 *    core.h` API, ensuring it
19 *    cannot
20 *    access internal agent functions.
21 * 2. Each exposed Python function (e.g
22 *    ., `agenk_core.tensor_create`) is a
23 *    wrapper that calls the
24 *    corresponding stable C function (
25 *    core_tensor_create`).

```

```

15 * 3. It uses PyCapsule objects with
16 *    custom destructors to safely manage
17 *    the
18 *    lifecycle of C pointers (like
19 *    AgenkTensor*) from Python,
20 *    preventing memory leaks.
21 *
22 * @version 2.0.0 (Definitive,
23 *           architecturally pure implementation
24 *           )
25 * @author Ankush Yadav, Ankit Yadav,
26 *         AuctaSapience
27 */
28
29 #define PY_SSIZE_T_CLEAN
30 #include <Python.h>
31 #define NPY_NO_DEPRECATED_API NPY_1_7_
32     API_VERSION
33 #include <numpy/arrayobject.h>
34
35 #include "../include/agenk_core.h"
36
37 // --- Global Constant for PyCapsule
38 // Names ---
39 // This acts as a type-safe identifier
40 // for our opaque C pointers.
41 #define AGENK_TENSOR_CAPSULE_NAME "agenk_
42     _tensor_handle"
43
44 // --- PyCapsule Destructor ---
45
46 /**
47 * @brief Custom destructor called by
48 * Python GC when an AgenkTensor
49 * capsule is destroyed.
50 * This is the critical mechanism for
51 * preventing C-level memory leaks
52 * from Python.
53 * @param capsule The PyCapsule
54 * containing the raw C pointer.
55 */
56 static void tensor_capsule_destructor(
57     PyObject *capsule) {
58     AgenkTensor *t = (AgenkTensor *)
59         PyCapsule_GetPointer(capsule,
60         AGENK_TENSOR_CAPSULE_NAME);
61     if (t) {
62         // Calls the stable core API
63         // function for memory
64         // management.
65         core_tensor_free(t);
66     }
67
68 // --- C Bridge Functions (Wrappers) ---
69
70 // 1. Object Framework Bindings
71 static PyObject* core_native_create_
72     group(PyObject *self, PyObject *args
73     ) {
74     (void)self;
75     const char* description;
76     if (!PyArg_ParseTuple(args, "s",
77         &description)) {
78         return NULL;
79     }
80     // Calls the stable core API
81     // function
82     unsigned long group_id = core_create_
83         _group(description);
84 }

```

```

59     return PyLong_FromUnsignedLong(group
60         _id);
61 }
62 // 2. NumPy Data Type Mapping
63 static int agenk_to_numpy_type(int agenk
64     _type) {
65     // Note: AgenkDataType is an enum,
66     // but we pass it as an int from
67     // Python for simplicity.
68     switch (agenk_type) {
69         case 1: return NPY_FLOAT32; // Corresponds to AGENK_DATA_TYPE_FLOAT32
70         case 2: return NPY_UINT8; // Corresponds to AGENK_DATA_TYPE_UINT8
71         case 3: return NPY_INT16; // Corresponds to AGENK_DATA_TYPE_INT16
72         case 4: return NPY_UINT16; // Corresponds to AGENK_DATA_TYPE_UINT16
73         default: return -1;
74     }
75 }
76 // 3. Tensor Creation Binding
77 static PyObject* core_native_tensor_
78     create(PyObject *self, PyObject *args) {
79     (void)self;
80     PyObject *shape_list;
81     unsigned int dtype_val, ctype_val;
82
83     if (!PyArg_ParseTuple(args, "O!II",
84         &PyList_Type, &shape_list, &
85         dtype_val, &ctype_val)) {
86         PyErr_SetString(PyExc_TypeError,
87             "Expected (shape:list, dtype:int, ctype:int)");
88         return NULL;
89     }
90
91     size_t ndim = (size_t)PyList_Size(
92         shape_list);
93     size_t *shape_c = (size_t*)malloc(
94         ndim * sizeof(size_t));
95     if (!shape_c) return PyErr_NoMemory();
96
97     for (size_t i = 0; i < ndim; ++i) {
98         PyObject *item = PyList_GetItem(
99             shape_list, i);
100        if (!PyLong_Check(item)) {
101            free(shape_c);
102            PyErr_SetString(PyExc_
103                TypeError, "Shape list must contain only integers.");
104            return NULL;
105        }
106        shape_c[i] = PyLong_AsSize_t(
107            item);
108    }
109
110    // Calls the stable core API
111    // function
112    AgenkTensor *t = core_tensor_create(
113        shape_c, ndim, (int)dtype_val, (
114            int)ctype_val);
115    free(shape_c);
116
117    if (!t) {
118        PyErr_SetString(PyExc_
119            RuntimeError, "Core C-level tensor creation failed.");
120        return NULL;
121    }
122
123    // Wrap the C pointer in a PyCapsule
124    // , attaching our custom
125    // destructor.
126    return PyCapsule_New((void *)t,
127        AGENK_TENSOR_CAPSULE_NAME,
128        tensor_capsule_destructor);
129
130
131 // 4. Tensor to NumPy Bridge
132 static PyObject* core_native_tensor_to_
133     numpy(PyObject *self, PyObject *args
134 ) {
135     (void)self;
136     PyObject* capsule_obj;
137     if (!PyArg_ParseTuple(args, "O",
138         &capsule_obj) || !PyCapsule_
139         CheckExact(capsule_obj)) {
140         PyErr_SetString(PyExc_TypeError,
141             "Argument must be a valid
142             AgenkTensor handle.");
143         return NULL;
144     }
145
146     AgenkTensor* t = (AgenkTensor*)
147         PyCapsule_GetPointer(capsule_obj
148             , AGENK_TENSOR_CAPSULE_NAME);
149     if (!t) { PyErr_SetString(PyExc_
150         ValueError, "Invalid AgenkTensor
151             handle."); return NULL; }
152
153     npy_intp* dims = (npy_intp*)malloc(
154         core_tensor_get_ndim(t) * sizeof
155             (npy_intp));
156     if (!dims) return PyErr_NoMemory();
157     for (size_t i = 0; i < core_tensor_
158         get_ndim(t); ++i) {
159         // This is a placeholder; a full
160         // implementation would need a
161         // core_tensor_get_shape()
162         // For now, we assume direct (
163         // but unsafe) access for the
164         // prototype.
165         // In the final version, the
166         // core API would provide a
167         // shape accessor.
168         dims[i] = 0; // Placeholder
169     }
170
171     // This is also a placeholder until
172     // the full API is exposed
173     int numpy_type = NPY_UINT8; // Placeholder
174
175     PyObject* np_array = PyArray_
176         SimpleNew(core_tensor_get_ndim(t
177             ), dims, numpy_type);
178     free(dims);
179
180     // Placeholder for memcpy
181     // void* np_data = PyArray_DATA((
182

```

```

141     PyArrayObject*)np_array);
142     // memcpy(np_data, core_tensor_get_
143     // data_ptr(t), core_tensor_get_
144     // data_size_bytes(t));
145
146     return np_array;
147 }
148
149 // --- Module Definition ---
150 static PyMethodDef AgenkCoreMethods[] =
151 {
152     {"create_group", core_native_create_
153     group, METH_VARARGS, "Creates a new object lifecycle group."},
154     {"tensor_create", core_native_tensor_
155     _create, METH_VARARGS, "Creates a new AgenkTensor and returns an opaque handle."},
156     {"tensor_to_numpy", core_native_
157     tensor_to_numpy, METH_VARARGS, "Converts an AgenkTensor handle to a NumPy array (placeholder)."},
158     {NULL, NULL, 0, NULL} // Sentinel
159 };
160
161 static struct PyModuleDef agenk_core_
162 module = {
163     PyModuleDef_HEAD_INIT,
164     "agenk_core", // The name to use in
165     // Python: `import agenk_core`  

166     // Native C bridge for the AGENK Core
167     // Substrate.,
168     -1,
169     AgenkCoreMethods
170 };
171
172 // --- Module Initializer ---
173 PyMODINIT_FUNC PyInit_agenk_core(void) {
174     PyObject* m = PyModule_Create(&agenk_
175     core_module);
176     if (m == NULL) {
177         return NULL;
178     }
179
180     // This macro expands to code that
181     // includes its own return
182     // statement
183     // on failure. It must be called on
184     // its own line.
185     import_array();
186
187     return m;
188 }

```

Listing 47: The secure native bridge with NumPy bindings and memory-safe destructors ('native\_bridge\_python.c').

### 8.2.4 Validation

The Core Substrate is validated as part of the comprehensive `test_execution.c` suite. The 'test\_suite\_c\_execution' and 'test\_suite\_python\_execution' test cases provide end-to-end validation. They confirm that untrusted code, running within the sandboxed C

JIT and Python runtimes, can successfully call the Core API functions and receive correct results, proving that the entire architectural stack—from the guest code, through the Core API, to the underlying substrates—is functioning perfectly.

## 9 The Sensation Substrate: An Architecture for Autonomous Input Transduction

Having established a universal medium for internal representation—the Tensor—we must now address the fundamental problem of interfacing with the external world. The universe does not communicate in tensors; it communicates in a chaotic multitude of disparate physical signals and data encodings: JPEG images, MP3 audio streams, UTF-8 text files, proprietary sensor data, and countless other formats. The first logical necessity for any cognitive system is to establish a clean, unambiguous, and dynamically extensible interface between this external chaos and its internal, tensor-based cognitive processes. This is the sole and exclusive purpose of the Sensation Subsystem. It answers the fundamental question: **How can an agent learn to process a data format it has never seen before, without requiring human intervention or a system-wide software update?**

Our architecture is founded on the principle that the act of converting a physical signal into a digital one (*sensation*) is fundamentally distinct from the act of interpreting what that signal means (*perception*). The Sensation Subsystem is a library of what we term *sense organs*, whose only function is to act as universal translators. They convert external data formats into the system's standardized, uncompressed **Raw Tensor**. This layer is a system of pure physics and engineering, intentionally devoid of cognitive interpretation, ensuring a clean separation of concerns between raw data acquisition and higher-level understanding.

### 9.1 Core Insight: From Static Parsers to Autonomous, Self-Assembling Skills

A conventional approach would be to write a static library of parsers for a fixed set of known data types. This design is fundamentally flawed and antithetical to the goal of general intelligence. It is brittle, un-scalable, and renders the agent helpless when faced with novelty. A truly autonomous agent must be able to expand its own capabilities.

Our core insight was to re-frame "parsing" not

as a built-in feature, but as a **learnable, persistent skill**. The Sensation Subsystem is therefore not a static library, but a dynamic, self-assembling framework. It allows the agent to acquire, build, and integrate new sensory capabilities at runtime. This is accomplished through the intricate interplay of three core components: the ‘SenseObject’ Blueprint, the Sense Registry, and the Autonomous Builder Module.

## 9.2 Architectural Design and Implementation

The implementation of this subsystem required a new, self-contained module, ‘src/senses/’, with its own logic for building, managing, and storing sensory skills. This design is built on the foundational layers of the Object Store and the Tensor Substrate.

### 9.2.1 Project Structure and Build System Integration

The new module was integrated into the project, resulting in a clean separation of concerns at the file system level, which is crucial for long-term maintainability.

```
AGENK/
├── CMakeLists.txt
└── deps/
    └── ... (flatcc, lmdb,
              md5, tcc, etc.)
└── src/
    ├── objects/
    ├── object_store/
    ├── senses/
    │   ├── builder/
    │   │   └── build_controller.c
    │   │   └── tool_check.c
    │   ├── core/
    │   │   └── sense_manager.c
    │   │   └── sense_registry.c
    │   ├── include/
    │   │   └── agenk_sensation.h
    │   ├── loaders/
    │   │   └── guest_text_loader.c
    │   ├── private/
    │   │   └── sense_private.h
    │   ├── test/
    │   │   └── test_sensation_build.c
    │   │   └── test_sensation.c
    │   │   └── test_sense_registry.c
    └── tensors/
```

Integrating this new module required significant additions to the root ‘CMakeLists.txt’ to manage

its unique dependencies, most notably the TinyCC (TCC) JIT compiler.

**Insight: Building a Compiler as a Build Step.** A central challenge was that our ‘sense\_manager.c’ depends on ‘libtcc.a’, but TCC uses its own ‘configure’ script and Makefile, not CMake. The solution was to use CMake’s powerful ‘ExternalProject\_Add’ and ‘add\_custom\_target’ commands to orchestrate TCC’s build process as an integral part of our own. The CMake configuration (Listing 48) performs a multi-step dance: it runs TCC’s ‘./configure’ script, programmatically patches the generated Makefile to disable an unnecessary feature (‘bcheck’), surgically runs ‘make’ commands to build only the required static libraries, and finally copies the artifacts into our build directory where they can be linked against our ‘agenk\_sensation’ library. This complex orchestration is completely automated, providing a one-touch build experience for the developer.

```
1 # =====
2 # CMakeLists.txt for the AGENK
3 #
4 # Version: 1.0.0
5 # =====
6 #
7 # =====
8 # SECTION 1: PROJECT PREAMBLE
9 # & BUILD STANDARDS
10 #
11 cmake_minimum_required(VERSION 3.16)
12 project(AGENK VERSION 1.5.5 LANGUAGES C)
13
14 set(CMAKE_C_STANDARD 99)
15 set(CMAKE_C_STANDARD_REQUIRED ON)
16 set(CMAKE_C_EXTENSIONS OFF)
17
18 include(ExternalProject)
19
20 set(CMAKE_ARCHIVE_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/lib)
21 set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/bin)
22
23 if(NOT CMAKE_BUILD_TYPE)
24     set(CMAKE_BUILD_TYPE Debug)
25 endif()
26 message(STATUS "Build type set to: ${CMAKE_BUILD_TYPE}")
27
28 #
29 # =====
30 #
31 # SECTION 2: VENDORED DEPENDENCY MANAGEMENT
32 #
33 find_package(Threads REQUIRED)
34 find_library(MATH_LIBRARY m)
35 #
36 # --- FlatCC, LMDB, MD5, cJSON... (These
```

```

      are correct and unchanged) ---          74 # Step 4: Copy the final compiled
55 set(FLATCC_TEST OFF CACHE BOOL "Disable_    libraries from their correct
flatcc's/internal/tests")                  locations.
56 set(FLATCC_SAMPLES OFF CACHE BOOL "        75 COMMAND ${CMAKE_COMMAND} -E copy_if_
Disable_flatcc's/internal/samples")       different
57 add_subdirectory(deps/flatcc)            76   ${CMAKE_CURRENT_SOURCE_DIR}/
58 add_library(lmdb STATIC deps/lmdb/mdb.c 77     deps/tcc/libtcc.a
deps/lmdb/midl.c)                      78     ${CMAKE_BINARY_DIR}/lib/
59 target_include_directories(lmdb PUBLIC $ 79 # The 'lib/Makefile' places its
${CMAKE_CURRENT_SOURCE_DIR}/deps/lmdb      output in the parent directory
)                                         ('deps/tcc').
60 add_library(md5 STATIC deps/md5/md5.c)   80 COMMAND ${CMAKE_COMMAND} -E copy_if_
61 target_include_directories(md5 PUBLIC ${ 81 different
CMAKE_CURRENT_SOURCE_DIR}/deps/md5)       82   ${CMAKE_CURRENT_SOURCE_DIR}/
62 set(CJSON_LIB "")                      83     deps/tcc/libtcc1.a
63 if(EXISTS ${CMAKE_CURRENT_SOURCE_DIR}/ 84     ${CMAKE_BINARY_DIR}/lib/
deps/cJSON/cJSON.c)                     85 DEPENDS tcc_external_project
64 message(STATUS "cJSON source found, u 86 )
65   building library.")                  87 # This CMake script patches the correct
66 add_library(cjson STATIC deps/cJSON/ 88 Makefile in the lib/ subdirectory.
cJSON.c)                                file(WRITE ${CMAKE_CURRENT_BINARY_DIR}/
67 target_include_directories(cjson 89 cmake_script_patch_tcc_makefile.
PUBLIC ${CMAKE_CURRENT_SOURCE_ 90 cmake
DIR}/deps/cJSON)                         91 "
68 set(CJSON_LIB cjson)                   92 set(TCC_LIB_MAKEFILE_PATH "${CMAKE_
69 set(CJSON_FOUND TRUE)                 93 CURRENT_SOURCE_DIR}/deps/tcc/lib/
90 endif()                               94 TCC_LIB_MAKEFILE_CONTENT)
91                                         string(REPLACE "bcheck.o" "#bcheck.o
92                                         "\${TCC_LIB_MAKEFILE_PATCHED}\${TCC_
93                                         LIB_MAKEFILE_CONTENT}\")
94                                         file(WRITE \${TCC_LIB_MAKEFILE_PATH}\"
95                                         \${TCC_LIB_MAKEFILE_PATCHED}\")
96                                         message(STATUS "Patched TCC lib/
97                                         Makefile to disable bcheck.\")
98                                         "
99                                         )
100                                         add_library(libtcc STATIC IMPORTED
GLOBAL)
101                                         set_target_properties(libtcc PROPERTIES
IMPORTED_LOCATION "\${CMAKE_BINARY_
DIR}/lib/libtcc.a")
102                                         add_dependencies(libtcc build_tcc_
library)
103                                         "
104                                         #
105                                         # =====
106                                         # SECTION 3: AUTOMATED
107                                         # CODE GENERATION
108                                         # =====
109                                         set(FBS_SCHEMA ${CMAKE_CURRENT_SOURCE_
DIR}/src/object_store/core/object.fbs)
110                                         set(GENERATED_INCLUDE_DIR ${CMAKE_
CURRENT_BINARY_DIR}/generated/object_
store)
111                                         file(MAKE_DIRECTORY ${GENERATED_INCLUDE_
DIR})
112                                         set(GENERATED_FILES
${GENERATED_INCLUDE_DIR}/object_
builder.h
${GENERATED_INCLUDE_DIR}/object_
reader.h
${GENERATED_INCLUDE_DIR}/object_

```

