

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.

```

```

17 * It provides a complete, self-
18 * contained system for creating and
19 * managing objects in
20 * a pure C99 environment. It achieves
21 * object-oriented capabilities like
22 * inheritance
23 * (via composition and VTables),
24 * polymorphism, and managed
25 * lifecycles without
26 * relying on C++ or any non-standard
27 * compiler extensions.
28 *
29 * The file is divided into three major
30 * sections:
31 * 1. **Public API:** A set of
32 * functions (`CLASS_INIT`, `CREATE_
33 * GROUP`) and macros
34 * (`NEW_ROOT`, `CALL`, `GET`, `
35 * DESTROY`) that form the public-
36 * facing interface
37 * for interacting with the
38 * framework.
39 * 2. **Domain-Specific Language (DSL)
40 * ** A collection of powerful
41 * preprocessor
42 * macros (`OBJECT_START`, `METHOD`,
43 * `CONSTRUCTOR`, etc.) that allow
44 * developers
45 * to declare new "classes" in a
46 * structured, readable way. This DSL
47 * abstracts
48 * away the complex C boilerplate
49 * required for manual VTable
50 * management and
51 * data structure setup.
52 * 3. **Internal Helper Prototypes:**
53 * Forward declarations for the
54 * private backend
55 * functions that power the
56 * framework (e.g., registry and
57 * database interactions).
58 *
59 * @section design_philosophy Design
60 * Philosophy
61 * - **Abstraction & Specialization:**
62 * The framework is built on the idea
63 * that
64 * a generic `Object` (the
65 * abstraction) can be specialized
66 * into concrete types
67 * (like `Task` or `FileParser`).
68 * The DSL is the mechanism for this
69 * specialization.
70 * - **Polymorphism in C:** The core
71 * of the object-oriented design is
72 * the Virtual
73 * Table (VTable). Every object
74 * contains a pointer to a table of
75 * its methods.
76 * This allows generic code to
77 * operate on an `Object*` and call
78 * its methods
79 * without knowing the object's
80 * concrete type at compile time. This
81 * is achieved
82 * safely through carefully designed
83 * macros and struct inheritance.
84 * - **Strict C99 Compliance:** The
85 * framework avoids all non-standard
86 * compiler
87 * extensions (like `typeof`) to
88 * ensure maximum portability. It can
89 * be compiled
90 * with any standard C99 compiler,
91 * from GCC to TinyCC.
92 * - **Managed Lifecycle:** Memory
93 * management in large C projects is
94 * notoriously
95 * difficult. The framework tackles
96 * this head-on with a group-based
97 * lifecycle
98 * system. Objects are created
99 * within a "group," and a single call
100 * to
101 * `DESTROY_BY_GROUP` can reliably
102 * clean up all memory associated with
103 * a specific
104 * task, preventing leaks.
105 * - **Safety and Robustness:** Macros
106 * are designed to be "NULL-safe,"
107 * meaning they
108 * will not crash if passed a `NULL`
109 * object pointer. This reduces the
110 * amount of
111 * defensive boilerplate code
112 * developers need to write.
113 */
114
115 #include <stdio.h>
116 #include <stdlib.h>
117 #include <string.h>
118 #include <stdbool.h>
119 #include <assert.h>
120 #include <stdint.h>
121
122 // Forward declare the generic Object
123 handle.
124 typedef struct AgenkObject Object;
125
126 // --- Core VTable Definition ---
127 /**
128 * @struct _BaseVTable
129 * @brief The foundational structure for
130 * all virtual tables.
131 * @details Every VTable for every class
132 * created with this framework *must*
133 * begin
134 * with this exact structure. This is
135 * the cornerstone of polymorphism in
136 * this system.
137 * It allows any `Object*` to be safely
138 * cast to a handle with a `_
139 * BaseVTable*`,
140 * guaranteeing that we can always call
141 * these fundamental methods (like `
142 * destroy` or
143 * `get_id`) regardless of the object's
144 * specific type.
145 */
146 typedef struct _BaseVTable {
147     void (*destroy)(Object*);
148     unsigned long (*get_id)(const void*)
149     ;
150     unsigned long (*get_group_id)(const
151     void*);
152     unsigned long (*get_parent_id)(const
153     void*);
154     void (*_private_destroy)(void*); //

```

```

        Internal hook for the class's
        own destructor logic.
84 } _BaseVTable;
85
86 // --- The Generic Object Handle ---
87 /**
88  * @struct AgenkObject
89  * @brief The public-facing, generic
        handle for any object in the
        framework.
90  * @details This is the "void pointer"
        of the object world. It's an opaque
        handle
91  * that contains the two essential
        pieces of information for
        polymorphism:
92  * 1. `vtable`: A pointer to the
        object's method table. Crucially,
        it's typed
93  * as a `_BaseVTable*` to allow
        safe access to the common methods.
94  * 2. `data`: A void pointer to the
        object's unique, private data
        members.
95  */
96 struct AgenkObject {
97     const _BaseVTable* vtable;
98     void* data;
99 };
100
101 /* ===== */
102 /* ===== PUBLIC FRAMEWORK API ===== */
103 /* ===== */
104
105 /** @brief Initializes the entire object
        framework runtime. Must be called
        once at application start. */
106 void CLASS_INIT(void);
107 /** @brief Shuts down the framework and
        cleans up all global resources. Must
        be called once at application end.
        */
108 void CLASS_SHUTDOWN(void);
109 /** @brief Creates a new lifecycle group
        for objects, returning a unique ID
        for that group. */
110 unsigned long CREATE_GROUP(const char*
        description);
111 /** @brief Destroys all objects
        associated with a specific group ID,
        preventing memory leaks. */
112 void DESTROY_BY_GROUP(unsigned long
        group_id);
113
114 /* ===== */
115 /* ===== PUBLIC OBJECT LIFECYCLE API ===== */
116 /* ===== */
117
118 /** @brief Creates a new "root" object (
        one without a parent) in a specified
        group. */
119 #define NEW_ROOT(ClassName, group_id,
        ...) \
120     ClassName##_create(group_id, 0, ##_
        VA_ARGS_)
121
122 /** @brief Creates a "child" object,
        automatically inheriting its parent'
        s group and setting its parent ID.
        */
123
        */
124 #define NEW_CHILD(ClassName, parent_obj,
        ...) \
125     ClassName##_create( \
126         (parent_obj ? GET(parent_obj,
        group_id, unsigned long) :
        0), \
127         (parent_obj ? GET(parent_obj, id
        , unsigned long) : 0), \
128         ##_VA_ARGS_ \
129     )
130
131 /** @brief Destroys a single object
        instance. Safe to call on NULL. */
132 #define DESTROY(obj) \
133     do { \
134         if (obj) { \
135             ((const Object*)obj)->vtable
        ->destroy((Object*)obj);
        \
136         } \
137     } while (0)
138
139 /** @brief (Internal) Destroys an object
        using its unique ID. Not intended
        for public use. */
140 void _class_destroy_by_id(unsigned long
        object_id);
141
142 /* ===== */
143 /* = PUBLIC OBJECT INTERACTION API = */
144 /* ===== */
145
146 /** @brief (C99 Compliant) Calls a
        method on an object that does not
        return a value. */
147 #define CALL(ClassName, obj, method,
        ...) \
148     do { \
149         if (obj) { \
150             ((const ClassName##_VTable*)
        ((const Object*)obj)->
        vtable)->method((void*)
        obj, ##_VA_ARGS_); \
151         } \
152     } while (0)
153
154 /** @brief (C99 Compliant) Calls a
        method on an object that returns a
        value. */
155 #define CALL_R(ClassName, obj, method,
        ret_type, ...) \
156     (obj ? \
157         ((const ClassName##_VTable*)((
        const Object*)obj)->vtable)
        ->method((void*)obj, ##_VA_
        ARGS_) \
158         : (ret_type)0)
159
160 /** @brief Gets the value of a built-in
        property (id, group_id, parent_id).
        */
161 #define GET(obj, property, type) \
162     (obj && ((const Object*)obj)->vtable
        ? \
163         ((const Object*)obj)->vtable->
        get_##property((const void*)
        obj) \
164         : (type)0)
165

```