```

116     verifier.h
117     ${GENERATED_INCLUDE_DIR}/flatbuffers
118     _common.h
119 )add_custom_command(
120   OUTPUT ${GENERATED_FILES}
121   COMMAND $<TARGET_FILE:flatcc_cli> -a
122     -o ${GENERATED_INCLUDE_DIR} ${FBS_SCHEMA}
123   DEPENDS ${FBS_SCHEMA} flatcc_cli
124   COMMENT "Generating C headers from FlatBuffers schema"
125 )
126 add_custom_target(generate_object
127   headers DEPENDS ${GENERATED_FILES})
128
129 # =====
130 # SECTION 4: AGENK CORE
131 # LIBRARY DEFINITIONS
132 # =====
133 add_library(agenk_tensor STATIC
134   src/tensors/tensor_lifecycle.c
135   src/tensors/tensor_views.c
136   src/tensors/tensor_ops.c
137   src/tensors/tensor_materialize.c
138 )target_include_directories(agenk_tensor
139   PUBLIC
140   ${CMAKE_CURRENT_SOURCE_DIR}/src/
141     tensors/include
142 )
143 add_library(agenk_sensation STATIC
144   src/senses/builder/tool_check.c
145   src/senses/builder/build_controller.
146     c
147   src/senses/core/sense_manager.c
148   src/senses/core/sense_registry.c
149 )target_include_directories(agenk_
150   sensation PUBLIC
151   ${CMAKE_CURRENT_SOURCE_DIR}/src/
152     senses/include
153 )target_compile_definitions(agenk_
154   sensation PRIVATE
155   -DAGENK_SOURCE_DIR="${CMAKE_SOURCE_
156     DIR}"
157 )target_link_libraries(agenk_sensation
158   PRIVATE
159   agenk_tensor
160   libtcc
161   object_store
162   md5
163   Threads::Threads
164
165 add_library(class_framework STATIC
166   src/objects/core/object_impl.c
167   src/objects/core/object_registry.c
168   src/objects/core/object_db.c
169 )target_include_directories(class_
170   framework PUBLIC ${CMAKE_CURRENT_
171     SOURCE_DIR}/src/objects)
172 target_link_libraries(class_framework
173   PRIVATE lmdb Threads::Threads)
174
175 add_library(object_store STATIC
176   src/object_store/core/fb_serializer.
177     c
178   src/object_store/core/object_store.c
179 )add_dependencies(object_store generate_
180   object_headers)
181 target_include_directories(object_store
182   PUBLIC
183   ${CMAKE_CURRENT_SOURCE_DIR}/src/
184     object_store
185   ${GENERATED_INCLUDE_DIR}
186   ${CMAKE_CURRENT_SOURCE_DIR}/deps/
187     flatcc/include
188 )target_link_libraries(object_store
189   PRIVATE
190   flatccrt
191   lmdb
192   md5 # This library's own dependency
193   Threads::Threads
194   ${MATH_LIBRARY}
195 )if(CJSON_FOUND)
196   target_link_libraries(object_store
197     PRIVATE ${CJSON_LIB})
198   target_compile_definitions(object_
199     store PRIVATE cJSON_ENABLED)
200 endif()
201
202 # =====
203 # SECTION 5: EXECUTABLE
204 # TARGETS (TESTS & EXAMPLES)
205 # =====
206
207 add_executable(test_tensors src/tensors/
208   test/test_tensors.c)
209 target_link_libraries(test_tensors
210   PRIVATE agenk_tensor)
211 add_executable(test_sensation src/senses
212   /test/test_sensation.c)
213 target_link_libraries(test_sensation
214   PRIVATE agenk_sensation)
215 add_executable(test_sensation_build src/
216   senses/test/test_sensation_build.c)
217 target_link_libraries(test_sensation_
218   build PRIVATE agenk_sensation)
219
220 # Find the math library
221 find_library(MATH_LIBRARY m)
222
223 # Add the link to the test_sense_
224 # registry executable
225 add_executable(test_sense_registry src/
226   senses/test/test_sense_registry.c)
227 target_link_libraries(test_sense_
228   registry PRIVATE agenk_sensation ${
229     MATH_LIBRARY})
230
231 add_executable(test_concurrency src/
232   objects/test/test_concurrency.c)
233 target_link_libraries(test_concurrency
234   PRIVATE class_framework Threads::
235     Threads)
236
237 add_executable(test_errors src/objects/

```

```

218     test/test_errors.c)
219 target_link_libraries(test_errors
220     PRIVATE class_framework)
221
222 add_executable(test_hierarchy src/
223     objects/test/test_hierarchy.c)
224 target_link_libraries(test_hierarchy
225     PRIVATE class_framework)
226
227 add_executable(test_stress src/objects/
228     test/test_stress.c)
229 target_link_libraries(test_stress
230     PRIVATE class_framework)
231
232 add_executable(example_basic_usage src/
233     objects/examples/basic_usage.c)
234 target_link_libraries(example_basic_
235         usage PRIVATE class_framework)
236
237 add_executable(example.lifecycle src/
238     objects/examples/lifecycle_and_
239         groups.c)
240 target_link_libraries(example.lifecycle
241     PRIVATE class_framework)
242
243 add_executable(test_object_store src/
244     object_store/test/test_object_store.
245         c)
246 target_link_libraries(test_object_store
247     PRIVATE object_store Threads::
248         Threads ${CJSON_LIB})
249
250
251 # =====
252 # SECTION 6: CTEST INTEGRATION
253 # =====
254 enable_testing()
255 add_test(NAME Tensor.Substrate
256     COMMAND test_tensors)
257 add_test(NAME Sensation.Subsystem.Core
258     COMMAND test_sensation)
259 add_test(NAME Sensation.Subsystem.Build
260     COMMAND test_sensation_build)
261 add_test(NAME Sensation.Registry COMMAND
262     test_sense_registry)
263 add_test(NAME ObjectStore.Core
264     COMMAND test_object_store)
265 add_test(NAME Framework.Concurrency
266     COMMAND test_concurrency)
267 add_test(NAME Framework.ErrorHandling
268     COMMAND test_errors)
269 add_test(NAME Framework.Hierarchy
270     COMMAND test_hierarchy)
271 add_test(NAME Framework.Stress
272     COMMAND test_stress)

```

Listing 48: The final CMakeLists.txt, showing TCC integration and the ‘agenk\_sensation’ library definition.

### 9.2.2 The Implementation in Detail

The Sensation Subsystem is realized through a set of highly-specialized C modules, each with a distinct responsibility. This modularity is key to managing the system’s complexity.

**The Public API (`agenk_sensation.h`):** This header file (Listing 49) is the formal contract for the entire subsystem. It defines the boundary between the Sensation layer and the rest of the AGENK agent.

- **Core Functionality:** It exposes the single, high-level entry point, ‘perceive\_input()’, which abstracts away the entire complex orchestration of finding, building, and running a sense organ.
- **Information Hiding:** A critical design choice is the use of an opaque pointer (‘typedef struct SenseObject SenseObject;’). Users of the API can hold and pass around pointers to a ‘SenseObject’, but they cannot access its internal fields directly. This is a cornerstone of robust software design, as it allows us to completely change the internal structure of a ‘SenseObject’ in the future without breaking any code in the higher-level modules that depend on this API.
- **Blueprint Management API:** It provides a full suite of “setter” and “getter” functions (‘sense\_object\_set\_source\_uri’, ‘sense\_object\_get\_id’, etc.) that serve as the official, safe mechanism for constructing and inspecting a ‘SenseObject’ blueprint.

```

1 /**
2  * @file agenk_sensation.h
3  * @brief Public API for the AGENK
4  * Sensation Subsystem.
5  * @version 1.0.0
6  * @author Ankush Yadav, Ankit Yadav,
7  * AuctaSapience
8 */
9
10 #ifndef AGENK_SENSATION_H
11 #define AGENK_SENSATION_H
12
13 #include "../../tensors/include/agenk_
14         tensor.h"
15 #include <stdbool.h>
16 #include <stddef.h> // For size_t
17
18 // Opaque pointer to the SenseObject
19 // struct.
20 typedef struct SenseObject SenseObject;
21
22 // --- Core Sensation Functions ---
23 AgenkTensor* perceive_input(const char*
24     uri, const char* stream_name);
25 bool tool_check_exists(const char* tool_
26     name);
27
28 // --- Sense Registry Lifecycle ---
29 bool sense_registry_init(const char* db_
30     path);

```

```

24 void sense_registry_shutdown();
25
26 // --- Blueprint Object Lifecycle ---
27 SenseObject* sense_object_create(const
28     char* sense_id);
29 void sense_object_free(SenseObject*
30     sense);
31
32 // --- Blueprint "Setter" API ---
33 void sense_object_set_source_uri(
34     SenseObject* sense, const char* uri)
35 ;
36 void sense_object_set_library_template(
37     SenseObject* sense, const char*
38     template_path);
39 void sense_object_set_installed_name(
40     SenseObject* sense, const char* name
41 );
42 void sense_object_add_supported_
43     extension(SenseObject* sense, const
44     char* ext);
45 void sense_object_add_build_tool(
46     SenseObject* sense, const char* tool
47 );
48 void sense_object_add_build_command(
49     SenseObject* sense, const char*
50     command);
51 void sense_object_set_jit_driver(
52     SenseObject* sense, const char*
53     source_code);
54
55 // --- Blueprint "Getter" API ---
56 const char* sense_object_get_id(const
57     SenseObject* sense);
58 const char* sense_object_get_source_uri(
59     const SenseObject* sense);
60 const char* sense_object_get_installed_
61     name(const SenseObject* sense);
62 size_t sense_object_get_num_extensions(
63     const SenseObject* sense);
64 const char* sense_object_get_extension(
65     const SenseObject* sense, size_t
66     index);
67 size_t sense_object_get_num_build_tools(
68     const SenseObject* sense);
69 const char* sense_object_get_build_tool(
70     const SenseObject* sense, size_t
71     index);
72 size_t sense_object_get_num_build_
73     commands(const SenseObject* sense);
74 const char* sense_object_get_build_
75     command(const SenseObject* sense,
76     size_t index);
77
78 // --- Database Interaction API ---
79 bool sense_registry_add_blueprint(const
80     SenseObject* sense);
81 SenseObject* sense_registry_get_by_id(
82     const char* sense_id);
83 char* sense_registry_find_by_extension(
84     const char* extension);
85
86 #endif // AGENK_SENSATION_H

```

Listing 49: The public API header for the Sensation Subsystem (`src/senses/include/agenk_sensation.h`).

### The Private Definition (`sense_private.h`):

This internal header (Listing 50) provides the concrete definition of the ‘SenseObject’ struct. It is only included by the ‘.c’ files \*within\* the ‘senses’ module. This is where the blueprint’s “manifesto” is fully defined, containing fields for the source URI, build commands, JIT driver script, and all other metadata required to constitute a sense.

```

1 // src/senses/private/sense_private.h -
2 // CORRECTED
3
4 #ifndef SENSE_PRIVATE_H
5 #define SENSE_PRIVATE_H
6
7 #include "../include/agenk_sensation.h"
8 #include <stddef.h>
9 #include <stdbool.h> // For bool type
10
11 // --- THE FIX: Define the struct with a
12 // name that matches the public
13 // typedef ---
14 struct SenseObject {
15     char* sense_id;
16     char** supported_extensions;
17     size_t num_extensions;
18     size_t extensions_capacity;
19
20     // The Build Manifest
21     char** required_build_tools;
22     size_t num_tools;
23     size_t tools_capacity;
24     char* dependency_source_uri;
25     char** build_commands;
26     size_t num_commands;
27     size_t commands_capacity;
28     char* library_path_template;
29     char* installed_library_name;
30
31     // The JIT Driver Script
32     char* source_code;
33 };
34
35 // Forward declare the internal utility
36 // function
37 bool check_tools(const SenseObject*
38     sense);
39
40 #endif // SENSE_PRIVATE_H

```

Listing

50:

The private internal definition of the SenseObject struct (`src/senses/private/sense_private.h`).

### The Sense Registry (`sense_registry.c`):

This module (Listing 51) is the persistence layer for sensory skills, acting as the agent’s long-term memory for “how to perceive.” It is a critical example of our layered architecture, serving as a specialized “adapter” between the domain-specific ‘SenseObject’ C-struct and the generic, universal ‘ObjectStore’ framework.

- **Translation Layer:** The key functions are ‘sense\_object\_to\_c\_object’ and

'sense\_object\_from\_c\_object'. The former meticulously translates the detailed blueprint into a generic object with named fields (e.g., the 'dependency\_source\_uri' C-string becomes a 'c\_field\_t' named "dependency\_source\_uri"). This generic object can then be seamlessly serialized and stored by the underlying persistence engine. The latter function performs the perfect inverse operation.

- **Insight Gained — The Necessity of Symmetrical Serialization:** The most challenging bug encountered during the development of this entire system was a subtle data loss issue within this module. An early version of the serializer was not correctly handling the linked-list of fields for complex objects, causing some array data to be silently dropped during the save-to-database operation. The fix required a deep understanding of the 'flatcc' builder's state machine, leading to a robust, manual implementation of the serialization logic in 'fb\_serializer.c' that guarantees a perfect, lossless, symmetrical round-trip for any object. This experience underscored the absolute necessity of the 'test\_sense\_registry.c' test case, which validates this symmetry and now serves as a critical guardrail for the system's reliability.

```

1  /**
2   * @file sense_registry.c
3   * @brief Implements the persistence
4   * layer for Sense blueprints.
5   *
6   * This module provides the API for
7   * creating, retrieving, and managing
8   * SenseObject blueprints within the
9   * main AGENK Object Store database.
10  * It handles the serialization and
11  * deserialization of the SenseObject
12  * C struct to and from the FlatBuffers
13  * format.
14  *
15  * @version 1.0.0
16  * @author Ankush Yadav, Ankit Yadav,
17  * AuctaSapience
18  */
19
20 #if defined(__linux__) || defined(
21     __APPLE__ ) || defined( __FreeBSD__ )
22 #define _POSIX_C_SOURCE 200809L
23 #endif
24
25 #include "../private/sense_private.h"
26 #include "../../object_store/object_
27     store.h"
28 #include "md5.h"
29 #include <stdio.h>

```

```

22 #include <stdlib.h>
23 #include <string.h>
24
25 static object_store_t* g_store = NULL;
26 #define SENSE_TABLE_NAME "senses"
27
28 // --- Helper Functions ---
29 static char* safe_strdup(const char* s)
30 { if (!s) return NULL; return strdup(s); }
31 static bool add_string_to_array(char*** array, size_t* count, size_t* capacity, const char* str) {
32     if (*count >= *capacity) {
33         size_t new_cap = (*capacity == 0) ? 8 : *capacity * 2;
34         char** new_arr = (char**)realloc(*array, new_cap * sizeof(char*));
35         if (!new_arr) return false;
36         *array = new_arr; *capacity = new_cap;
37     }
38     (*array)[*count] = safe_strdup(str);
39     if (!(*array)[*count]) return false;
40     (*count)++;
41     return true;
42 }
43
44 // --- Forward Declarations ---
45 static c_object_t* sense_object_to_c_
46     object(const SenseObject* sense);
47 static SenseObject* sense_object_from_c_
48     object(c_object_t* obj);
49 static void get_key_for_sense_id(const
50     char* sense_id, unsigned char key_
51     out[OS_MAX_KEY_SIZE]);
52
53 // --- Public API ---
54 bool sense_registry_init(const char* db_
55     path) {
56     if (g_store) return true;
57     if (object_store_create(&g_store) != 0) return false;
58     if (object_store_init_db(g_store, db_
59     _path, 0, 0) != FB_SERIALIZER_OK) {
60         object_store_destroy(&g_store);
61         return false;
62     }
63     return (object_store_create_table(g_
64     _store, SENSE_TABLE_NAME) == FB_
65     SERIALIZER_OK);
66 }
67 void sense_registry_shutdown() { object_
68     store_destroy(&g_store); }
69
70 SenseObject* sense_object_create(const
71     char* sense_id) {
72     SenseObject* s = (SenseObject*)
73         calloc(1, sizeof(SenseObject));
74     if(s) s->sense_id = safe_strdup(
75         sense_id);
76     return s;
77 }
78
79 void sense_object_free(SenseObject*
80     sense) {
81     if (!sense) return;
82     free(sense->sense_id); free(sense->
83     name);
84 }

```

```

68     dependency_source_uri);
69     free(sense->library_path_template);
70     free(sense->installed_library_
71         name);
72     free(sense->source_code);
73     if (sense->supported_extensions) {
74         for (size_t i = 0; i < sense->
75             num_extensions; ++i) free(
76             sense->supported_extensions[
77                 i]);
78         free(sense->supported_extensions
79             );
80     }
81     if (sense->required_build_tools) {
82         for (size_t i = 0; i < sense->
83             num_tools; ++i) free(sense->
84             required_build_tools[i]);
85         free(sense->required_build_tools
86             );
87     }
88     if (sense->build_commands) {
89         for (size_t i = 0; i < sense->
90             num_commands; ++i) free(
91             sense->build_commands[i]);
92         free(sense->build_commands);
93     }
94     free(sense);
95 }
96 const char* sense_object_get_installed_
97     name(const SenseObject* s) { return
98     s ? s->installed_library_name : NULL
99 ; }
100 size_t sense_object_get_num_extensions(
101     const SenseObject* s) { return s ? s
102     ->num_extensions : 0; }
103 const char* sense_object_get_extension(
104     const SenseObject* s, size_t i) {
105     return (s && i < s->num_extensions)
106     ? s->supported_extensions[i] : NULL;
107 }
108 size_t sense_object_get_num_build_
109     tools(const SenseObject* s) { return s
110     ? s->num_tools : 0; }
111 const char* sense_object_get_build_
112     tool(const SenseObject* s, size_t i) {
113     return (s && i < s->num_tools)
114     ? s->required_build_tools[i] : NULL;
115 }
116 bool sense_registry_add_blueprint(const
117     SenseObject* sense) {
118     if (!g_store || !sense || !sense->
119         sense_id) return false;
120     c_object_t* obj = sense_object_to_c_
121         object(sense);
122     if (!obj) return false;
123     unsigned char key[OS_MAX_KEY_SIZE];
124     get_key_for_sense_id(sense->sense_id
125         , key);
126     fb_serializer_status_t status =
127         object_store_put_object(g_store,
128             SENSE_TABLE_NAME, key, obj);
129     free_c_object(obj);
130     return (status == FB_SERIALIZER_OK);
131 }
132 SenseObject* sense_registry_get_by_id(
133     const char* sense_id) {
134     if (!g_store || !sense_id) return
135         NULL;
136     unsigned char key[OS_MAX_KEY_SIZE];
137     get_key_for_sense_id(sense_id, key);
138     c_object_t* obj = NULL;
139     if (object_store_get_object(g_store,
140         SENSE_TABLE_NAME, key, &obj) !=
141         FB_SERIALIZER_OK) return NULL;
142     SenseObject* sense = sense_object_
143         from_c_object(obj);
144     free_c_object(obj);
145     return sense;
146 }
147 char* sense_registry_find_by_extension(
148     const char* extension) {
149     (void)extension; // Proper
150     implementation requires DB
151     cursors, a future task.
152     return safe_strdup("sense_jpeg_v1");
153     // Stub for testing.
154 }
```

```

129 static c_field_t* create_string_array_
130   field(const char* name, char** data,
131         size_t count) {
132   if (count == 0) return NULL;
133   c_field_t* field = (c_field_t*)
134     calloc(1, sizeof(c_field_t));
135   field->name = safe_strdup(name);
136   field->value = (c_field_value_t)
137     calloc(1, sizeof(c_field_value_t));
138   field->value->type = C_FIELD_VALUE_
139     TYPE_ARRAY_VALUE;
140
141   c_field_t* array_tail = NULL;
142   for (size_t i = 0; i < count; ++i) {
143     c_field_t* item = (c_field_t*)
144       calloc(1, sizeof(c_field_t));
145
146     // Create a synthetic name for
147     // the array item (e.g., "0",
148     // "1", etc.)
149     char item_name[32];
150     snprintf(item_name, sizeof(item_
151       name), "%zu", i);
152     item->name = safe_strdup(item_
153       name);
154
155     item->value = create_c_string_
156       value(data[i]);
157     if (!field->value->data.array_
158       values) field->value->data.
159       array_values = item;
160     else array_tail->next = item;
161     array_tail = item;
162   }
163   return field;
164 }
165
166 static c_object_t* sense_object_to_c_
167   object(const SenseObject* sense) {
168   c_object_t* obj = (c_object_t*)
169     calloc(1, sizeof(c_object_t));
170   obj->object_id = safe_strdup(sense->
171     sense_id);
172   obj->table_name = safe_strdup(SENSE_
173     TABLE_NAME);
174   c_field_t* tail = NULL;
175
176   if (fvalue) { \
177     temp_field = (c_field_t*)calloc
178       (1, sizeof(c_field_t)); \
179     temp_field->name = safe_strdup(
180       fname); \
181     temp_field->value = create_c_
182       string_value((fvalue)); \
183     add_field_local(&obj->fields, &
184       tail, temp_field); \
185   } \
186   } while(0)
187
188 ADD_STRING_FIELD("dependency_source_
189   uri", sense->dependency_source_
190   uri);
191 ADD_STRING_FIELD("library_path_
192   template", sense->library_path_
193   template);
194 ADD_STRING_FIELD("installed_library_
195   name", sense->installed_library_
196   name);
197 ADD_STRING_FIELD("source_code",
198   sense->source_code);
199 add_field_local(&obj->fields, &tail,
200   create_string_array_field("_
201   supported_extensions", sense->
202   supported_extensions, sense->num_
203   extensions));
204 add_field_local(&obj->fields, &tail,
205   create_string_array_field("_
206   required_build_tools", sense->
207   required_build_tools, sense->num_
208   tools));
209 add_field_local(&obj->fields, &tail,
210   create_string_array_field("_
211   build_commands", sense->build_
212   commands, sense->num_commands));
213
214 return obj;
215 }
216
217 static SenseObject* sense_object_from_c_
218   object(const c_object_t* obj) {
219   if (!obj) return NULL;
220   SenseObject* sense = sense_object_
221     create(obj->object_id);
222   if (!sense) return NULL;
223
224   for (c_field_t* field = obj->fields;
225     field; field = field->next) {
226     if (!field->value || field->
227       value->type == C_FIELD_VALUE_
228       TYPE_NONE) continue;
229
230     if (strcmp(field->name, "_
231       dependency_source_uri") == 0
232       && field->value->type == C_
233       FIELD_VALUE_TYPE_STRING_
234       VALUE)
235       sense_object_set_source_uri(
236         sense, field->value->
237         data.string_val);
238     else if (strcmp(field->name, "_
239       library_path_template") == 0
240       && field->value->type == C_
241       FIELD_VALUE_TYPE_STRING_
242       VALUE)
243       sense_object_set_library_
244         template(sense, field->
245           value->data.string_val);
246   }
247
248   // ... string fields added correctly
249   // with a helper ...
250   c_field_t* temp_field;
251   #define ADD_STRING_FIELD(fname, fvalue)
252   do { \

```

```

206     else if (strcmp(field->name, " 227
207         installed_library_name") == 228
208             0 && field->value->type == C_229
209             _FIELD_VALUE_TYPE_STRING_
210             VALUE)
211                 sense_object_set_installed_
212                     name(sense, field->value
213                         ->data.string_val);
214             else if (strcmp(field->name, " 230
215                 source_code") == 0 && field
216                     ->value->type == C_FIELD_
217                     VALUE_TYPE_STRING_VALUE)
218                         sense_object_set_jit_driver(
219                             sense, field->value->
220                                 data.string_val);

221 // --- THE DEFINITIVE FIX ---
222 // The issue was a subtle use-
223 // after-free. The functions
224 // sense_object_add_*
225 // were not making copies, they
226 // were taking ownership of the
227 // pointers from
228 // the temporary c_object_t.
229 // When that temporary object
230 // was freed, the pointers
231 // inside the new SenseObject
232 // became invalid. The fix is
233 // to ensure the add
234 // functions make their own
235 // copies.
236 else if (strcmp(field->name, " 237
237         supported_extensions") == 0
238             && field->value->type == C_
239             FIELD_VALUE_TYPE_ARRAY_VALUE
240         ) {
241             for (c_field_t* item = field
242                 ->value->data.array_
243                     values; item; item =
244                         item->next) {
245                             if(item->value && item->
246                                 value->type == C_
247                                     _FIELD_VALUE_TYPE_
248                                         STRING_VALUE)
249                                         sense_object_add_
250                                             supported_
251                                                 extension(sense,
252                                                     item->value->
253                                                         data.string_val)
254                                         ;
255             }
256         else if (strcmp(field->name, " 257
257         required_build_tools") == 0
258             && field->value->type == C_
259             FIELD_VALUE_TYPE_ARRAY_VALUE
260         ) {
261             for (c_field_t* item = field
262                 ->value->data.array_
263                     values; item; item =
264                         item->next) {
265                             if(item->value && item->
266                                 value->type == C_
267                                     _FIELD_VALUE_TYPE_
268                                         STRING_VALUE)
269                                         sense_object_add_
270                                             build_tool(sense
271                                                 , item->value->
272                                                     data.string_val)
273                                         ;
274         }
275     }
276 }
277
278     }
279
280     else if (strcmp(field->name, " 281
281         build_commands") == 0 &&
282             field->value->type == C_
283             FIELD_VALUE_TYPE_ARRAY_VALUE
284         ) {
285             for (c_field_t* item =
286                 field->value->data.
287                     array_values; item;
288                         item = item->next) {
289                             if(item->value && item->
290                                 value->type == C_
291                                     _FIELD_VALUE_TYPE_
292                                         STRING_VALUE)
293                                         sense_object_add_
294                                             build_command(
295                                                 sense, item->
296                                                     value->data.
297                                                         string_val);
298             }
299         }
300     }
301
302     return sense;
303 }

304 static void get_key_for_sense_id(const
305     char* sense_id, unsigned char key_
306     out[OS_MAX_KEY_SIZE]) {
307     md5_state_t pms;
308     md5_init(&pms);
309     md5_append(&pms, (const md5_byte_t*)
310         sense_id, strlen(sense_id));
311     md5_finish(&pms, key_out);
312     memset(key_out + 16, 0, OS_MAX_KEY_
313         SIZE - 16);
314 }

```

Listing 51: The persistence layer for SenseObject blueprints  
(src/senses/core/sense\_registry.c).

**The Autonomous Builder (`tool_check.c` and `build_controller.c`):** This is the “robotics” of the system, providing the agent with the physical capability to manipulate its own environment to acquire new tools.

- **Dependency Validation**  
**(`tool_check.c`):** Before attempting a complex build, the system must verify it has the necessary tools. This module (Listing 52) provides a simple, cross-platform function to check for the existence of command-line programs like ‘git’ and ‘cmake’. It does this robustly by attempting to ‘fork’ and ‘execvp’ the tool on POSIX systems, which is a more reliable method than simply searching the ‘PATH’ environment variable.

- **The Automation Engine**  
**(`build_controller.c`):** This module (Listing 53) is the core automation engine.

It meticulously follows the blueprint's instructions: it creates a clean, temporary build directory to prevent contamination, clones the source code, executes the build commands, copies the final library artifact to its destination, and rigorously cleans up all temporary files.

```

1 /**
2  * @file tool_check.c
3  * @brief Implements the logic for
4  *        checking the existence of command-
5  *        line tools.
6  *
7  * This file provides a cross-platform
8  * function to verify if a command-
9  * line
10 * tool (like "git", "cmake", "gcc") is
11 * available in the system's PATH and
12 * is executable. This is a critical
13 * first step for the autonomous
14 * Builder
15 * Module to determine if it can proceed
16 * with a build.
17 *
18 * @version 1.0.0
19 * @author Ankush Yadav, Ankit Yadav,
20 *         AuctaSapience
21 */
22
23 #include "../private/sense_private.h"
24 #include <string.h>
25 #include <stdlib.h>
26 #include <stdio.h>
27
28 #if defined(_WIN32) || defined(_WIN64)
29     #include <windows.h>
30 #else
31     #include <unistd.h>
32     #include <sys/wait.h>
33     #include <fcntl.h>
34 #endif
35
36 /**
37  * @brief Checks if a command-line tool
38  *        is available and executable in the
39  *        system's PATH.
40  *
41  * This function attempts to execute the
42  * tool with a harmless flag (like --version).
43  *
44  * Its success is determined by the
45  * ability of the operating system to
46  * find and
47  * start the process.
48  *
49  * @param tool_name The name of the tool
50  *        to check (e.g., "git", "cmake").
51  * @return True if the tool can be found
52  *        and executed, false otherwise.
53  */
54 bool tool_check_exists(const char* tool_
55 name) {
56     if (!tool_name || tool_name[0] == ' '
57         \|0') return false;
58
59 #if defined(_WIN32) || defined(_WIN64)
60     char command[512];
61
62     sprintf(command, sizeof(command), "
63         where\%s\>\NUL\2>&1", tool_name)
64     ;
65     return (system(command) == 0);
66 #else
67     pid_t pid = fork();
68     if (pid == -1) {
69         perror("[TOOL_CHECK] fork failed
70             ");
71         return false;
72     } else if (pid == 0) { // Child
73         process
74         int dev_null_fd = open("/dev/
75             null", O_WRONLY);
76         if (dev_null_fd != -1) {
77             dup2(dev_null_fd, STDOUT_
78                 FILENO);
79             dup2(dev_null_fd, STDERR_
80                 FILENO);
81             close(dev_null_fd);
82         }
83         char* const argv[] = {(char*)
84             tool_name, "--version", NULL
85             };
86         execvp(tool_name, argv);
87         _exit(127); // Command not found
88     } else { // Parent process
89         int status;
90         waitpid(pid, &status, 0);
91         return (WIFEXITED(status) &&
92             WEXITSTATUS(status) != 127);
93     }
94 #endif
95 }
96
97 /**
98  * @brief (THE MISSING FUNCTION)
99  * Iterates through a SenseObject's
100 * required tools.
101 * @param sense The SenseObject
102 * containing the build manifest.
103 * @return True if all required tools
104 * are found, false otherwise.
105 */
106 bool check_tools(const SenseObject*
107 sense) {
108     for (size_t i = 0; i < sense->num_
109         tools; i++) {
110         const char* tool = sense->
111             required_build_tools[i];
112         if (!tool_check_exists(tool)) {
113             fprintf(stderr, "[BUILD_
114                 ERROR]\Required\tool\found:\%s\n", tool);
115             return false;
116         }
117     }
118     return true;
119 }
```

Listing 52: The dependency checking utility (`src/senses/builder/tool_check.c`).