```

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

```



```

    obj->data;
244
245 /** @brief Begins the implementation
    block for a specific class method.
    */
246 #define METHOD_IMPL(Classname, ret_type,
    name, ...) \
247     static ret_type __##Classname## __
    name(void* self_void, __##_VA_
    ARGS__); \
248     static ret_type __##Classname## __
    name(void* self_void, __##_VA_
    ARGS__) { \
249         Classname* self_obj = (Classname
    *)self_void; \
250         Classname##_Data* self = self_
    obj->data;
251
252 /** @brief Concludes a method
    implementation block. */
253 #define METHOD_IMPL_END }
254
255 /** @brief Begins the initialization of
    the global static VTable for a class
    . */
256 #define VTABLE_IMPL(Classname, ...) \
257     const Classname##_VTable VTABLE_##
    Classname = { \
258         .base = { \
259             .destroy = __##Classname##_
    destroy, \
260             .get_id = __##Classname##_get
    _id, \
261             .get_group_id = __##Classname
    __##_get_group_id, \
262             .get_parent_id = __##
    Classname##_get_parent_
    id, \
263             ._private_destroy = __##
    Classname##_private_
    destroy \
264         }, \
265         __##_VA_ARGS__ \
266     };
267
268 /* ===== */
269 /* ===== INTERNAL HELPERS ===== */
270 /* ===== */
271
272 // Prototypes for the backend functions
    hidden from the end-user.
273 unsigned long _class_registry_register(
    Object* obj);
274 void _class_registry_unregister(unsigned
    long id);
275 bool _class_db_log_creation(unsigned
    long object_id, unsigned long group_
    id, unsigned long parent_id,
    unsigned long class_hash);
276 void _class_db_log_destruction(unsigned
    long object_id);
277 unsigned long _class_db_create_group(
    const char* description);
278 void _class_db_find_by_group(unsigned
    long group_id, unsigned long** out_
    ids, size_t* out_count);
279 void _class_db_destroy_group(unsigned
    long group_id);
280 unsigned long _CLASS_HASH(const char*
    str);

```

```

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

```

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: `'pthread'` 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.
```

```
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:**
39  *       The 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  *       (pthread 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  */
72
73 #include "object.h"
74 #include <stdlib.h>
```

```

36 #include <stdio.h>
37
38
39 /*=====*/
40 /*=THREADING ABSTRACTION LAYER (TAL)=*/
41 /*=====*/
42 // This section uses preprocessor
43 // directives to select the correct
44 // implementation at compile time. This
45 // is the key to portability.
46 #ifndef _WIN32
47 // --- Windows Implementation (using
48 // Win32 API) ---
49 #include <windows.h>
50 typedef CRITICAL_SECTION lock_t;
51 #define LOCK_INIT(lock)
52     InitializeCriticalSection(lock)
53 #define LOCK_DESTROY(lock)
54     DeleteCriticalSection(lock)
55 #define LOCK(lock)
56     EnterCriticalSection(lock)
57 #define UNLOCK(lock)
58     LeaveCriticalSection(lock)
59 #else
60 // --- POSIX Implementation (using
61 // pthreads) ---
62 #include <pthread.h>
63 typedef pthread_mutex_t lock_t;
64 #define LOCK_INIT(lock) pthread_
65     mutex_init(lock, NULL)
66 #define LOCK_DESTROY(lock) pthread_
67     mutex_destroy(lock)
68 #define LOCK(lock) pthread_mutex_
69     lock(lock)
70 #define UNLOCK(lock) pthread_mutex_
71     unlock(lock)
72 #endif
73
74 /* --- Hash Table Configuration and
75 Structures --- */
76 #define REGISTRY_INITIAL_CAPACITY 4096
77 // A power of 2, larger means fewer
78 // collisions
79 typedef struct RegistryNode {
80     unsigned long id;
81     Object* object_ptr;
82     struct RegistryNode* next;
83 } RegistryNode;
84
85 static struct {
86     RegistryNode** table; // The
87     // array of pointers to hash table
88     // buckets (chains).
89     lock_t* locks; // An
90     // array of locks, one for each
91     // bucket.
92     size_t capacity;
93     size_t count; // Total
94     // number of objects. Must be
95     // modified under lock.
96 } g_registry;
97
98 // A dedicated lock for the ID counter
99 // ensures ID generation is atomic.
100 static unsigned long g_next_object_id =

```

```

1;
83 static lock_t g_id_lock;
84
85 static volatile bool g_is_initialized =
86     false;
87 static lock_t g_init_lock;
88
89 /* --- FNV-1a Hash Function
90 Implementation --- */
91 #define FNV_OFFSET_BASIS 2166136261U
92 #define FNV_PRIME 16777619U
93 static unsigned long _pyc_hash_id(
94     unsigned long id) {
95     unsigned long hash = FNV_OFFSET_
96     BASIS;
97     const char* p = (const char*)&id;
98     for (size_t i = 0; i < sizeof(
99     unsigned long); i++) {
100         hash ^= (unsigned long)p[i];
101         hash *= FNV_PRIME;
102     }
103     return hash;
104 }
105
106 /* ===== */
107 /* == INTERNAL LIFECYCLE FUNCTIONS == */
108 /* ===== */
109 // These are called only by object_impl.
110 // c's CLASS_INIT and CLASS_SHUTDOWN.
111
112 /**
113 * @brief Initializes the global object
114 * registry.
115 * This function is thread-safe using a
116 * double-checked locking pattern.
117 */
118 void _class_registry_init_internal(void)
119 {
120     if (g_is_initialized) return;
121
122     // Use a dedicated lock for one-time
123     // initialization.
124     LOCK_INIT(&g_init_lock);
125     LOCK(&g_init_lock);
126
127     // Re-check inside the lock in case
128     // another thread finished while we
129     // waited.
130     if (g_is_initialized) {
131         UNLOCK(&g_init_lock);
132         LOCK_DESTROY(&g_init_lock); //
133         // Clean up the lock if we didn't
134         // use it.
135         return;
136     }
137
138     g_registry.capacity = REGISTRY_
139     INITIAL_CAPACITY;
140     g_registry.count = 0;
141     g_registry.table = (RegistryNode**)
142     calloc(g_registry.capacity,
143     sizeof(RegistryNode));
144     g_registry.locks = (lock_t*)malloc(g
145     _registry.capacity * sizeof(lock
146     _t));
147
148     if (!g_registry.table || !g_registry
149     .locks) {

```



```

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

```

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

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  * @version 1.0.0
7  * @author Ankush Yadav, Ankit Yadav,
8  *        AuctaSapience
9  */
```

```

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 /* ===== */
45 /* ===== STATIC GLOBALS & HELPERS ===== */
46 /* ===== */
47
48 static struct {
49     MDB_env *env;
50     MDB_dbi objects_dbi;
51     MDB_dbi groups_dbi;
52     MDB_dbi counters_dbi;
53 } g_db;
54
55 /**
56  * @brief A suite of robust error
57  *        handling macros for LMDB operations
58  *
59  * @details This variadic macro handles
60  *        both void and non-void functions.
61  *        - MDB_CHECK(op, txn) -> used
62  *        in a void function, expands to `
63  *        return;`
64  *        - MDB_CHECK(op, txn, retval)
65  *        -> used in a non-void function,
66  *        expands to `return retval;`
67  */
68 #define MDB_CHECK(op, txn, ...) \
69     do { \
70         int rc = (op); \
71         if (rc != MDB_SUCCESS) { \
72             fprintf(stderr, "LMDB_ERROR: \
73                 %s:%d-%s-%s\n", \
74                     FILE__, __LINE__, #op, \
75                     mdb_strerror(rc)); \
76             if (txn) mdb_txn_abort(txn); \
77             return __VA_ARGS__; \
78         } \
79     } while (0)
80
81 // --- Static helper function
82 // declarations ---
83 static bool get_app_data_path(char*
84     buffer, size_t len);
85 static void ensure_dir_exists(const
86     char* path);
87 static unsigned long db_get_next_id(MDB
88     _txn* txn, const char* counter_key);
89
90 /* ===== */
91 /* == INTERNAL LIFECYCLE FUNCTIONS == */
92 /* ===== */
93
94 void class_db_init_internal(void) {
95     char db_path[1024];
96     if (!get_app_data_path(db_path,
97         sizeof(db_path))) {
98         fprintf(stderr, "CLASS_DB_FATAL:
99             \uCould\u\nnot\u\ndetermine\u\
100             application\u\data\u\ndirectory.\u\
101             n");
102         exit(EXIT_FAILURE);
103     }
104     ensure_dir_exists(db_path);
105
106     MDB_env *env;
107     MDB_CHECK(mdb_env_create(&env), NULL
108         );
109     MDB_CHECK(mdb_env_set_mapsize(env,
110         1024L * 1024L * 1024L), NULL);
111     MDB_CHECK(mdb_env_set_maxdbs(env, 4)
112         , NULL);
113
114     int rc = mdb_env_open(env, db_path,
115         MDB_WRITEMAP | MDB_NOSYNC, 0664)
116         ;
117     if (rc != MDB_SUCCESS) {
118         fprintf(stderr, "CLASS_DB_FATAL:
119             \uCould\u\nnot\u\nopen\u\nLMDB\u\
120             environment\u\nat\u\n%s:\u\n%s\n", db
121             _path, mdb_strerror(rc));
122         mdb_env_close(env);
123         exit(EXIT_FAILURE);
124     }
125     g_db.env = env;
126
127     MDB_txn *txn;
128     MDB_CHECK(mdb_txn_begin(g_db.env,
129         NULL, 0, &txn), NULL);
130     MDB_CHECK(mdb_dbi_open(txn, "objects
131         ", MDB_CREATE | MDB_INTEGERKEY,
132         &g_db.objects_dbi), txn);
133     MDB_CHECK(mdb_dbi_open(txn, "groups"
134         , MDB_CREATE | MDB_INTEGERKEY, &
135         g_db.groups_dbi), txn);
136     MDB_CHECK(mdb_dbi_open(txn, "
137         counters", MDB_CREATE, &g_db.
138         counters_dbi), txn);
139     // [FIX] Pass the transaction handle
140     // 'txn' to the macro. If commit
141     // fails, the transaction
142     // will now be properly aborted,
143     // preventing database corruption.
144     MDB_CHECK(mdb_txn_commit(txn), txn);
145 }

```

```

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:
209             Failed to delete object ID %
210             lu: %s\n", object_id, mdb_
211             strerror(rc));
212         mdb_txn_abort(txn);
213         return;
214     }
215     // [FIX] Pass the transaction handle
216     'txn' to the macro for proper
217     cleanup on failure.
218     MDB_CHECK(mdb_txn_commit(txn), txn);
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 =
            (unsigned long*)
            realloc(ids,
254             capacity * sizeof(
255                 unsigned long));
208         if (!new_ids) { free(ids
            ); goto cleanup; }
209         ids = new_ids;
210     }
211     ids[(*out_count)++] = meta->
        object_id;
212 }
213 }
214
215 cleanup:
216     mdb_cursor_close(cursor);
217     mdb_txn_abort(txn);
218     *out_ids = ids;
219 }
220
221 void _class_db_destroy_group(unsigned
    long group_id) {
222     if (group_id == 0) return;
223     MDB_txn *txn;
224     MDB_val key;
225     key.mv_size = sizeof(unsigned long);
226     key.mv_data = &group_id;
227
228     MDB_CHECK(mdb_txn_begin(g_db.env,
        NULL, 0, &txn), NULL);
229     int rc = mdb_del(txn, g_db.groups_
        dbi, &key, NULL);
230     if (rc != MDB_SUCCESS && rc != MDB_
        NOTFOUND) {
231         fprintf(stderr, "LMDB_WARNING:
            Failed to delete group ID %
            lu: %s\n", group_id, mdb_
            strerror(rc));
232         mdb_txn_abort(txn);
233         return;
234     }
235     // [FIX] Pass the transaction handle
        'txn' to the macro for proper
        cleanup on failure.
236     MDB_CHECK(mdb_txn_commit(txn), txn);
237 }
238
239 /* ===== */
240 /* === STATIC HELPER DEFINITIONS === */
241 /* ===== */
242
243
244 static bool _get_app_data_path(char*
    buffer, size_t len) {
245     #ifdef WIN32
246         if (SHGetFolderPath(NULL, CSIDL
            _APPDATA, NULL, 0, buffer)
            != S_OK) return false;
247         strncat(buffer, "\\AuctaSapience
            _AGENK", len - strlen(buffer
            ) - 1);
248     #else
249         const char* home = getenv("HOME
            ");
250         if (!home) home = getpwuid(
            getuid())->pw_dir;
251         if (!home) return false;
252         #ifdef APPLE
253             snprintf(buffer, len, "%s/

```