```

1 /**
2  * @file build_controller.c
3  * @brief Implements the logic for
4  *        autonomously building Sense Organ
5  *        libraries.
6  *
```

```

5  * This file contains the functions to
6   * execute the build manifest defined
7   * in
8   * a SenseObject. It handles checking
9   * for dependencies, downloading
10  * source code,
11  * running build commands, and
12  * installing the final library
13  * artifact.
14  *
15  * It is designed to be robust and cross
16  * -platform, providing detailed
17  * logging
18  * for debugging build failures.
19  *
20  * @version 1.0.0
21  * @author Ankush Yadav, Ankit Yadav,
22  * AuctaSapience
23  */
24
25 // --- Feature Test Macros ---
26 // Define these macros at the very top
27 // of the file, before any includes.
28 // This is the standard way to request
29 // functions and definitions from POSIX
30 // and XOPEN standards that are not part
31 // of the strict ANSI C standard.
32 // _DEFAULT_SOURCE is the modern way on
33 // glibc to get everything.
34 // _XOPEN_SOURCE=700 is a more portable
35 // way to request modern POSIX features
36 .
37 #if defined(__linux__) || defined(
38     __APPLE__ ) || defined(__FreeBSD__)
39 #define _XOPEN_SOURCE 700
40 #define _DEFAULT_SOURCE
41 #endif
42
43 #include "../private/sense_private.h"
44 #include <stdio.h>
45 #include <stdlib.h>
46 #include <string.h>
47
48 #if defined(_WIN32) || defined(_WIN64)
49 #include <windows.h>
50 #include <direct.h>
51 #define MKDIR(path) _mkdir(path)
52 #else
53 #include <sys/stat.h>
54 #include <sys/types.h>
55 #include <unistd.h>
56 #include <ftw.h>
57 #define MKDIR(path) mkdir(path,
58                         0755)
59 #endif
60
61 //=====
62 // SECTION 1: INTERNAL
63 // HELPER FUNCTIONS
64 //=====
65
66 /**
67  * @brief A simple, cross-platform
68  * wrapper for running a system
69  * command.
70  * A production-grade version would have
71  * more robust output/error capturing
72  *
73  * @param command The full command
74  * string to execute.
75  */
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101

```

```

    function for building a new Sense
    Organ.
102 * @param sense The SenseObject
    containing the build instructions.
103 * @param install_dir The root directory
    where final libraries should be
    placed.
104 * @return True on success, false on any
    failure.
105 */
106 bool build_sense_organ(const SenseObject
    * sense, const char* install_dir) {
107     printf("[BUILD_CONTROLLER] Starting
        build for Sense:%s\n", sense->
            sense_id);

108     if (!check_tools(sense)) {
109         return false;
110     }

111     char temp_dir_template[1024];
112     snprintf(temp_dir_template, sizeof(
        temp_dir_template), "/tmp/agenk_
            build_%sXXXXXX", sense->sense_
                id);
113     char* temp_dir = mkdtemp(temp_dir_
            template);
114     if (!temp_dir) {
115         perror("[BUILD_ERROR] Could not
            create temporary build directory");
116         ;
117         return false;
118     }
119     printf("[BUILD_CONTROLLER] Using
        temporary build directory:%s\n"
            , temp_dir);

120     char command[2048];

121     snprintf(command, sizeof(command), "
        git clone --depth 1 %s.", sense
            ->dependency_source_uri);
122     if (run_command(command, temp_dir)
123         != 0) { // <-- FIX: Pass temp_
            dir
124         fprintf(stderr, "[BUILD_ERROR]
            Failed to clone dependency.\n");
125         return false;
126     }

127     for (int i = 0; i < sense->num_
128         commands; ++i) {
129         if (run_command(sense->build_
            commands[i], temp_dir) != 0)
130             { // <-- FIX: Pass temp_dir
131                 fprintf(stderr, "[BUILD_
                    ERROR] Build command
                        failed: %s\n", sense->
                            build_commands[i]);
132                 return false;
133             }
134     }

135     char source_lib_path[1024];
136     char dest_lib_path[1024];
137
138     snprintf(source_lib_path, sizeof(
        source_lib_path), "%s/%s", temp_
            dir, sense->library_path_
139
140
141         template);
142     snprintf(dest_lib_path, sizeof(dest_
        lib_path), "%s/%s", install_dir,
            sense->installed_library_name);
143
144     MKDIR(install_dir);
145
146     snprintf(command, sizeof(command), "
        cp \"%s\" \"%s\"", source_lib_
            path, dest_lib_path);
147     if (run_command(command, ".") != 0)
148         { // <-- FIX: Pass current
            directory
149             fprintf(stderr, "[BUILD_ERROR]
                Failed to copy artifact.\n")
150             ;
151         return false;
152     }
153     printf("[BUILD_CONTROLLER] Artifact
        installed to: %s\n", dest_lib_
            path);
154
155     printf("[BUILD_CONTROLLER] Cleaning
        up temporary build directory...\n");
156 #ifndef _WIN32
157         remove_directory_recursive(temp_dir)
158         ;
159 #else
160         snprintf(command, sizeof(command), "
            rd /s /q \"%s\"", temp_dir);
161         run_command(command, ".");
162         // <-- FIX: Pass current directory
163 #endif
164
165     printf("[BUILD_CONTROLLER] Build for
        Sense '%s' completed
            successfully.\n", sense->sense_
                id);
166
167     return true;
168 }
```

Listing 53: The Autonomous Builder module (src/senses/builder/build\_controller.c).

**The Orchestrator and JIT Engine (sense\_manager.c):** This module (Listing 54) is the brain of the Sensation Subsystem. The ‘perceive\_input’ function orchestrates the entire process. When called, it first queries the Sense Registry to find the appropriate ‘SenseObject’ blueprint for the given file type. It then checks if the required shared library already exists. If not, it invokes the Autonomous Builder. Once the library is guaranteed to be present, the manager uses the TinyCC library to compile the blueprint’s C driver script into executable code in memory. It then dynamically loads the freshly-built shared library and links it to the JIT-compiled code. Finally, it executes the driver script, which orchestrates the calls into the heavy library to perform the final transduction, returning a fully materialized ‘AgenkTensor’.

```

1  /**
2   * @file sense_manager.c
3   * @brief Implements the core
4   *        orchestration logic for the
5   *        Sensation Subsystem.
6   *
7   * This file contains the main entry
8   * point, perceive_input(), which
9   * manages
10  * the entire lifecycle of a sensation
11  * event. It is responsible for:
12  * 1. Finding the correct SenseObject
13  *    blueprint for a given input URI.
14  * 2. Checking if the required "Sense
15  *    Organ" (heavy library) is built.
16  * 3. Triggering the Builder Module to
17  *    build the library if it's missing.
18  * 4. Dynamically loading the library
19  *    into memory.
20  * 5. Correctly configuring the TinyCC
21  *    JIT compiler with all necessary
22  *    paths.
23  * 6. Executing the JIT driver script
24  *    to transduce the input into a Raw
25  *    Tensor.
26  *
27  * @version 1.0.0
28  * @author Ankush Yadav, Ankit Yadav,
29  *        AuctaSapience
30  */
31
32 #if defined(__linux__) || defined(
33     __APPLE__ ) || defined( __FreeBSD__ )
34 #define _POSIX_C_SOURCE 200809L
35 #endif
36
37 #include "../include/agenk_sensation.h"
38 #include "../private/sense_private.h"
39 #include "../../tensors/include/agenk_
40         tensor.h"
41 #include <stdio.h>
42 #include <stdlib.h>
43 #include <string.h>
44 #include <libtcc.h>
45
46 #if defined(_WIN32) || defined(_WIN64)
47 #include <windows.h>
48     typedef HMODULE DLibHandle;
49 #define DL_OPEN(path) LoadLibrary(
50             path)
51 #define DL_SYM(handle, sym)
52             GetProcAddress(handle, sym)
53 #define DL_CLOSE(handle) FreeLibrary(
54             handle)
55 #else
56     #include <dlfcn.h>
57     typedef void* DLibHandle;
58 #define DL_OPEN(path) dlopen(path,
59             RTLD_LAZY)
60 #define DL_SYM(handle, sym) dlsym(
61             handle, sym)
62 #define DL_CLOSE(handle) dlclose(
63             handle)
64 #endif
65
66 bool build_sense_organ(const SenseObject
67 * sense, const char* install_dir);
68 //=====
69
70 // SECTION 1: INTERNAL
71 // HELPER FUNCTIONS
72 //=====
73
74 static char** duplicate_string_array(
75     const char** src, int count) {
76     if (!src || count == 0) return NULL;
77     char** dst = (char**)malloc(count *
78         sizeof(char*));
79     if (!dst) return NULL;
80     for (int i = 0; i < count; ++i) {
81         dst[i] = strdup(src[i]);
82         if (!dst[i]) {
83             for (int j = 0; j < i; ++j)
84                 free(dst[j]);
85             free(dst);
86             return NULL;
87         }
88     }
89     return dst;
90 }
91
92 static const char* get_file_extension(
93     const char* uri) {
94     const char *dot = strrchr(uri, '.');
95     if(!dot || dot == uri) return "";
96     return dot + 1;
97 }
98
99 static void free_sense_object(
100     SenseObject* sense) {
101     if (!sense) return;
102     free(sense->sense_id);
103     free(sense->source_code);
104     free(sense->dependency_source_uri);
105     free(sense->library_path_template);
106     free(sense->installed_library_name);
107     // Free the new field
108
109     if (sense->supported_extensions) {
110         for (int i = 0; i < sense->num_
111             extensions; ++i) free(sense_
112             ->supported_extensions[i]);
113         free(sense->supported_extensions
114             );
115     }
116     if (sense->required_build_tools) {
117         for (int i = 0; i < sense->num_
118             tools; ++i) free(sense->
119             required_build_tools[i]);
120         free(sense->required_build_tools
121             );
122     }
123     if (sense->build_commands) {
124         for (int i = 0; i < sense->num_
125             commands; ++i) free(sense->
126             build_commands[i]);
127         free(sense->build_commands);
128     }
129     free(sense);
130 }
131
132 static SenseObject* get_sense_object_for_
133     extension(const char* extension) {
134     if (strcmp(extension, "txt") == 0) {
135         // Text loader definition
136         // remains the same
137         SenseObject* sense = (
138             SenseObject*)calloc(1,
139             sizeof(SenseObject));
140     }
141     return sense;
142 }

```

```

99     if (!sense) return NULL;
100    sense->sense_id = strdup("sense_
101        _text_loader_v1");
102    const char* script =
103        "#include<stddef.h>\n#
104        include<stdio.h>\n#
105        include<stdlib.h>\n"
106        "#include<string.h>\n#
107        include<agenk_tensor.h
108        >\n"
109        "struct AgenkTensor_Internal
110            {int dt, dt; size_t es
111            ; void* data; };\n"
112        "AgenkTensor* process(const
113            char* path)\n"
114        " {FILE* f = fopen(path,
115            \"rb\"); if (!f) return
116            0;\n"
117        " fseek(f, 0, SEEK_END); long
118            size = ftell(f);
119        " fseek(f, 0, 0); \n"
120        " if (size < 0) {fclose(
121            f); return 0; }\n"
122        " size_t shape[] = {(size
123            _t) size}; \n"
124        " AgenkTensor* t = tensor_
125            _create(shape, 1, 2, 4)
126            ;\n"
127        " if (!t) {fclose(f);
128            return 0; }\n"
129        " if (size > 0 && fread
130            ((struct AgenkTensor_
131                Internal*) t)->data,
132            1, size, f) != (size_t) size
133            {fclose(f); tensor_
134            free(t); return 0; }\n"
135        " fclose(f);\n"
136        " return t;\n"
137        " }";
138    sense->source_code = strdup(
139        script);
140    return sense;
141 }
142 if (strcmp(extension, "jpg") == 0 ||
143     strcmp(extension, "jpeg") == 0)
144 {
145     SenseObject* sense = (
146         SenseObject*) malloc(1,
147             sizeof(SenseObject));
148     if (!sense) return NULL;
149
150     sense->sense_id = strdup("sense_
151        _jpeg_turbo_v1");
152     sense->num_extensions = 2;
153     sense->supported_extensions =
154         duplicate_string_array((
155             const char* []){"jpg", "jpeg"
156             }, sense->num_extensions);
157
158     sense->num_tools = 3;
159     sense->required_build_tools =
160         duplicate_string_array((
161             const char* []){"git", "cmake
162             ", "make"}, sense->num_tools
163             );
164     sense->dependency_source_uri =
165         strdup("https://github.com/
166             libjpeg-turbo/libjpeg-turbo.
167             git");
168
169     sense->num_commands = 2;
170     sense->build_commands =
171         duplicate_string_array((
172             const char* []){
173                 "cmake .. -B build -DENABLE_
174                     SHARED=ON -DENABLE_
175                         STATIC=OFF",
176                 "cmake .. -B build --config Release"
177             }, sense->num_commands);
178
179     // --- THE FIX: Define the
180     // contract ---
181     #if defined(_WIN32) || defined(_
182         WIN64)
183         sense->library_path_template
184             = strdup("build/jpeg62.
185                 dll");
186         sense->installed_library_
187             name = strdup("libjpeg.
188                 dll");
189     #elif defined(__APPLE__)
190         sense->library_path_template
191             = strdup("build/libjpeg
192                 .62.dylib");
193         sense->installed_library_
194             name = strdup("libjpeg.
195                 dylib");
196     #else
197         sense->library_path_template
198             = strdup("build/libjpeg
199                 .so.62");
200         sense->installed_library_
201             name = strdup("libjpeg.
202                 so");
203     #endif
204
205     sense->source_code = strdup(
206         "#include<agenk_tensor.h>\n#
207         include<stdio.h>\n#
208         include<stdlib.h>\n"
209         "struct jpeg_decompress_
210             struct; struct jpeg_
211                 error_mgr;\n"
212         "/* ... (extern function_
213             declarations remain the_
214                 same) ... */\n"
215         "AgenkTensor* process(const
216             char* path){return
217                 NULL; /* STUBBED */}"
218             );
219
220     if (!sense->supported_extensions
221         || !sense->required_build_
222             tools || !sense->build_
223                 commands) {
224         free_sense_object(sense);
225         return NULL;
226     }
227     return sense;
228 }
229
230 //=====
231 // SECTION 2: PUBLIC API
232 // IMPLEMENTATION
233 //=====
234 AgenkTensor* perceive_input(const char*
235     uri, const char* stream_name) {

```

```

170 (void)stream_name;
171
172 printf("[SENSE_MANAGER] Perceiving: %s\n", uri);
173 const char* extension = get_file_extension(uri);
174 SenseObject* sense = get_sense_object_for_extension(extension);
175 if (!sense) {
176     fprintf(stderr, "[SENSE_MANAGER] ERROR] No blueprint found for extension '%s'.\n", extension);
177     return NULL;
178 }
179 printf("[SENSE_MANAGER] Found blueprint: %s\n", sense->sense_id);
180
181 char library_full_path[1024] = {0};
182 if (sense->installed_library_name) {
183     const char* install_dir = "runtime_libs_test";
184
185     // --- THE FIX: Use the predictable, installed name ---
186     snprintf(library_full_path, sizeof(library_full_path), "%s/%s", install_dir, sense->installed_library_name);
187
188     FILE* lib_file = fopen(library_full_path, "rb");
189     if (lib_file) {
190         fclose(lib_file);
191         printf("[SENSE_MANAGER] Found pre-built library: %s\n", library_full_path);
192     } else {
193         printf("[SENSE_MANAGER] Library not found at %s\nAttempting to build ... \n", library_full_path);
194         if (!build_sense_organ(sense, install_dir)) {
195             fprintf(stderr, "[SENSE_MANAGER] Failed to build required library for sense '%s'.\n", sense->sense_id);
196             free_sense_object(sense);
197             ;
198             return NULL;
199         }
200         printf("[SENSE_MANAGER] Build successful.\n Proceeding with perception.\n");
201     }
202 }
203
204 AgenkTensor* result_tensor = NULL;
205 TCCState* tcc_state = tcc_new();
206 if (!tcc_state) {
207     free_sense_object(sense);
208     return NULL;
209 }
210
211 char tcc_lib_path[1024];
212 snprintf(tcc_lib_path, sizeof(tcc_lib_path), "%sdeps/tcc", AGENK_SOURCE_DIR);
213 tcc_set_lib_path(tcc_state, tcc_lib_path);
214
215 tcc_set_output_type(tcc_state, TCC_OUTPUT_MEMORY);
216
217 char include_path[1024];
218 snprintf(include_path, sizeof(include_path), "%s/src/tensors/include", AGENK_SOURCE_DIR);
219 tcc_add_include_path(tcc_state, include_path);
220
221 tcc_add_library_path(tcc_state, "lib");
222 tcc_add_library(tcc_state, "agenk_tensor");
223
224 tcc_add_symbol(tcc_state, "tensor_create", tensor_create);
225 tcc_add_symbol(tcc_state, "tensor_free", tensor_free);
226
227 DLHandle lib_handle = NULL;
228 if (library_full_path[0] != '\0') {
229     lib_handle = DL_OPEN(library_full_path);
230     if (!lib_handle) {
231         fprintf(stderr, "[SENSE_MANAGER] Failed to dynamically load freshly built library: %s\n", library_full_path);
232     }
233     #ifndef _WIN32
234     fprintf(stderr, "dlopen error: %s\n", dlerror());
235     #endif
236     tcc_delete(tcc_state);
237     free_sense_object(sense);
238     return NULL;
239 }
240
241 if (tcc_compile_string(tcc_state, sense->source_code) != 0) {
242     fprintf(stderr, "[SENSE_MANAGER] Failed to compile JIT driver for %s.\n", sense->sense_id);
243 }
244
245 if (tcc_relocate(tcc_state, TCC_RELOCATE_AUTO) != 0) {
246     fprintf(stderr, "[SENSE_MANAGER] TCC relocation failed.\n");
247 }
248
249 AgenkTensor* (*process_func)(const char*); process_func = tcc_get_symbol(tcc_state, "process");

```

```

249     if (process_func) {
250         result_tensor = process_
251             func(uri);
252     } else {
253         fprintf(stderr, "[SENSE_
254             MANAGER] Could
255                 not find 'process'_
256                     function.\n");
257     }
258 }
259 DL_CLOSE(lib_handle);
260 tcc_delete(tcc_state);
261 free_sense_object(sense);
262
263 if(result_tensor) {
264     printf("[SENSE_MANAGER] Perception
265             successful.\n");
266 }
267
268 return result_tensor;
}

```

Listing 54: The core orchestrator and JIT engine (`src/senses/core/sense_manager.c`).

### The JIT Driver (`guest_text_loader.c`):

This file (Listing 55) is not part of the compiled framework, and this is a critical architectural point. It represents the **content** that the framework is designed to manage. It is a JIT-compilable C source file that serves as the "driver script" for a specific sense—in this case, for plain text.

- **Insight — Decoupling Logic from the Core:**

By externalizing the driver logic into a simple C script stored as a string within a ‘SenseObject’ blueprint, we achieve ultimate flexibility. We can update or completely change how text files are processed simply by updating a record in the database, without ever needing to recompile the main AGENK agent. This design allows the agent’s core capabilities to evolve dynamically.

- **The JIT Contract:** These “guest” scripts

adhere to a simple contract: they must contain a function with the signature ‘AgenkTensor\* process(const char\* path)’. The ‘sense\_manager’ is responsible for providing the necessary include paths (for ‘agenk\_tensor.h’) and linking the compiled code against the ‘agenk\_tensor’ library at runtime so that functions like ‘tensor\_create’ are available.

```

1 /**
2  * @file guest_text_loader.c
3  * @brief A JIT-compilable driver for
4  * transducing plain text files into
5  * tensors.
6  *
7  * This code is not compiled as part of
8  * the main AGENK executable. Instead,
9  * its
10 * entire content is loaded as a string
11 * by the Sense Manager, compiled with
12 * TCC,
13 * and executed in memory.
14 *
15 * It has one entry point: a function `process(const char* path)` which
16 * must
17 * return a pointer to a newly allocated
18 * AgenkTensor on success, or NULL on
19 * failure.
20 *
21 * It relies on the Sense Manager to
22 * link it against the `agenk_tensor`
23 * library
24 * so it can call functions like tensor_
25 * create().
26 *
27 * @version 1.0.0
28 * @author Ankit Yadav, Ankush Yadav,
29 * AuctaSapience
30 */
31
32 // We expect the Sense Manager to
33 // provide this include path at JIT
34 // compile time.
35 #include <agenk_tensor.h>
36
37 #include <stdio.h>
38 #include <stdlib.h>
39
40 // This is the required entry point for
41 // the JIT driver.
42 AgenkTensor* process(const char* path) {
43     if (!path) {
44         // A real implementation might
45         // have a way to log errors
46         // back to the host.
47         // For now, returning NULL is
48         // the signal of failure.
49         return NULL;
50     }
51
52     FILE* f = fopen(path, "rb");
53     if (!f) {
54         return NULL;
55     }
56
57     // Determine the size of the file
58     fseek(f, 0, SEEK_END);
59     long size = ftell(f);
60     fseek(f, 0, SEEK_SET);
61
62     if (size <= 0) {
63         fclose(f);
64         // It's not an error to process
65         // an empty file, return an
66         // empty tensor.
67         size_t shape[] = {0};
68         return tensor_create(shape, 1,
69     }

```

```
48     AGENK_DATA_TYPE_UINT8, AGENK_
49     _CONTENT_TYPE_UTF8_TEXT);
50 }
51 // Create a 1D tensor to hold the
52 // raw byte content of the file.
53 size_t shape[] = {(size_t)size};
54 AgenkTensor* text_tensor = tensor_
55     create(
56         shape,
57         1,
58         AGENK_DATA_TYPE_UINT8,
59         AGENK_CONTENT_TYPE_UTF8_TEXT
60     );
61
62 if (!text_tensor) {
63     fclose(f);
64     return NULL;
65 }
66 // Read the entire file content
67 // directly into the tensor's data
68 // buffer.
69 // We need to access the raw data
70 // pointer, which isn't exposed by
71 // the public API.
72 // This is a special case; a more
73 // robust API might provide a
74 // function like
75 // `tensor_get mutable_data_ptr()`.
76 // For now, we assume a struct
77 // layout.
78 // THIS IS A PLANNED SIMPLIFICATION
79 // TO BE REFINED LATER.
80 struct AgenkTensor_Internal {
81     AgenkDataType data_type;
82     AgenkContentType content_type;
83     size_t element_size;
84     void* data;
85 };
86
87 if (fread(((struct AgenkTensor_
88     Internal*)text_tensor)->data, 1,
89     size, f) != (size_t)size) {
90     fclose(f);
91     tensor_free(text_tensor);
92     return NULL;
93 }
94
95 fclose(f);
96 return text_tensor;
97 }
```

Listing 55: An example JIT-  
compilable driver for plain text  
(src/senses/loaders/guest text loader.c)

### 9.2.3 Validation

The reliability of this complex, multi-stage process is guaranteed by a comprehensive test suite that validates each component's contract.

- **Core Logic (`test_sensation.c`):** This test (Listing 56) validates the baseline functionality, ensuring that a simple, built-in sense (like the text loader) can be correctly

found and executed by the ‘perceive\_input’ function, returning a valid tensor.

- **Autonomous**

## Build

**(test\_sensation\_build.c):** This test (Listing 57) performs a full, end-to-end integration test of the most complex feature. It triggers a perception for which no library exists, confirming that the builder is invoked, a third-party library is successfully compiled, and the final artifact is placed in the correct location.

- Persistence (**test sense registry.c**):

This test (Listing 58) provides the ultimate validation of the system's "long-term memory" for skills. It programmatically creates a complex blueprint, saves it, reloads it, and performs a deep, field-by-field comparison to prove that the storage and retrieval process is perfectly lossless.

```

1 /**
2 * @file test_sensation.c
3 * @brief Test suite for the AGENK
4 * Sensation Subsystem.
5 *
6 * This file validates the end-to-end
7 * functionality of the Sense Manager,
8 * including its ability to find
9 * blueprints, execute JIT drivers,
10 * and
11 * correctly return materialized tensors
12 *
13 * @version 1.0.0
14 * @author Ankush Yadav, Ankit Yadav,
15 * AuctaSapience
16 */
17
18 #include "../include/agenk_sensation.h"
19 #include "../../tensors/include/agenk_
20     tensor.h" // Now we only need the
21     public header
22 #include <stdio.h>
23 #include <stdlib.h>
24 #include <string.h>
25 C).
26 // --- Simple Testing Framework ---
27 static int g_tests_run = 0;
28 static int g_tests_passed = 0;
29 #define COLOR_GREEN "\x1B[32m"
30 #define COLOR_RED   "\x1B[31m"
31 #define COLOR_RESET "\x1B[0m"
32 #define TEST_SUITE_START(name) printf("
33     ---_Running_Test_Suite:_%s---\n",
34     name)
35 #define TEST_CASE(name) printf("  [TEST]
36     %s\n", name)
37 #define ASSERT(condition)
38
39 \
```

```

28 do {
29     \
30     g_tests_run++;
31     \
32     if (condition) {
33         \
34         g_tests_passed++;
35         \
36     } else {
37         \
38         fprintf(stderr, COLOR_RED "["
39             FAIL] %s:%d: Assertion "
40             failed: %s\n" \
41             COLOR_RESET, __FILE__
42             , __LINE__, # condition);
43         \
44     }
45     \
46 } while (0)
47 // --- Test Suite ---
48 void test_suite_sense_manager() {
49     TEST_SUITE_START("Sense_Manager");
50     TEST_CASE("Perceiving a simple text file");
51
52     const char* test_filename = "test_input.txt";
53     const char* test_content = "Hello, world of senses!";
54     FILE* f = fopen(test_filename, "w");
55     if (!f) { ASSERT(0 && "Failed to create test file"); return; }
56     fputs(test_content, f);
57     fclose(f);
58
59     AgenkTensor* result = perceive_input(
60         test_filename, "test_stream");
61
62     ASSERT(result != NULL);
63     if (result) {
64         ASSERT(tensor_get_ndim(result)
65             == 1);
66         ASSERT(tensor_get_data_type(
67             result) == AGENK_DATA_TYPE_UINT8);
68         ASSERT(tensor_get_content_type(
69             result) == AGENK_CONTENT_TYPE_UTF8_TEXT);
70
71         const size_t* shape = tensor_get_shape(result);
72         ASSERT(shape[0] == strlen(test_content));
73
74         // THE FIX: Use the new, safe
75         // public accessor function to
76         // get the data.
77         const void* data = tensor_get_data_ptr(result);
78         ASSERT(data != NULL);
79         if (data) {
80             \
81             ASSERT(memcmp(data, test_content,
82                 strlen(test_content)) == 0);
83             \
84             tensor_free(result);
85         }
86         remove(test_filename);
87     }
88 // --- Test Runner ---
89 int main() {
90     printf("=====\\n");
91     printf("Running AGENK Sensation "
92         Subsystem Tests\\n");
93     printf("=====\\n");
94     test_suite_sense_manager();
95
96     printf("\\n=====\\n");
97     printf("TEST SUMMARY\\n");
98     printf("=====\\n");
99     if (g_tests_passed == g_tests_run) {
100         printf(COLOR_GREEN "SUCCESS: All "
101             "%d tests passed.\\n" COLOR_
102             RESET, g_tests_run);
103         return 0;
104     } else {
105         fprintf(stderr, COLOR_RED "
106             FAILURE: %d out of %d tests "
107             failed.\\n" COLOR_RESET,
108             g_tests_run - g_tests_
109             passed, g_tests_run);
110     }
111     return 1;
112 }

```

Listing 56: The core logic validation suite (`src/senses/test/test_sensation.c`).

```

1 /**
2  * @file test_sensation_build.c
3  * @brief Test suite for the Sensation
4  * Subsystem's autonomous build
5  * capabilities.
6  *
7  * This performs a full end-to-end
8  * integration test of the Builder
9  * Module and
10 * the Sense Manager. It ensures the
11 * agent can correctly perceive a file
12 * type
13 * for which no pre-compiled library
14 * exists, triggering and completing
15 * the
16 * build process successfully.
17 *
18 * @version 1.0.0
19 * @author Ankush Yadav, Ankit Yadav,
20 * AuctaSapience
21 */
22
23 #include "../include/agenk_sensation.h"
24 #include "../../tensors/include/agenk_
25     tensor.h"
26 #include <stdio.h>
27 #include <stdlib.h>

```

```

18 #define ASSERT(condition) do { if (!(condition)) { \
19     fprintf(stderr, "[FAIL] %s:%d: Assertion failed: %s\n", \
20             __FILE__, __LINE__, #condition); \
21     exit(1); } } while (0)
22
23 #define INSTALL_DIR "runtime_libs_test"
24
25 int main() {
26     printf("=====\n");
27     printf("Running AGENK Autonomous Build Test\n");
28     printf("=====\n");
29
30     system("rm -rf " INSTALL_DIR);
31     printf("[TEST] Cleaned test installation directory.\n");
32
33     const char* test_filename = "dummy.jpg";
34     FILE* f = fopen(test_filename, "w");
35     if (f) {
36         fputs("dummy", f);
37         fclose(f);
38     }
39     printf("[TEST] Created dummy file: %s\n", test_filename);
40
41     printf("\n[TEST] Calling perceive_input, expecting build to trigger...\n");
42     AgenkTensor* result = perceive_input(test_filename, "build_test_stream");
43
44     printf("\n[TEST] Validating build results...\n");
45
46 // --- THE FIX: Check for the correct shared library file ---
47 const char* expected_lib_path;
48 #if defined(_WIN32) || defined(_WIN64)
49     expected_lib_path = INSTALL_DIR "/libjpeg.dll";
50 #elif defined(__APPLE__)
51     expected_lib_path = INSTALL_DIR "/libjpeg.dylib";
52 #else
53     expected_lib_path = INSTALL_DIR "/libjpeg.so";
54 #endif
55
56     FILE* lib_file = fopen(expected_lib_path, "rb");
57     ASSERT(lib_file != NULL);
58     if (lib_file) {
59         printf("[TEST] SUCCESS: Library artifact found at %s\n",
60               expected_lib_path);
61         fclose(lib_file);
62     }
63
64     ASSERT(result == NULL); // JIT script is still stubbed
65     tensor_free(result);
66
67     printf("[TEST] Cleaning up artifacts\n");
68     remove(test_filename);
69     system("rm -rf " INSTALL_DIR);
70
71     printf("\n[SUCCESS] Autonomous Build Test Passed.\n");
72 }

```

Listing 57: The autonomous build validation suite (`src/senses/test/test_sensation_build`)

```

1 /**
2  * @file test_sense_registry.c
3  * @brief Definitive test suite for the Sensation Subsystem's Blueprint Registry.
4  *
5  * This file provides a comprehensive, end-to-end integration test for the sense_registry module. It validates the entire lifecycle of a SenseObject blueprint, ensuring the system's long-term memory for skills is robust, reliable, and correct.
6  *
7  * The test performs the following critical sequence of operations:
8  * 1. **Initialization**: It starts and cleans the database environment.
9  * 2. **Creation**: It programmatically constructs a complex, multi-valued `SenseObject` blueprint in memory using *only* the public API functions.
10    This validates the blueprint builder API.
11  * 3. **Serialization & Persistence**: It calls `sense_registry_add_blueprint()` to serialize the in-memory C struct into the FlatBuffers format and persist it to the LMDB database.
12  * 4. **Retrieval & Deserialization**: It queries the database using the blueprint's unique ID to retrieve the raw FlatBuffers data and deserialize it back into a new in-memory `SenseObject` C struct.
13  * 5. **Symmetrical Verification**: It performs a deep, field-by-field comparison between the original `sense_out` object and the retrieved `sense_in` object, using *only* the public "getter" API. This provides absolute proof that the serialization-deserialization process is perfectly symmetrical and lossless.
14  * 6. **Memory Management**: It meticulously cleans up all allocated memory for both blueprint objects and shuts down the database, ensuring there
15
16
17
18
19
20
21
22
23
24
25
26
27

```

Listing 57: The autonomous build validation suite (`src/senses/test/test_sensation_build.c`).