```

Library/Application_
Support/AuctaSapience_
AGENK", home);
254 #else
255     snprintf(buffer, len, "%s/.
        local/share/
        AuctaSapience_AGENK",
        home);
256 #endif
257 #endif
258 return true;
259 }
260
261 static void _ensure_dir_exists(const
    char* path) {
262     #ifdef WIN32
263         CreateDirectoryA(path, NULL);
264     #else
265         mkdir(path, 0755);
266     #endif
267 }
268
269 static unsigned long _db_get_next_id(MDB
    txn* txn, const char* counter_key)
270 {
271     unsigned long next_id = 1;
272     MDB_val key, data;
273     key.mv_size = strlen(counter_key);
274     key.mv_data = (void*)counter_key;
275
276     int rc = mdb_get(txn, g_db.counters_
        dbi, &key, &data);
277     if (rc == MDB_SUCCESS) {
278         memcpy(&next_id, data.mv_data,
            sizeof(unsigned long));
279     } else if (rc != MDB_NOTFOUND) {
280         fprintf(stderr, "LMDB_ERROR:
            Failed to read counter '%s':
            %s\n", counter_key, mdb_
            strerror(rc));
281         return 0;
282     }
283
284     unsigned long new_val = next_id + 1;
285     data.mv_size = sizeof(unsigned long)
        ;
286     data.mv_data = &new_val;
287     if (mdb_put(txn, g_db.counters_dbi,
        &key, &data, 0) != MDB_SUCCESS)
288     {
289         fprintf(stderr, "LMDB_ERROR:
            Failed to update counter '%s'
            : %s\n", counter_key, mdb_
            strerror(rc));
290         return 0;
291     }
292     return next_id;
293 }

```

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 #include "object.h" // The public API
51                       // header, provides type definitions
52                       // and prototypes.
53 #include <stdlib.h> // For NULL, free.
54 #include <stdint.h> // For portable

```

integer types.

```

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

```

```

78 /* == FRAMEWORK STATE & LIFECYCLE == */
79 /* ===== */
80
81 // A static global flag to ensure the
82 // initialization logic runs only once.
83 // This makes CLASS_INIT idempotent (
84 // safe to call multiple times).
85 static bool g_is_initialized = false;
86
87 /**
88  * @brief Public API function to
89  * initialize the object runtime.
90  * @details Initializes subsystems in
91  * the correct order: persistent
92  * storage (DB)
93  * first, then the volatile cache (
94  * Registry).
95  */
96 void CLASS_INIT(void) {
97     if (g_is_initialized) {
98         return;
99     }
100     _class_db_init_internal();
101     _class_registry_init_internal();
102     g_is_initialized = true;
103 }
104
105 /**
106  * @brief Public API function to shut
107  * down the object runtime.
108  * @details Shuts down subsystems in the
109  * reverse order of initialization to
110  * ensure a clean teardown.
111  */
112 void CLASS_SHUTDOWN(void) {
113     if (!g_is_initialized) {
114         return;
115     }
116     _class_registry_shutdown_internal();
117     _class_db_shutdown_internal();
118     g_is_initialized = false;
119 }
120
121 /* ===== */
122 /* === PUBLIC API IMPLEMENTATIONS === */
123 /* ===== */
124
125 /**
126  * @brief Public API function to create
127  * a new group.
128  * @details This is a thin wrapper that
129  * delegates the actual work to the
130  * database layer, which handles the
131  * creation transactionally.
132  */
133 unsigned long CREATE_GROUP(const char*
134     description) {
135     return _class_db_create_group(
136         description);
137 }
138
139 /**
140  * @brief Implements the DESTROY_BY_ID
141  * macro logic.
142  * @details This function bridges the
143  * gap between the "ID world" (which
144  * can be
145  * stored and passed around) and the "
146  * pointer world" (live objects in
147  * memory).
148  * It looks up the ID in the registry to
149  * get a live pointer, then calls the
150  * standard DESTROY macro on it.
151  * @param id The unique ID of the object
152  * to find and destroy.
153  */
154 void _class_destroy_by_id(unsigned long
155     id) {
156     // Find the live object pointer in
157     // the in-memory registry.
158     Object* obj_to_destroy = (Object*)_
159         class_registry_find_by_id(id);
160
161     // If the object is live, destroy it
162     // using the public DESTROY macro.
163     // This
164     // is crucial because it guarantees
165     // the full, correct destruction
166     // cascade
167     // is triggered (calling the user's
168     // DESTRUCTOR, unregistering, etc.)
169
170     // If the object is not found (
171     // already destroyed), this does
172     // nothing.
173     if (obj_to_destroy) {
174         DESTROY(obj_to_destroy);
175     }
176 }
177
178 /**
179  * @brief Public API function to destroy
180  * all objects in a group.
181  * @details This is a high-level
182  * orchestration function.
183  * 1. It queries the persistent
184  * database (the "source of truth")
185  * for a list
186  * of all object IDs belonging to
187  * the group.
188  * 2. It iterates through this list,
189  * destroying each object by its ID.
190  * 3. It frees the list of IDs.
191  * 4. Finally, it destroys the group
192  * metadata record itself from the
193  * database.
194  * @param group_id The ID of the group
195  * to destroy.
196  */
197 void DESTROY_BY_GROUP(unsigned long
198     group_id) {
199     if (group_id == 0) return; // 0 is
200     // not a valid group ID.
201
202     unsigned long* ids_to_destroy = NULL
203     ;
204     size_t count = 0;
205
206     // Step 1: Query the database for
207     // all members of the group.
208     _class_db_find_by_group(group_id, &
209         ids_to_destroy, &count);
210
211     if (count > 0 && ids_to_destroy) {
212         // Step 2: Iterate and destroy
213         // each object.
214         for (size_t i = 0; i < count; ++
215             i) {
216             _class_destroy_by_id(ids_to_