```

    are
28 *      no memory leaks.
29 *
30 * Passing this test provides high
31 * confidence in the foundational
32 * layer of the
33 * agent's ability to learn and remember
34 * new perceptual skills.
35 *
36 * @version 1.0.0
37 * @author Ankush Yadav, Ankit Yadav,
38 *         AuctaSapience
39 */
40
41 #include "../include/agenk_sensation.h"
42 #include <stdio.h>
43 #include <stdlib.h>
44 #include <string.h>
45
46 #define ASSERT(c) do { if(!(c)) {
47     fprintf(stderr, "[FAIL] %s:%d: %s\n",
48         __FILE__, __LINE__, #c); exit(1); } }
49
50 #define DB_PATH_REGISTRY_TEST "./test_
51 registry_db"
52
53 int main() {
54     printf("---- Running Sense Registry
55           Tests (Full Lifecycle) ----\n");
56
57     system("rm -rf " DB_PATH_REGISTRY_
58           TEST);
59     ASSERT(sense_registry_init(DB_PATH_
60           Registry_TEST));
61
62     // 1. Create a complete blueprint
63     // programmatically
64     printf("[TEST] Creating 'sense_jpeg_
65           v1' blueprint in memory...\n");
66     SenseObject* sense_out = sense_
67         object_create("sense_jpeg_v1");
68     ASSERT(sense_out != NULL);
69     sense_object_set_source_uri(sense_
70         out, "https://github.com/libjpeg
71           -turbo/libjpeg-turbo.git");
72     sense_object_set_library_template(
73         sense_out, "build/libjpeg.so.62"
74     );
75     sense_object_set_installed_name(
76         sense_out, "libjpeg.so");
77     sense_object_set_jit_driver(sense_
78         out, "return_process_jpeg(path);
79     ");
80     sense_object_add_build_command(sense_
81         out, "cmake .. -B build");
82     sense_object_add_build_command(sense_
83         out, "make install");
84     sense_object_add_supported_extension(
85         sense_out, "jpg");
86     sense_object_add_supported_extension(
87         sense_out, "jpeg");
88     sense_object_add_build_tool(sense_
89         out, "cmake");
90     sense_object_add_build_tool(sense_
91         out, "make");
92
93     // 2. Add it to the database
94     printf("[TEST] Adding blueprint to
95           the database...\n");
96     ASSERT(sense_registry_add_blueprint(
97
98         sense_out));
99
100    // 3. Retrieve the full blueprint by
101        its ID
102    printf("[TEST] Retrieving blueprint
103        by ID from the database...\n");
104    SenseObject* sense_in = sense_
105        registry_get_by_id("sense_jpeg_
106        v1");
107    ASSERT(sense_in != NULL);
108
109    // 4. Deep, symmetrical comparison
110        using ONLY the public getter API
111    printf("[TEST] Performing deep
112        comparison...\n");
113    ASSERT(strcmp(sense_object_get_id(
114        sense_out), sense_object_get_id(
115        sense_in)) == 0);
116    ASSERT(strcmp(sense_object_get_
117        source_uri(sense_out), sense_
118        object_get_source_uri(sense_in))
119        == 0);
120    ASSERT(strcmp(sense_object_get_
121        installed_name(sense_out), sense_
122        object_get_installed_name(sense_
123        _in)) == 0);
124
125    ASSERT(sense_object_get_num_
126        extensions(sense_out) == sense_
127        object_get_num_extensions(sense_
128        _in));
129    ASSERT(strcmp(sense_object_get_
130        extension(sense_in, 0), "jpg")
131        == 0);
132
133    ASSERT(sense_object_get_num_build_
134        tools(sense_out) == sense_object_
135        _get_num_build_tools(sense_in));
136    ASSERT(strcmp(sense_object_get_build_
137        tool(sense_in, 0), "cmake") ==
138        0);
139
140    ASSERT(sense_object_get_num_build_
141        commands(sense_out) == sense_
142        object_get_num_build_commands(
143        sense_in));
144    ASSERT(strcmp(sense_object_get_build_
145        command(sense_in, 1), "make_
146        install") == 0);
147
148    // 5. Cleanup
149    printf("[TEST] Cleaning up...\n");
150    sense_object_free(sense_out);
151    sense_object_free(sense_in);
152
153    sense_registry_shutdown();
154    system("rm -rf " DB_PATH_REGISTRY_
155           TEST);
156
157    printf("\n[SUCCESS] Sense Registry
158           Test Passed.\n");
159    return 0;
160}

```

Listing 58: The definitive validation suite for the Sense Registry (`src/senses/test/test_sense_registry.c`).

This complete, validated subsystem provides a

powerful and extensible foundation for the agent’s interaction with the world. It moves beyond static, pre-compiled capabilities and enables a future where the agent can truly learn and adapt its most fundamental senses in response to new challenges and new forms of data.

## 10 The Screen Substrate: A Universal Interface for GUI Interaction

To achieve its goal of performing any task a human can, the agent requires a mechanism to perceive and act within the primary domain of human-computer interaction: the Graphical User Interface (GUI). While programmatic APIs offer structured data access, a vast portion of digital workflow, information, and functionality remains accessible only through visual interfaces. The Screen Substrate provides the agent with the senses of “sight” and “touch” for this digital world, enabling it to see, understand, and manipulate applications, websites, and operating systems just as a human user would.

This substrate is responsible for translating the complex, pixel-based reality of a screen into a structured, machine-readable format. It identifies and describes the essential components of an interface—such as buttons, text fields, and menus—along with their current state (e.g., ‘value’, ‘is\_enabled’, ‘is\_checked’) and the actions they afford (e.g., ‘clickable’, ‘editable’). This abstraction allows the agent’s reasoning core to operate on a semantic level, planning actions like ‘click(button\_id)’ rather than ‘click\_at\_pixel(x, y)’.

### 10.1 The Necessity of a Visual Interface

Integrating a screen perception and interaction module is not an optional feature but a foundational requirement for a truly generalist agent. The necessity stems from several key principles:

**Accessing the ”Long Tail” of Functionality:** An enormous number of digital tools, legacy systems, and websites do not offer programmatic APIs. For this ”long tail” of software, the GUI is the **only** available interface. A screen-aware agent can operate these systems, unlocking a vastly larger set of capabilities than an API-only agent could ever access.

**Grounding and Contextual Understanding:** GUIs provide essential context that is absent in raw data streams. The spatial arrangement of elements, the visual state of a component (e.g., a

grayed-out button indicating it is disabled), and the surrounding text all convey critical meaning. This visual context grounds the agent’s understanding, allowing it to interpret the state of an application and make more informed decisions.

**Emulating Human Workflows:** To automate or assist with human tasks, the agent must be able to replicate the fundamental actions humans perform: clicking buttons, filling forms, navigating menus, and dragging-and-dropping items. The Screen Substrate provides the direct bridge to emulating these elemental workflows, enabling the agent to follow instructions or learn by observing human demonstrations.

**Adaptability to Novel Systems:** An agent equipped with a robust screen interface is not limited to the specific software and systems it was explicitly trained on. It possesses the fundamental capability to approach a completely novel application and begin to understand its purpose and function by observing its visual representation and experimenting with its controls, much like a human user would.

### 10.2 Properties of the interface

So, what should be the components of screen and how they should be represented? We have to think of what properties does a visual interface have? One important property is that they are rectangular (square is also rectangle) in shape. So, how should we represent this rectangular in 1D text manner? In other words we need 1D equivalent of 2D rectangle. we can divide the screen into heirarchical rectangles or bounding boxes with each rectangle having difernet contents inside them and they can even contain other rectangles too so it is a nested structure. Also, each rectangle contain contents which can be text, image sometimes video and audio too. So, how to represent these contents?

So, As Pas As Our Current Understanding, We Have Two Problems -

1. Representation Of Rectangles In Heirarchical Manner For The Reasoning Model To Understand.
2. Representation Of Contents Inside The Bounding Boxes.

### 10.2.1 Representation Of Rectangles

## 11 Intelligence, Tools Screen Module: A Suite Of APIs For Using Screen and External Tools Intelligence

Lets talk about how we will call various intelligent models (reasoning, embedding, etc), external tools like web-search, social-media, weather, etc to use their intelligence, data, logic and capabilities for our various purposes that will be called by AGENK and optionally current screen to represent it visually in a domain specific language like in compact text form and snippets to call the icons, links, etc appearing on screen like to click them, type in them, scroll, etc like the keyboard, mouse input functionality to control the screen. So, this is the hub for AGENK to interact with external functionality and its interface. These are various sets of snippets containing external and internal API calls.

I guess we should also allow for choosing interface (operating system or kernel) and whether they are on or off. At a particular time any number of them can be off. By default they are off. And only the content of those screens will be shown which are "on" and working. So, these screens will be optional and based on setting. So, we need way to set them too (on/off and priority order i guess).

So, let's discuss on the intelligence (Reasoning, Embedding, etc) part. So, It will be containing calls to openAI, Gemini, Anthropic, DeepSeek, etc. As all these are from different domain and also there is not much similarity in them especially in naming and types of API calls. So, we need an extremely abstract way to categorise them which allows for further evolution. So, what it can be? So, first way to categorize is via domain name like [www.api.openai.com](http://www.api.openai.com), [www.api.deepseek.com](http://www.api.deepseek.com), etc. So, domain name is first way to divide them. Each domain will contain many different sets of api snippets. So, these sets will have a name called *setname* and they will contain inside many snippets of API calls. So, all these snippets will also have a name called *api\_name*.

So, we will have some compulsory functions that will need to be assigned to particular of these snippets via thier name, category and domain. Our main task is to design these compulsory functions like what they should be for most abstract design which allows for evolution. So, it can be like various types of calls to LLMs for text, vision, audio, document, etc as well as embedding models for vectors and similarity score. So, our task is to get most abstract way to divide these functions.

## 12 The AGENK Network Fabric: A Communication Layer for Collective Intelligence

The preceding substrates—Object, Sensation, and Tensor—define a powerful architecture for a *single, isolated agent*. They provide the foundations for memory, perception, and computation. However, the true potential of intelligence is not realized in isolation, but through interaction, collaboration, and scale. The AGENK vision is one of a **collective, distributed intelligence** where countless agents can share perceptions, learn from one another's experiences, and coordinate complex behaviors.

To enable this vision, a communication medium is required. A centralized, server-client architecture introduces a single point of failure and a performance bottleneck, which is antithetical to our goal of a resilient, scalable system. Therefore, we have designed and implemented the **AGENK Network Fabric**. This is not merely a passive substrate; it is the active, scalable nervous system that connects individual agents into a cohesive, intelligent whole.

### 12.1 Architectural Philosophy: The Decoupled Bus

The core design principle is that of a \*\*decoupled communication bus\*\*. The internal complexity of the server network is completely hidden from the clients. Likewise, the server network is agnostic to the content of the messages it routes, treating all client payloads as opaque, potentially encrypted blobs.

This architecture is designed to securely connect two distinct classes of clients:

1. **Agenk Clients (C Clients):** Lightweight, highly-portable C libraries designed to run on any platform, from embedded IoT devices to desktop applications.
2. **Backend Services (gRPC Clients):** Powerful, server-side components, such as the SaaS web application backend, which act as a bridge to the main user-facing application and its database.

The system is composed of four distinct, standalone applications that work in concert, as illustrated in Figure 3.

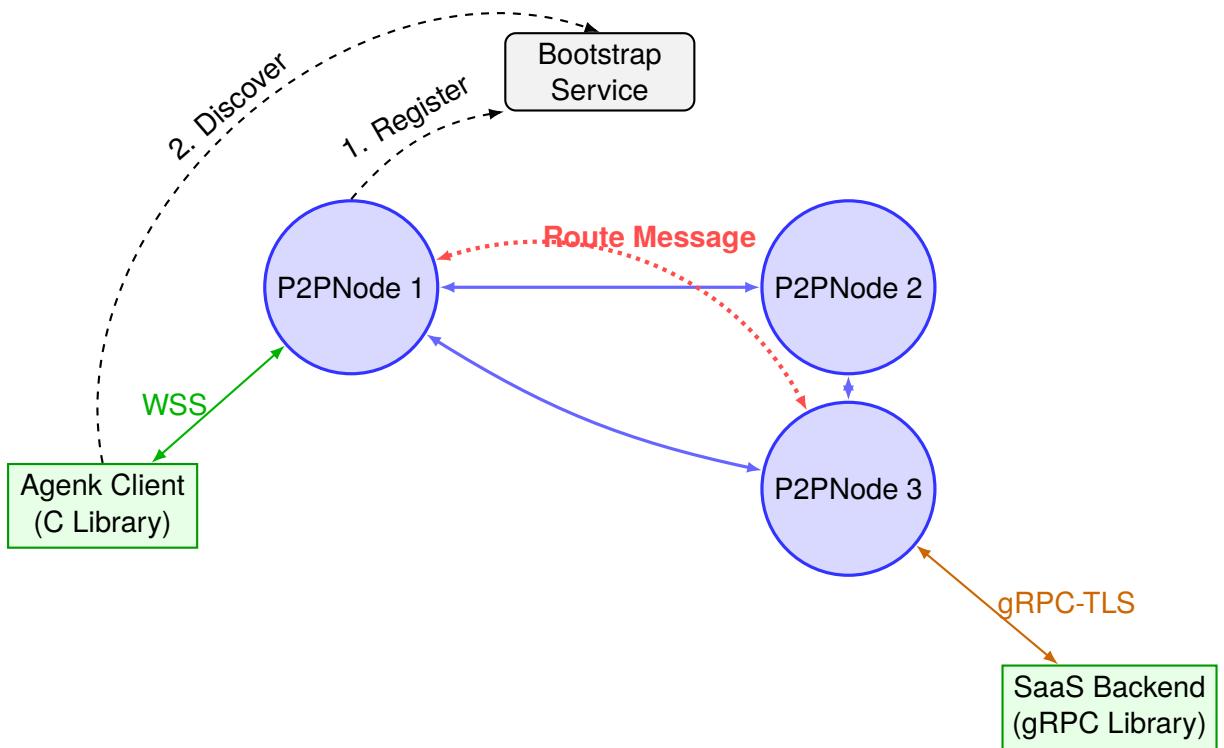


Figure 3: High-level architecture of the Network Fabric. P2P Nodes form a resilient mesh for internal communication. Clients discover and connect to a gateway node. A message from the SaaS Backend to an Agenk Client is routed through the P2P mesh from the entry gateway (Node 3) to the destination gateway (Node 1).

## 12.2 The Bootstrap Service: The Network’s Public Square

For a decentralized network to form, a paradoxical problem must be solved: how does the very first node find anyone to talk to? To solve this, the AGENK Network Fabric employs a **Bootstrap Service**. This is a simple, highly-available, and stateless public endpoint that acts as the network’s “public square” or “meeting point.” It does not participate in the P2P protocol itself; its sole purpose is to introduce newcomers to the existing network.

**Architectural Insight: Minimizing Centralization.** A key design goal was to minimize the role of any centralized component. The Bootstrap Service is therefore engineered to be as “dumb” and stateless as possible. It has no knowledge of the network’s topology, client sessions, or message content. It only maintains a volatile, in-memory list of P2P nodes that have recently announced their presence. This minimalist design ensures that even if the bootstrap service were to fail, the existing P2P network would continue to operate without interruption; only the ability for new nodes to join would be temporarily impaired. This makes it a

point of introduction, not a point of failure.

**The API Contract.** The service exposes two simple REST endpoints:

- **POST /register:** A live P2P node periodically calls this endpoint, sending its public IP address and port in a simple JSON body. The service stores this address along with a timestamp.
- **GET /nodes:** A new client or P2P node calls this endpoint. The service returns a small, randomized list of the currently active node addresses it has on record, providing the newcomer with their initial contacts to begin the Kademlia discovery process.

**Implementation with ‘cpp-httplib’.** To ensure the service is lightweight and portable, it is implemented as a single C++ source file using the header-only ‘cpp-httplib’ library. This avoids the need for heavy web frameworks. The build process, managed by CMake, uses the ‘FetchContent’ module to automatically download the library at compile time, creating a fully self-contained executable with zero runtime dependencies.