```

```

171         destroy[i]);
172     }
173     // Step 3: Free the memory
174     // allocated by the database
175     // function.
176     free(ids_to_destroy);
177 }
178 // Step 4: Destroy the group record
179 // itself.
180 _class_db_destroy_group(group_id);
181 }

```

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
5 # required to build this project.
6 # This ensures that developers use a
7 # compatible build system version.
8 cmake_minimum_required(VERSION 3.14)
9
10 # Define the project name, version, and
11 # primary language.
12 # This information is used by IDEs and
13 # packaging tools.
14 project(AGENK VERSION 1.0.0 LANGUAGES C)
15
16 # =====
17 # COMPILER AND BUILD STANDARDS
18 # =====
19 # Enforce the C99 standard for the
20 # entire project. This guarantees that
21 # portable language features are used
22 # and prevents reliance on compiler-
23 # specific extensions. This is critical
24 # for portability to compilers like
25 # TCC.
26 set(CMAKE_C_STANDARD 99)
27 set(CMAKE_C_STANDARD_REQUIRED ON)
28 set(CMAKE_C_EXTENSIONS OFF) # Explicitly
29 # disable GNU extensions
30
31 # Set the default build type to "Debug"
32 # if none is specified by the user.
33 # Debug builds include debugging symbols
34 # (-g) and do not define NDEBUG,
35 # which enables our framework's runtime
36 # safety assertions.
37 # A user can override this with `cmake -
38 # DCMAKE_BUILD_TYPE=Release ..`
39 if(NOT CMAKE_BUILD_TYPE)
40     set(CMAKE_BUILD_TYPE Debug)
41 endif()
42
43 # Add a message to inform the user of
44 # the build type.

```

```

32 message(STATUS "Build_type_set_to:${CMAKE_BUILD_TYPE}")
33
34
35 # =====
36 # DEPENDENCY MANAGEMENT (VENDORED LMDB)
37 # =====
38
39 # --- Define the LMDB library from its
40 # source files ---
41 # Since the vendored LMDB source does
42 # not have its own CMakeLists.txt, we
43 # define it as a library target here.
44 # This tells CMake to compile these
45 # specific source files into a static
46 # library named "lmdb".
47 add_library(lmdb STATIC
48     deps/lmdb/mdb.c
49     deps/lmdb/midl.c
50 )
51 # Tell any target that links against our
52 # `lmdb` library where to find its
53 # public header files (lmdb.h and midl.h
54 #). We declare the `deps/lmdb`
55 # directory as a PUBLIC include
56 # directory for the `lmdb` target.
57 target_include_directories(lmdb
58     PUBLIC
59     ${CMAKE_CURRENT_SOURCE_DIR}/deps
60     /lmdb
61 )
62
63 # =====
64 # CORE FRAMEWORK LIBRARY DEFINITION
65 # =====
66 # Define our core object system as a
67 # static library named "class_
68 # framework".
69 # A library is a bundle of pre-compiled
70 # code that can be reused by multiple
71 # executables. This is a fundamental
72 # principle of good software design.
73 # The paths are now corrected to match
74 # your project's file structure.
75 add_library(class_framework STATIC
76     src/objects/core/object_impl.c
77     src/objects/core/object_registry.c
78     src/objects/core/object_db.c
79 )
80 # Tell any target that links against `
81 # class_framework` where to find its
82 # public header files. The header `
83 # object.h` is in `src/objects`.
84 # We declare this PUBLIC so any
85 # executable linking to our framework
86 # also
87 # gets this include path automatically.
88 target_include_directories(class_
89     framework
90     PUBLIC
91     ${CMAKE_CURRENT_SOURCE_DIR}/src/
92     objects
93 )
94
95 # Link our framework against the LMDB
96 # library that we just defined above.
97 # We declare this as a PRIVATE
98
99 dependency because the end-user's
100 code does not
101 # need to know about LMDB; it's an
102 # internal implementation detail of
103 # our
104 # framework.
105 target_link_libraries(class_framework
106     PRIVATE
107     lmdb
108 )
109 # LMDB requires the pthreads library for
110 # its locking mechanisms on POSIX
111 # systems.
112 # This finds the library and links it.
113 # On non-POSIX systems like Windows,
114 # CMake handles this gracefully.
115 find_package(Threads REQUIRED)
116 target_link_libraries(class_framework
117     PRIVATE
118     Threads::Threads
119 )
120
121 # =====
122 # EXECUTABLE TARGETS (EXAMPLES & TESTS)
123 # =====
124 # The paths to all source files have
125 # been corrected to match your
126 # structure.
127
128 # --- Examples ---
129 add_executable(example_basic src/
130     objects/examples/basic_usage.c)
131 add_executable(example_lifecycle src/
132     objects/examples/lifecycle_and_
133     groups.c)
134
135 # --- Tests ---
136 add_executable(test_stress src/
137     objects/test/test_stress.c)
138 add_executable(test_errors src/
139     objects/test/test_errors.c)
140 add_executable(test_hierarchy src/
141     objects/test/test_hierarchy.c)
142 add_executable(test_concurrency src/
143     objects/test/test_concurrency.c)
144
145 # --- Link all executables against our
146 # framework ---
147 # This makes all the public functions
148 # and macros from our object system
149 # available to our example and test
150 # programs.
151 target_link_libraries(example_basic
152     PRIVATE class_framework)
153 target_link_libraries(example_lifecycle
154     PRIVATE class_framework)
155 target_link_libraries(test_stress
156     PRIVATE class_framework)
157 target_link_libraries(test_errors
158     PRIVATE class_framework)
159 target_link_libraries(test_hierarchy
160     PRIVATE class_framework)
161 target_link_libraries(test_concurrency
162     PRIVATE class_framework)
163
164 # =====
165 # INSTALLATION AND PACKAGING

```



```

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
134 #         DESTINATION lib)
135 # install(FILES src/objects/object.h
136 #         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 (
7     cond) { g_tests_passed++; printf("[
8     PASS]\n", msg); } else { printf("[
9     FAIL]\n", msg); } } while (0)
10
11 // --- Object Definitions ---
12 OBJECT_START(GrandParentObject)
13 VTABLE(GrandParentObject)
14 METHOD(Object*, create_child)
15 OBJECT_END(GrandParentObject)
16
17 OBJECT_START(ParentObject)
18 VTABLE(ParentObject)
19 METHOD(Object*, create_child)
20 OBJECT_END(ParentObject)
21
22 OBJECT_START(ChildObject)
23 VTABLE(ChildObject)
24 OBJECT_END(ChildObject)
25
26 // --- Constructor / Destructor
27 // Implementations ---
28 CONSTRUCTOR(GrandParentObject)
29 DESTRUCTOR(GrandParentObject) }

```

```

25 CONSTRUCTOR(ParentObject) DESTRUCTOR(
26     ParentObject) }
27 CONSTRUCTOR(ChildObject) DESTRUCTOR(
28     ChildObject) }
29
30 // --- Method Implementations ---
31 METHOD_IMPL(GrandParentObject, Object*,
32     create_child) {
33     return NEW_CHILD(ParentObject, self_
34         obj);
35 } METHOD_IMPL_END
36
37 METHOD_IMPL(ParentObject, Object*,
38     create_child) {
39     return NEW_CHILD(ChildObject, self_
40         obj);
41 } METHOD_IMPL_END
42
43 // --- VTable Implementations ---
44 VTABLE_IMPL(GrandParentObject, .create_
45     child = _GrandParentObject_create_
46     child)
47 VTABLE_IMPL(ParentObject, .create_child
48     = _ParentObject_create_child)
49 VTABLE_IMPL(ChildObject, )
50
51 int main() {
52     printf("Hierarchy & Context Test
53         Suite...\n");
54     CLASS_INIT();
55
56     unsigned long test_group = CREATE_
57         GROUP("HierarchyTest");
58
59     // 1. Create the root GrandParent
60     // object
61     Object* gp = NEW_ROOT(
62         GrandParentObject, test_group);
63     unsigned long gp_id = GET(gp, id,
64         unsigned long);
65
66     // 2. Test its properties
67     ASSERT_TRUE(GET(gp, group_id,
68         unsigned long) == test_group, "
69         Root has correct GroupID.");
70     ASSERT_TRUE(GET(gp, parent_id,
71         unsigned long) == 0, "Root has
72         ParentID of 0.");
73
74     // 3. Create a child Parent object
75     // UPDATED CALL_R: Added '
76     // GrandParentObject' as the first
77     // argument
78     Object* p = CALL_R(GrandParentObject
79         , gp, create_child, Object*);
80     unsigned long p_id = GET(p, id,
81         unsigned long);
82
83     // 4. Test the child's properties
84     ASSERT_TRUE(GET(p, group_id,
85         unsigned long) == test_group, "
86         Child inherits GroupID.");
87     ASSERT_TRUE(GET(p, parent_id,
88         unsigned long) == gp_id, "Child
89         has correct ParentID.");
90
91     // 5. Create a grandchild Child
92     // object
93     // 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)
        do { if (condition) { g_tests_passed
        ++; printf("[_PASS_]_%s\n", message)
        ; } else { printf("[_FAIL_]_%s\n",
        message); } } while (0)
7
8  // Define a simple object class for the
        test
9  OBJECT_START(ErrorObject)
10 VTABLE(ErrorObject)
11 OBJECT_END(ErrorObject)
12
13 CONSTRUCTOR(ErrorObject)
14 DESTRUCTOR(ErrorObject)
15 }
16 VTABLE_IMPL(ErrorObject,)
17
18
19 int main() {
20     printf("Error_Handling_Test_Suite
        ... \n");
21     CLASS_INIT();
22
23     // Test operations on a NULL object
24     Object* null_obj = NULL;
25     DESTROY(null_obj);
26     ASSERT_TRUE(GET(null_obj, id,
        unsigned long) == 0, "GET_on_
        NULL_returns_zero.");
27
28     // Test destroying a non-existent
        object ID and group ID
29     // We now use the correct internal
        function name: _class_destroy_by_
        _id
30     _class_destroy_by_id(999999);
31     DESTROY_BY_GROUP(999999);
32     ASSERT_TRUE(true, "Destroy_by_

```

```

        invalid_ID/Group_did_not_crash."
        );
33
34     // Test double destruction
35     unsigned long group = CREATE_GROUP("
        Double_D_Test");
36     Object* obj = NEW_ROOT(ErrorObject,
        group);
37     unsigned long id = GET(obj, id,
        unsigned long);
38
39     // First destruction via the object
        pointer
40     DESTROY(obj);
41
42     // Second destruction attempt via
        the (now invalid) ID
43     _class_destroy_by_id(id);
44     ASSERT_TRUE(true, "Double_
        destruction_did_not_crash.");
45
46     CLASS_SHUTDOWN();
47     printf("Error_Handling_Tests_Passed:
        _%d/3\n", g_tests_passed);
48     return !(g_tests_passed == 3);
49 }

```

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", self->target_name, self
        ->id);
20 METHOD_IMPL_END
21
22 VTABLE_IMPL(Greeting, .say_hello = _
        Greeting_say_hello)
23
24 int main() {
25     printf("---_Basic_Usage_Example_---\
        n\n");

```

```

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

```

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 lifecycle 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 OBJECT_START(Subtask) char* subtask_name;
10     VTABLE(Subtask) OBJECT_END(Subtask, const char* name);
11
12 CONSTRUCTOR(Task, const char* name) self
13     ->task_name = strdup(name);
14     DESTRUCTOR(Task) free(self->task_name); }
15 CONSTRUCTOR(Subtask, const char* name)
16     self->subtask_name = strdup(name);
17     DESTRUCTOR(Subtask) free(self->subtask_name); }
18
19 METHOD_IMPL(Task, Object*, add_subtask, const char* subtask_name)
20     printf("Creating a child subtask...\n", self->task_name, self->id);
21     return NEW_CHILD(Subtask, self_obj, subtask_name);
22 METHOD_IMPL_END

```