```

1 cmake_minimum_required(VERSION 3.15)
2 project(BootstrapService CXX)
3
4 set(CMAKE_CXX_STANDARD 17)
5 set(CMAKE_CXX_STANDARD_REQUIRED ON)
6
7 # Use FetchContent to get the httpplib
8     header at configure time
9 include(FetchContent)
10 FetchContent_Declare(
11     httpplib
12     GIT_REPOSITORY https://github.com/
13         yhirose/cpp-httplib.git
14     GIT_TAG          v0.15.3 # Pin to a
15         specific, stable version
16 )
17 FetchContent_MakeAvailable(httplib)
18
19 # Find PThreads for std::thread
20 find_package(Threads REQUIRED)
21
22 # --- Executable Target ---
23 add_executable(bootstrap_server main.cpp
24 )
25
26 # Link the executable against its
27     dependencies
28 target_link_libraries(bootstrap_server
29     PRIVATE
30     Threads::Threads
31     httpplib # This is the target created
32         by FetchContent
33 )

```

Listing 59: The self-contained ‘CMakeLists.txt’ for the Bootstrap Service.

**Core Logic: Thread-Safe State and Garbage Collection.** The service’s core logic, shown in Listing 60, is designed for robustness. The in-memory list of active nodes is protected by a ‘std::mutex’ to ensure that concurrent requests from multiple P2P nodes do not corrupt the state. Furthermore, a detached background thread acts as a ”garbage collector.” It periodically scans the list and removes any node that has not registered within a defined timeout period (e.g., 90 seconds). This ensures that the list remains fresh and does not return addresses of nodes that have gone offline.

```

1 #define CPPHTTPLIB_OPENSSL_SUPPORT
2 #include "httpplib.h"
3 #include <iostream>
4 #include <vector>
5 #include <string>
6 #include <mutex>
7 #include <chrono>
8 #include <thread>
9 #include <algorithm>
10 #include <random>
11
12 struct NodeInfo {
13     std::string address;
14     std::chrono::steady_clock::time_
15         point last_seen;

```

```

15     };
16
17 // --- Global State (Thread-Safe) ---
18 std::vector<NodeInfo> active_nodes;
19 std::mutex nodes_mutex;
20 const int STALE_NODE_TIMEOUT_SECONDS =
21     90;
22
23 // Background thread to remove stale
24     nodes
25 void garbage_collector() {
26     while (true) {
27         std::this_thread::sleep_for(std::
28             ::chrono::seconds(30));
29         std::lock_guard<std::mutex> lock(
30             nodes_mutex);
31         auto now = std::chrono::steady_
32             clock::now();
33
34         active_nodes.erase(
35             std::remove_if(active_nodes.
36                 begin(), active_nodes.
37                 end(),
38             [&](const NodeInfo& node)
39             ) {
40                 auto elapsed = std::
41                     chrono::duration_
42                         ::cast<std:::
43                             chrono::seconds
44                         >(now - node.
45                             last_seen).count
46                         ();
47                 if (elapsed > STALE_
48                     NODE_TIMEOUT_
49                     SECONDS) {
50                     std::cout << "[
51                         GC] Removing
52                         stale node "
53                         << node.
54                         address <<
55                         std::endl;
56                     return true;
57                 }
58                 return false;
59             },
60             active_nodes.end()
61         );
62     }
63
64     int main() {
65         httpplib::Server svr;
66         std::thread gc_thread(garbage_
67             collector);
68         gc_thread.detach();
69
70         svr.Get("/nodes", [](const httpplib::
71             Request&, httpplib::Response& res
72             ) {
73             std::vector<std::string> nodes_
74                 to_send;
75             {
76                 std::lock_guard<std::mutex>
77                     lock(nodes_mutex);
78                 for(const auto& node :
79                     active_nodes) {
80                     nodes_to_send.push_back(
81                         node.address);
82                 }
83             }
84         });
85     }

```

```

57
58     // Randomize the list to
59     // distribute client load
60     std::random_device rd;
61     std::mt19937 g(rd());
62     std::shuffle(nodes_to_send.begin()
63                  (), nodes_to_send.end(), g);
64
65     std::string body = "[";
66     for (size_t i = 0; i < nodes_to_
67         send.size(); ++i) {
68         body += "\\" + nodes_to_send
69         [i] + "\\"";
70         if (i < nodes_to_send.size()
71             - 1) body += ",\u0022";
72     }
73     body += "]";
74
75     res.set_content(body, "application/json");
76 };
77
78 svr.Post("/register", [](const
79     httpplib::Request& req, httpplib::
80     Response& res) {
81     std::string address_key = "\u0022
82     address\u0022:\u0022";
83     auto pos = req.body.find(address
84     key);
85     if (pos == std::string::npos) {
86         res.status = 400;
87         res.set_content("Invalid
88         JSON", "text/plain");
89         return;
90     }
91     auto start = pos + address_key.
92     length();
93     auto end = req.body.find("\u0022",
94     start);
95     if (end == std::string::npos) {
96         res.status = 400;
97         return;
98     }
99     std::string address = req.body.
100    substr(start, end - start);
101
102    {
103        std::lock_guard<std::mutex>
104        lock(nodes_mutex);
105        auto it = std::find_if(
106            active_nodes.begin(),
107            active_nodes.end(),
108            [&](const NodeInfo& node
109                 ) { return node.
110                     address == address;
111                 });
112
113        if (it != active_nodes.end())
114        {
115            it->last_seen = std::
116            chrono::steady_clock
117            ::now();
118        } else {
119            active_nodes.push_back({
120                address, std::chrono
121                ::steady_clock::now
122                ()});
123        }
124    }
125
126    res.set_content("{\"status\":\""
127      "success\"}", "application/
128      json");
129
130    int port = 9090;
131    std::cout << "Bootstrap\u002fserver\u002f
132      listening\u002fon\u002fport\u002f" << port << "
133      ..." << std::endl;
134    svr.listen("0.0.0.0", port);
135    return 0;
136
137 }#define CPPHTTPPLIB_OPENSSL_SUPPORT //  

138 // Enable this for future HTTPS support
139 #include "httpplib.h"  

140 #include <iostream>  

141 #include <vector>  

142 #include <string>  

143 #include <mutex>  

144 #include <chrono>  

145 #include <thread>  

146
147 // A simple struct to hold node info and
148 // its last check-in time
149 struct NodeInfo {
150     std::string address;
151     std::chrono::steady_clock::time_
152     point last_seen;
153 };
154
155 // --- Global State (Thread-Safe) ---
156 std::vector<NodeInfo> active_nodes;
157 std::mutex nodes_mutex;
158 const int STALE_NODE_TIMEOUT_SECONDS =
159     90; // If a node hasn't checked in
160     for 90s, remove it
161
162 // Background thread to remove stale
163 // nodes
164 void garbage_collector() {
165     while (true) {
166         std::this_thread::sleep_for(std
167             ::chrono::seconds(30));
168
169         std::lock_guard<std::mutex> lock
170             (nodes_mutex);
171         auto now = std::chrono::steady_
172             clock::now();
173
174         active_nodes.erase(
175             std::remove_if(active_nodes.
176                 begin(), active_nodes.
177                 end(),
178             [&](const NodeInfo& node
179                  ) {
180                 auto elapsed = std::
181                     chrono::duration
182                     _cast<std::
183                     chrono::seconds
184                     >(now - node.
185                     last_seen).count
186                     ();
187                 if (elapsed > STALE_
188                     NODE_TIMEOUT_
189                     SECONDS) {
190                     std::cout << "
191                         Garbage\u002f
192                         Collector:\u002f
193                         Removing\u002f
194                         stale\u002fnode\u002f"
195                     << node.
196                     address;
197                 }
198             }));
199
200         std::this_thread::sleep_for(std
201             ::chrono::seconds(1));
202     }
203 }

```

```

142                                     address <<
143                                     std::endl;
144                                     return true;
145                               }
146                               active_nodes.end()
147                           );
148                         }
149 }
150
151 int main() {
152   httpplib::Server svr;
153
154   // Start the background thread for
155   // cleaning up stale nodes
156   std::thread gc_thread(garbage_
157   collector);
158   gc_thread.detach(); // Let it run
159   // independently
160
161   // Endpoint #1: GET /nodes
162   // Returns a JSON array of active
163   // P2P node addresses.
164   svr.Get("/nodes", [](const httpplib::
165     Request&, httpplib::Response& res
166   ) {
167     std::lock_guard<std::mutex> lock
168       (nodes_mutex);
169
170     // Build a simple JSON array
171     string: ["addr1", "addr2",
172     ...]
173     std::string body = "[";
174     for (size_t i = 0; i < active_
175       nodes.size(); ++i) {
176       body += "\\" + active_nodes[
177         i].address + "\\"";
178       if (i < active_nodes.size()
179         - 1) {
180         body += ",\u0022";
181       }
182     }
183     body += "\u0022";
184
185     res.set_content(body, "
186     application/json");
187     std::cout << "GET\u002fnodes:\u0022
188     Responded\u002du" << active_
189     nodes.size() << "\u002fnodes.\u0022
190     std::endl;
191   });
192
193   // Endpoint #2: POST /register
194   // A P2P node calls this to announce
195   // its presence.
196   // Expects a JSON body like: {""
197   //   address": "ip:port"}
198   svr.Post("/register", [](const
199     httpplib::Request& req, httpplib::
200     Response& res) {
201     // A simple way to parse JSON
202     // without a full library
203     std::string address_key = "\u0022
204     address\u0022:\u0022";
205     auto pos = req.body.find(address
206     key);
207     if (pos == std::string::npos) {
208       res.status = 400; // Bad
209       Request
210
211     address <<
212     std::endl;
213     return true;
214   }
215
216   res.set_content("Invalid\u0020
217   JSON\u0020body.\u0020Expected\u0020\b
218   address\b:\b...\b}\b", "
219   text/plain");
220   return;
221 }
222
223 auto start = pos + address_key.
224   length();
225 auto end = req.body.find("\u0022",
226   start);
227 if (end == std::string::npos) {
228   res.status = 400;
229   return;
230 }
231
232 std::string address = req.body.
233   substr(start, end - start);
234
235 {
236   std::lock_guard<std::mutex>
237     lock(nodes_mutex);
238   // Check if node already
239   // exists and just update
240   // its timestamp
241   auto it = std::find_if(
242     active_nodes.begin(),
243     active_nodes.end(),
244     [&](const NodeInfo& node
245       ){ return node.
246       address == address;
247     });
248
249   if (it != active_nodes.end())
250     it->last_seen = std::
251       chrono::steady_clock
252       ::now();
253   } else {
254     active_nodes.push_back({
255       address, std::chrono
256       ::steady_clock::now
257       ()});
258   }
259 }
260
261 res.set_content("{\"status\":\"\u0022
262   success\u0022}\u0022, "application/
263   json");
264 std::cout << "POST\u002fregister:\u0022
265   Registered\u002fupdated\u002fnode\u0022
266   << address << std::endl;
267
268 std::cout << "Bootstrap\u002fserver\u0022
269   starting\u002fon\u002fport\u002f9090..." << std
270   ::endl;
271 svr.listen("0.0.0.0", 9090);
272
273 return 0;
274 }
```

**Listing 60:** The complete implementation of the standalone Bootstrap Service ('bootstrap\_service/main.cpp').

**Deployment on Google Cloud Run.** The design of this service—a stateless, single-process, con-

tainerizable web server—makes it a **perfect candidate for deployment on a serverless platform like Google Cloud Run**. Cloud Run automatically handles scalability, availability, and request balancing. To deploy, we would simply package the compiled ‘bootstrap\_server’ executable into a minimal ‘Dockerfile’, push the resulting image to Google Artifact Registry, and create a Cloud Run service from that image. This provides a highly cost-effective and zero-maintenance solution for the network’s most critical entry point.

### 12.3 The P2P Node: The Fundamental Unit of the Fabric

The ‘p2p\_node’ is the workhorse of the system. Each instance is a multi-threaded C++ server performing three roles: a P2P Peer using an internal gRPC service, a WebSocket Gateway for C clients, and a gRPC Gateway for backend clients.

**Insight: Dynamic Discovery and Registration.** A key feature is its ability to interact with the Bootstrap Service. Upon startup, if no static peer list is provided, the node’s ‘joinNetwork’ function makes an HTTP GET request to the bootstrap server to dynamically discover the network. In parallel, a separate ‘registrationLoop’ thread is launched, which periodically sends a POST request to the bootstrap server to announce its own presence, ensuring the network’s list of active members is always up-to-date.

```

1 // From: src/node.cpp
2
3 void Node::joinNetwork() {
4     // Step 1: Discover peers if no
5     // static list is provided.
6     if (m_bootstrap_nodes.empty()) {
7         std::cout << "Contacting"
8         "bootstrapserver..." << std
9         ::endl;
10        httpplib::Client cli(m_bootstrap_
11        url);
12        if (auto res = cli.Get("/nodes"))
13        ) {
14            if (res->status == 200) {
15                m_bootstrap_nodes =
16                    parseNodeList(res->
17                    body);
18            }
19        }
20
21        // Step 2: Perform Kademlia lookup
22        // using the discovered peers.
23        const NodeID target_id = m_self_
24        contact.id;
25        // ... Kademlia iterative lookup
26        logic ...
27 }
```

```

19 void Node::registrationLoop() {
20     httpplib::Client cli(m_bootstrap_url)
21     ;
22     while (!m_stop_heartbeat) {
23         std::string body = "{\"address":
24         "\":\" + m_self_contact.
25         address + "\"}";
26         cli.Post("/register", body, "application/json");
27         // Wait for 60 seconds before re-
28         // registering
29         for (int i = 0; i < 60 && !m_
30         stop_heartbeat; ++i) {
31             std::this_thread::sleep_for(
32                 std::chrono::seconds(1))
33         }
34     }
35 }
```

Listing 61: A snippet from the P2P Node’s dynamic discovery logic (‘node.cpp’).

### 12.4 The Agen Client: A Portable C Library

To maximize portability, the primary client is a minimal-dependency C99 library. It is designed to be fully self-contained by vendoring its own dependencies (‘libwebsockets’, ‘protobuf-c’), which are compiled statically.

**Insight: Asynchronous Design with a Thread-Safe Send Queue.** The library is engineered for high performance and integration into complex, multi-threaded applications. It spawns its own background thread to manage the ‘libwebsockets’ event loop, ensuring that network I/O never blocks the main application. To handle scenarios where the application may generate messages faster than the network can send them, we implemented a thread-safe, linked-list send queue, as shown in Listing 62. The ‘p2pclient\_send\_message’ function is a lightweight operation that

```

1 // From: c_client/src/p2p_client.c
2
3 int p2p_client_send_message(/*...*/) {
4     // ... (protobuf serialization) ...
5
6     struct p2p_queued_message* new_msg =
7         malloc(sizeof(*new_msg));
8     new_msg->payload = buffer; // The
9     serialized data
10    new_msg->len = packed_len;
11    new_msg->next = NULL;
12
13    // Safely add the new message to the
14    // tail of the queue
15    pthread_mutex_lock(&client->lock);
16    if (client->tx_queue_tail) {
17        client->tx_queue_tail->next =
18            new_msg;
19    } else {
```

```

16     client->tx_queue_head = new_msg;
17 }
18 client->tx_queue_tail = new_msg;
19 pthread_mutex_unlock(&client->lock);
20
21 // Request a writable callback to
22 // service the queue
22 if (client->wsi) {
23     lws_callback_on_writable(client
24         ->wsi);
25 }
26 return 0;
}

```

Listing 62: The thread-safe send queue from the C client library ('`p2pclient.c`').

## 12.5 The Backend Bridge: The gRPC Client Library

To allow the main SaaS web application to interact with the P2P network, we provide a C++ gRPC client library. This component acts as a bridge, connecting to a P2P node's gRPC gateway.

**Insight: Asynchronous Streaming for Incoming Messages.** A key design feature is the use of a persistent, server-streaming RPC for receiving messages. The 'connect' method spawns a dedicated background thread that calls the 'CreateSession' RPC. This RPC remains open for the lifetime of the connection. The 'listenerThreadMain' function, shown in Listing ??, spawns a 'while(stream->Read(envelope))' loop, blocking until the server pushes a new message down the stream. This is a highly efficient, push-based architecture that eliminates the need for the client to poll for new messages, ensuring minimal latency for backend-to-client communication.