```

16 VTABLE_IMPL(Task, .add_subtask = _Task_add_subtask)
17 VTABLE_IMPL(Subtask, )
18
19 void print_object_details(Object* obj, const char* obj_name) {
20     if (!obj) { printf("Cannot print details for NULL object '%s'.\n", obj_name); return; }
21     printf("Details for %s:\n", obj_name);
22     printf("ObjectID: %lu\n", GET(obj, id, unsigned long));
23     printf("GroupID: %lu\n", GET(obj, group_id, unsigned long));
24     printf("ParentID: %lu (0 means it is a root object)\n", GET(obj, parent_id, unsigned long));
25 }
26
27 int main() {
28     printf("--- Lifecycle and Group Management Example ---\n\n");
29     CLASS_INIT();
30     printf("1. Creating a 'Workflow' group...\n");
31     unsigned long workflow_group_id = CREATE_GROUP("Process daily sales report");
32     printf("New GroupID: %lu\n", workflow_group_id);
33     printf("2. Creating a root 'Task' object...\n");
34     Object* main_task = NEW_ROOT(Task, workflow_group_id, "Generate Report");
35     print_object_details(main_task, "Main Task");
36     printf("\n3. Creating child objects...\n");
37     Object* subtask1 = CALL_R(Task, main_task, add_subtask, Object*, "Fetch data");
38     Object* subtask2 = CALL_R(Task, main_task, add_subtask, Object*, "Format PDF");
39     printf("\nVerifying child object details...\n");
40     print_object_details(subtask1, "Subtask1");
41     print_object_details(subtask2, "Subtask2");
42     printf("\n4. Workflow complete. Destroying entire group...\n");
43     DESTROY_BY_GROUP(workflow_group_id);
44     printf("Group %lu and all its objects have been destroyed.\n", workflow_group_id);
45     printf("\n5. Shutting down...\n");
46     CLASS_SHUTDOWN();
47     printf("\n--- Example Finished Cleanly ---\n");
48     return 0;
49 }
50

```

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 *
   OBJECTS_PER_GROUP)
9
10 OBJECT_START(StressObject) int value;
   VTABLE(StressObject) OBJECT_END(
   StressObject, int val);
11 CONSTRUCTOR(StressObject, int val) self
   ->value = val; DESTRUCTOR(
   StressObject) }
12 VTABLE_IMPL(StressObject,)
13
14 int main() {
15     printf("StressTest...\n");
16     CLASS_INIT();
17     unsigned long groups[NUM_GROUPS];
18     clock_t start = clock();
19     for (int i = 0; i < NUM_GROUPS; ++i)
20         { groups[i] = CREATE_GROUP("
   StressTestGroup"); }
21     long long total_created = 0;
22     for (int i = 0; i < NUM_GROUPS; ++i)
23         {
24             for (int j = 0; j < OBJECTS_PER_
   GROUP; ++j) {
25                 NEW_ROOT(StressObject,
   groups[i], j);
26                 total_created++;
27                 if (total_created % 1000 ==
   0) { printf("\r->
   Creating:%lld/%d",
   total_created, TOTAL_
   OBJECTS); fflush(stdout)
   ; }
28             }
29         }
30     printf("\r->Creationcomplete.
   \n");
31     clock_t mid = clock();
32     for (int i = 0; i < NUM_GROUPS; ++i)
33         { DESTROY_BY_GROUP(groups[i]);
   }
34     clock_t end = clock();
35     printf("\r->Destructioncomplete.\n"
   );
36     double creation_time = ((double)(mid
```

```
34         - start)) / CLOCKS_PER_SEC;
   double destruction_time = ((double)(
   end - mid)) / CLOCKS_PER_SEC;
35     printf("Creation_time:%.4f_s,
   Destruction_time:%.4f_s\n",
   creation_time, destruction_time)
   ;
36     CLASS_SHUTDOWN();
37     printf("StressTestPassed.\n");
38     return 0;
39 }
```

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;
   VTABLE(ThreadObject) OBJECT_END(
   ThreadObject, int v);
11 CONSTRUCTOR(ThreadObject, int v) self->
   val = v; DESTRUCTOR(ThreadObject) }
12 VTABLE_IMPL(ThreadObject,)
13
14 void* worker_thread_func(void* arg) {
15     int thread_id = *(int*)arg;
16     unsigned long my_groups[GROUPS_PER_
   THREAD];
17     for (int i = 0; i < GROUPS_PER_
   THREAD; ++i) {
18         my_groups[i] = CREATE_GROUP("
   Threadgroup");
19         for (int j = 0; j < OBJECTS_PER_
   GROUP; ++j) {
20             if (!NEW_ROOT(ThreadObject,
   my_groups[i], j)) {
21                 printf("Thread%d:NEW_
   ROOTfailed.\n", thread_
   id); return NULL; }
22         }
23     }
24     for (int i = 0; i < GROUPS_PER_
   THREAD; ++i) { DESTROY_BY_GROUP(
   my_groups[i]); }
25     printf("\r->Thread%d:Complete.\n"
   , thread_id);
26     return NULL;
27 }
28
29 int main() {
30     printf("ConcurrencyTestSuite...\n"
   );
31     CLASS_INIT();
32     pthread_t threads[NUM_THREADS];
33     int thread_ids[NUM_THREADS];
34     for (int i = 0; i < NUM_THREADS; ++i
   ) {
35         thread_ids[i] = i;
36         pthread_create(&threads[i], NULL
   , worker_thread_func, &
   thread_ids[i]);
37     }
```

```

37     for (int i = 0; i < NUM_THREADS; ++i
        ) { pthread_join(threads[i],
            NULL); }
38     CLASS_SHUTDOWN();
39     printf("Concurrency Test Passed. \n");
40     return 0;
41 }

```

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 the inputs by the
          method of integration
          by parts."
5     ],
6     // if there is just one data inside
        a property then it can be
        written like property: "data"
        too.
7     some_property: "Hi, How Are You",
8     program_space :
9     [
10         (u, dv, x_start, x_end) => {
11             v = integrate(dv, x_start, x
                _end),
12             du = differentiate(u),
13             uv = multiply(u, v),
14             v_du = integrate(multiply(v,
                du), x_start, x_end),
15             result = subtract(uv, v_du),
16             return result
17         },
18         (some_other_arguements) => {
19             some_method
20         }
21         // a single object can contain
            many programs.
22     ],
23     some_data:
24     {
25         property: "some_property"Co
26         data:
27         [
28             {
29                 isCorrect: true,
30                 isProgram: false
31             },
32             {
33                 status:
34                 "Completed"
35             }
36         ]
37     }
38 }