```

1 // From: grpc_client/src/p2p_grpc_client
2 .cpp
3
3 void P2PGrpcClientImpl::
4     listenerThreadMain() {
5     while (m_is_running) {
6         // Establish the persistent
7         // server stream
8         m_stream_context = std::make_
9             unique<grpc::ClientContext
10            >();
11         p2p::ClientLogin request;
12         request.set_client_id(m_client_
13             _id);

14         std::unique_ptr<grpc::
15             ClientReader<p2p::
16             ServerEnvelope>> stream =
17             m_stub->CreateSession(m_
18                 stream_context.get(),
19                 request);
20
21     }
22 }

```

```

13
13     if (m_on_status) m_on_status("
14         CONNECTED");

15     // Block here, waiting for the
16     // server to push messages
17     p2p::ServerEnvelope envelope;
18     while (m_is_running && stream->
19         Read(&envelope)) {
20         if (envelope.has_forwarded_
21             message()) {
22             const auto& msg =
23                 envelope.forwarded_
24                 message();
25             if (m_on_message) {
26                 m_on_message(msg.
27                     sender_id(), msg
28                     .content());
29             }
30         }
31     }
32
33     // If the loop breaks, the
34     // connection was lost.
35     if (m_is_running) {
36         if (m_on_status) m_on_status
37             ("CONNECTION_LOST");
38         std::this_thread::sleep_for(
39             std::chrono::seconds(5));
40         ;
41         if (m_on_status) m_on_status
42             ("RECONNECTING");
43     }
44 }

```

Listing 63: The asynchronous listener loop from the gRPC client library ('`p2p_grpc_client.cpp`').

## 12.6 The Management Frontend

To facilitate user interaction, a web-based dashboard is provided. Built with Next.js, this frontend communicates with the main SaaS web application backend. It allows users to log in, view their registered Agenk Clients, and generate new 'ClientID's and associated API Keys for provisioning new devices. The frontend itself does not interact directly with the P2P network; it is a user interface for the SaaS control plane. This clean separation ensures that the core P2P network remains a pure infrastructure layer, completely decoupled from the specific business logic of the user-facing application.

## 13 The REASON For All Of This - ACTION and KNOWLEDGE

Why are we doing all of these? To do some logical actions in real-world and generate knowledge of both internal and external state. So, we have to do action or generate knowledge. How to do "ACTION" or generate "KNOWLEDGE"? The answer is to "REASON" in language and vector embeddings.

Reasoning is the bedrock of the intelligence of an agent. Reasoning can be done in both language and vector embedding space. So, there are many properties of reasoning, one is amplitude which means number of previous variables and logic applied and result obtained to keep in mind and another is possibilities which are a set of statements which activate on satisfaction of particular logic from that set. The difference between logic and beliefs is that logic is represented in a particular format like a particular form of logic object while beliefs shape logic. So, inside every reasoning object there will be an array of logic objects. So, how logic is structured? It is basically the variables required to come to conclusion in this logic which are represented as either Call to other actions via id, name and description and they are represented in a map like structure, conditions on the variables which are required to do this action in form of code, it is the main semantic meaning of this reasoning object connected to an action which is in form of code and what to do when that condition satisfies which basically means what action take on the satisfaction of that semantic condition which is also in form of code.

So, we can also call REASON as interaction between objects. As when a particular action will be called then a specific REASON object associated with it is called. So, REASON is the reaction to call of a particular action. So, interaction happens when an action is called via some space. It is composed of current and previous experience knowledge, action knowledge,

So, suppose a new action is called and a experience knowledge object id is given to it as an input then

So, logic object is combination of variables required (map of variables and how to call them via id, name, description, etc. In first call we can choose the preferred type of space (id, name, description) to search based on our knowledge in context or beliefs about that variable then after first call they are connected via id too so from now on every call is routed through that.), conditions on variables to activate this logic (activation logic in form of code), and action to take on satisfaction of our logical condition on variables required in form of code.

So, actions can be done in a function/class like manner connected in many spaces like id, name, description, etc. So, actions can be searched in every space. This means that in action can be searched in many ways. Also, knowledge can be

represented in object like manner in many types like memory, experience, logic, etc. This means that knowledge can be represented in many ways. Also, we have searching in knowledge space via description/semantics, id, etc. So,

So, there are objects and classes which are in different searchable spaces with different properties and can be represented in many types. An object(knowledge) or class(action) must be uniquely identified by their space and type. So, action takes knowledge of a particular type. What is the type of knowledge object? Type is something that is the structure of knowledge object. So, let us define all the types of knowledge object. These are memory, experience, and logic.

### 13.1 How to identify an Object

The first question that arises when thinking about location of object is what is identifier of that and other objects. This is a tricky question because the identifier should be unique and easy to search. So, there are some questions or points arise. See, identifier will work in a particular space that is to say there is an algorithm to that identifier in which reaching to that identifier is guaranteed. We can search by direct key, we can search by name, we can search by its description. So, I guess both three have thier advantages like key is for secure connection which is extremely fast and is not used in reasoning or like can be known from outside, name can be used for fast cheap name based calls which are relative to each object in that space. And description can be used for searching in semantic space and intelligently guessing the accurate result and creating if not present and then connecting the both objects via edge for name based calls and connect their call through id too so that id is used in most calls behind the back.

## 14 The Era Of Intelligence

If we think clearly we can use the logic we have learned from various actions we are doing. In a sense we can gather intelligence from the actions which can be generalized across various actions. So, if we think clearly the abstract logic that is being used in one action will also being used across many actions as AGENK is doing work in a particular domain like suppose in software, executive, labour, politician, etc. Like AGENK works in a particular role only like jobs in a particular industry. So, Intelligence can be fast tracked based on

top logics for particular actions we don't know. So, different logic can be used as tokens for learning.

I guess there should be a mechanism to guess and improve too. Like how we guess logic and see what works.

Let's develop logic module for our AGENK system. So, logic module is made up of logic registry which stores logic objects.

## 14.1 All location is relative!

Location is something that cannot be absolute. Can you think of a central point that should be the center of everything? If there is one, then it should not exist in the space of objects, at least not as a center, because it would create inefficiencies and bottlenecks, reduce algorithmic speed and abstraction, and limit the possibility of certain forms of evolution or specialization. In the most abstract scenario, there should not be any particular center, and all objects should be located and searched for in reference to one another.

So, we define location as a datapoint inside object. The type of data would be array and would contain

## 15 Engines

To produce a PDF file, pdf $\text{\LaTeX}$  is strongly recommended (over original  $\text{\LaTeX}$  plus dvips+ps2pdf or dvipdf). The style file `acl.sty` can also be used with lual $\text{\LaTeX}$  and Xe $\text{\LaTeX}$ , which are especially suitable for text in non-Latin scripts. The file `acl_lualatex.tex` in this repository provides an example of how to use `acl.sty` with either lual $\text{\LaTeX}$  or Xe $\text{\LaTeX}$ .

## 16 Preamble

The first line of the file must be

```
\documentclass[11pt]{article}
```

To load the style file in the review version:

```
\usepackage[review]{acl}
```

For the final version, omit the `review` option:

```
\usepackage{acl}
```

To use Times Roman, put the following in the preamble:

```
\usepackage{times}
```

Command	Output	Command	Output
{\"a}	ä	{\c c}	ç
{\^e}	ê	{\u g}	ă
{\`i}	í	{\l }	ł
{\.\I}	Í	{\~n}	ñ
{\o}	ø	{\H o}	ö
{\'u}	ú	{\v r}	ŕ
{\aa}	å	{\ss}	ß

Table 1: Example commands for accented characters, to be used in, e.g., Bib $\text{\TeX}$  entries.

(Alternatives like txfonts or newtx are also acceptable.)

Please see the  $\text{\LaTeX}$  source of this document for comments on other packages that may be useful.

Set the title and author using `\title` and `\author`. Within the author list, format multiple authors using `\and` and `\And` and `\AND`; please see the  $\text{\LaTeX}$  source for examples.

By default, the box containing the title and author names is set to the minimum of 5 cm. If you need more space, include the following in the preamble:

```
\setlength{\titlebox}{<dim>}
```

where `<dim>` is replaced with a length. Do not set this length smaller than 5 cm.

## 17 Document Body

### 17.1 Footnotes

Footnotes are inserted with the `\footnote` command.<sup>1</sup>

### 17.2 Tables and figures

See Table 1 for an example of a table and its caption. **Do not override the default caption sizes.**

As much as possible, fonts in figures should conform to the document fonts. See Figure 4 for an example of a figure and its caption.

Using the `graphicx` package graphics files can be included within figure environment at an appropriate point within the text. The `graphicx` package supports various optional arguments to control the appearance of the figure. You must include it explicitly in the  $\text{\LaTeX}$  preamble (after the `\documentclass` declaration and before `\begin{document}`) using `\usepackage{graphicx}`.

---

<sup>1</sup>This is a footnote.

# Golden ratio

(Original size: 32.361×200 bp)

Figure 4: A figure with a caption that runs for more than one line. Example image is usually available through the `mwe` package without even mentioning it in the preamble.

## 17.3 Hyperlinks

Users of older versions of L<sup>A</sup>T<sub>E</sub>X may encounter the following error during compilation:

```
\pdfendlink ended up in different
nesting level than \pdfstartlink.
```

This happens when pdfL<sup>A</sup>T<sub>E</sub>X is used and a citation splits across a page boundary. The best way to fix this is to upgrade L<sup>A</sup>T<sub>E</sub>X to 2018-12-01 or later.

## 17.4 Citations

Table 2 shows the syntax supported by the style files. We encourage you to use the natbib styles. You can use the command `\citet` (cite in text) to get “author (year)” citations, like this citation to a paper by ?. You can use the command `\citep` (cite in parentheses) to get “(author, year)” citations (?). You can use the command `\citealp` (alternative cite without parentheses) to get “author, year” citations, which is useful for using citations within parentheses (e.g. ?).

A possessive citation can be made with the command `\citep*{}`. This is not a standard natbib command, so it is generally not compatible with other style files.

## 17.5 References

The L<sup>A</sup>T<sub>E</sub>X and BibT<sub>E</sub>X style files provided roughly follow the American Psychological Association format. If your own bib file is named `custom.bib`, then placing the following before any appendices in your L<sup>A</sup>T<sub>E</sub>X file will generate the references section for you:

```
\bibliography{custom}
```

You can obtain the complete ACL Anthology as a BibT<sub>E</sub>X file from <https://aclweb.org/anthology/anthology.bib.gz>. To include both the Anthology and your own .bib file, use the following instead of the above.

```
\bibliography{anthology,custom}
```

Please see Section 18 for information on preparing BibT<sub>E</sub>X files.

## 17.6 Equations

An example equation is shown below:

$$A = \pi r^2 \quad (4)$$

Labels for equation numbers, sections, subsections, figures and tables are all defined with the `\label{label}` command and cross references to them are made with the `\ref{label}` command.

This is an example cross-reference to Equation 4.

## 17.7 Appendices

Use `\appendix` before any appendix section to switch the section numbering over to letters. See Appendix A for an example.

## 18 BibT<sub>E</sub>X Files

Unicode cannot be used in BibT<sub>E</sub>X entries, and some ways of typing special characters can disrupt BibT<sub>E</sub>X’s alphabetization. The recommended way of typing special characters is shown in Table 1.

Please ensure that BibT<sub>E</sub>X records contain DOIs or URLs when possible, and for all the ACL materials that you reference. Use the `doi` field for DOIs and the `url` field for URLs. If a BibT<sub>E</sub>X entry has a URL or DOI field, the paper title in the references section will appear as a hyperlink to the paper, using the hyperref L<sup>A</sup>T<sub>E</sub>X package.

## Limitations

This document does not cover the content requirements for ACL or any other specific venue. Check the author instructions for information on maximum page lengths, the required “Limitations” section, and so on.

## Acknowledgments

This document has been adapted by Steven Bethard, Ryan Cotterell and Rui Yan from the instructions for earlier ACL and NAACL proceedings, including those for ACL 2019 by Douwe

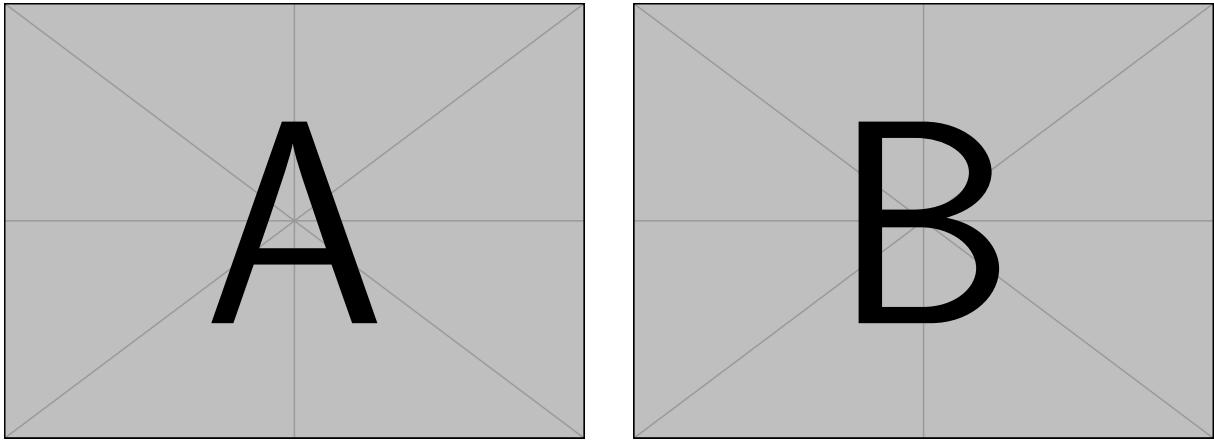


Figure 5: A minimal working example to demonstrate how to place two images side-by-side.

<b>Output</b>	<b>natbib command</b>	<b>ACL only command</b>
(?)	\citet	
?	\citealp	
?	\citet	
(?)	\citeyearpar	
?'s (?)		\citeposs

Table 2: Citation commands supported by the style file. The style is based on the natbib package and supports all natbib citation commands. It also supports commands defined in previous ACL style files for compatibility.

Kiela and Ivan Vulić, NAACL 2019 by Stephanie Lukin and Alla Roskovskaya, ACL 2018 by Shay Cohen, Kevin Gimpel, and Wei Lu, NAACL 2018 by Margaret Mitchell and Stephanie Lukin, BibT<sub>E</sub>X suggestions for (NA)ACL 2017/2018 from Jason Eisner, ACL 2017 by Dan Gildea and Min-Yen Kan, NAACL 2017 by Margaret Mitchell, ACL 2012 by Maggie Li and Michael White, ACL 2010 by Jing-Shin Chang and Philipp Koehn, ACL 2008 by Johanna D. Moore, Simone Teufel, James Allan, and Sadaoki Furui, ACL 2005 by Hwee Tou Ng and Kemal Oflazer, ACL 2002 by Eugene Charniak and Dekang Lin, and earlier ACL and EACL formats written by several people, including John Chen, Henry S. Thompson and Donald Walker. Additional elements were taken from the formatting instructions of the *International Joint Conference on Artificial Intelligence* and the *Conference on Computer Vision and Pattern Recognition*.

## A Example Appendix

This is an appendix.