```

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/                   (Self-contained third
│   ├── cJSON/
│   ├── flatcc/
│   ├── lmdb/
│   └── md5/
├── src/
│   ├── objects/           (The live object processing
│   │   ├── core/
│   │   ├── examples/
│   │   ├── test/
│   │   └── object.h
│   └── object_store/      (The new object persistence
│       ├── 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: 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
  for a generic object in the AGENK
  framework.
3 // This schema uses FlatBuffers for high
  -performance, zero-copy access to
4 // persistent object data.
5 namespace objstore;
6
7 // A union of all supported primitive
  and complex data types for a field's
  value.
8 // FlatBuffers automatically adds a
  hidden type tag, making this a
  tagged union.
9 union FieldValue {
10   IntValue,
11   BoolValue,
12   FloatValue,
13   StringValue,
14   ArrayValue,
15   MapValue,
16   ObjectValue,
17   PointerValue
```

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

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
13 if(POLICY CMP0048)
14     cmake_policy(SET CMP0048 NEW)
15 endif()
16
17 project(AGENK VERSION 1.1.9 LANGUAGES C)
18
19 set(CMAKE_C_STANDARD 99)
20 set(CMAKE_C_STANDARD_REQUIRED ON)
21 set(CMAKE_C_EXTENSIONS OFF)
22
23 if(NOT CMAKE_BUILD_TYPE)
24     set(CMAKE_BUILD_TYPE Debug)
25 endif()
```

```

27 message(STATUS "Build_type_set_to: ${
    CMAKE_BUILD_TYPE}")
28
29 # =====
30 # SECTION 2: DEPENDENCY MANAGEMENT
31 # =====
32 find_package(Threads REQUIRED)
33
34 set(FLATCC_TEST OFF CACHE BOOL "Disable_
    flatcc's_internal_tests")
35 set(FLATCC_SAMPLES OFF CACHE BOOL "
    Disable_flatcc's_internal_samples")
36 add_subdirectory(deps/flatcc)
37
38 add_library(lmdb STATIC deps/lmdb/mdb.c
    deps/lmdb/midl.c)
39 target_include_directories(lmdb PUBLIC $
    {CMAKE_CURRENT_SOURCE_DIR}/deps/lmdb
    )
40
41 add_library(md5 STATIC deps/md5/md5.c)
42 target_include_directories(md5 PUBLIC ${
    CMAKE_CURRENT_SOURCE_DIR}/deps/md5)
43
44 set(TEST_EXTRA_LIBS "")
45 if(EXISTS ${CMAKE_CURRENT_SOURCE_DIR}/
    deps/cJSON/cJSON.c)
46 message(STATUS "cJSON_source_found,
    building_library.")
47 add_library(cjson STATIC deps/cJSON/
    cJSON.c)
48 target_include_directories(cjson
    PUBLIC ${CMAKE_CURRENT_SOURCE_
    DIR}/deps/cJSON)
49 list(APPEND TEST_EXTRA_LIBS cjson)
50 set(CJSON_FOUND TRUE)
51 endif()
52
53 # =====
54 # SECTION 3: AUTOMATED CODE GENERATION
55 # =====
56 set(FBS_SCHEMA ${CMAKE_CURRENT_SOURCE_
    DIR}/src/object_store/core/object.
    fbs)
57 set(GENERATED_INCLUDE_DIR ${CMAKE_
    CURRENT_BINARY_DIR}/generated/object_
    store)
58 file(MAKE_DIRECTORY ${GENERATED_INCLUDE_
    DIR})
59
60 set(GENERATED_FILES
    ${GENERATED_INCLUDE_DIR}/object_
    builder.h
    ${GENERATED_INCLUDE_DIR}/object_
    reader.h
    ${GENERATED_INCLUDE_DIR}/object_
    verifier.h
    ${GENERATED_INCLUDE_DIR}/flatbuffers_
    common.h
    )
61
62 add_custom_command(
63     OUTPUT ${GENERATED_FILES}
64     COMMAND ${<TARGET_FILE:flatcc_cli> -a
        -o ${GENERATED_INCLUDE_DIR} ${
        FBS_SCHEMA}
65     DEPENDS ${FBS_SCHEMA} flatcc_cli
66     COMMENT "Generating_C_headers_from_
    FlatBuffers_schema: object.fbs"
    )
67
68 add_custom_target(generate_object_
    headers DEPENDS ${GENERATED_FILES})
69
70 # =====
71 # SECTION 4: CORE LIBRARY DEFINITIONS
72 # =====
73
74 add_library(class_framework STATIC
    src/objects/core/object_impl.c
    src/objects/core/object_registry.c
    src/objects/core/object_db.c
    )
75 target_include_directories(class_
    framework PUBLIC ${CMAKE_CURRENT_
    SOURCE_DIR}/src/objects)
76 target_link_libraries(class_framework
    PRIVATE lmdb Threads::Threads)
77
78 add_library(object_store STATIC
    src/object_store/core/fb_serializer.
    c
    src/object_store/core/object_store.c
    )
79 add_dependencies(object_store generate_
    object_headers)
80 target_include_directories(object_store
    PUBLIC
    ${CMAKE_CURRENT_SOURCE_DIR}/src/
    object_store
    ${GENERATED_INCLUDE_DIR}
    ${CMAKE_CURRENT_SOURCE_DIR}/deps/
    flatcc/include
    )
81 target_link_libraries(object_store
    PRIVATE
    flatccrt
    lmdb
    md5
    Threads::Threads
    )
82
83 if(CJSON_FOUND)
84     target_link_libraries(object_store
    PRIVATE cjson)
85     target_compile_definitions(object_
    store PRIVATE CJSON_ENABLED)
86 endif()
87
88 # =====
89 # SECTION 5: EXECUTABLE TARGETS (TESTS)
90 # =====
91
92 add_executable(test_stress src/
    objects/test/test_stress.c)
93 add_executable(test_errors src/
    objects/test/test_errors.c)
94 add_executable(test_hierarchy src/
    objects/test/test_hierarchy.c)
95 add_executable(test_concurrency src/
    objects/test/test_concurrency.c)
96
97 target_link_libraries(test_stress
    PRIVATE class_framework)
98 target_link_libraries(test_errors
    PRIVATE class_framework)
99 target_link_libraries(test_hierarchy
    PRIVATE class_framework)

```

```

125 target_link_libraries(test_concurrency
    PRIVATE class_framework Threads::
    Threads)
126
127 add_executable(test_object_store src/
    object_store/test/test_object_store.
    c)
128 # **FIX**: Add the math library 'm' to
    the list of linked libraries.
129 target_link_libraries(test_object_store
    PRIVATE
130     object_store
131     Threads::Threads
132     ${TEST_EXTRA_LIBS}
133     m # Link against the math
134     library
135 )
136
137 # =====
138 # SECTION 6: TESTING INTEGRATION
139 # =====
140
141
142 enable_testing()
143 add_test(NAME Framework.StressTest
    COMMAND test_stress)
144 add_test(NAME Framework.ErrorHandling
    COMMAND test_errors)
145 add_test(NAME Framework.Hierarchy
    COMMAND test_hierarchy)
146 add_test(NAME Framework.Concurrency
    COMMAND test_concurrency)
147 add_test(NAME ObjectStore.Core
    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
    structures that represent a
    deserialized object.
9 // This allows them to be used in
    pointers within the struct
    definitions themselves.
10 typedef struct c_object_t c_object_t;
11 typedef struct c_field_t c_field_t;
12 typedef struct c_field_value_t c_field_
    value_t;
13 typedef struct c_map_pair_t c_map_pair_t
    ;
14
15 // =====

```

```

16 // SECTION 1: ENUMS FOR STATUS AND TYPES
17 // =====
18
19 /**
20  * @enum fb_serializer_status_t
21  * @brief Represents the result of a
    serializer or object store
    operation.
22  * This comprehensive enum provides
    granular error reporting for the
    entire
23  * persistence layer, from memory
    allocation to database transaction
    failures.
24  */
25 typedef enum {
26     FB_SERIALIZER_OK = 0,
27     FB_SERIALIZER_ERROR_ALLOCATION,
28     FB_SERIALIZER_ERROR_INVALID_INPUT,
29     FB_SERIALIZER_ERROR_TYPE_MISMATCH,
30     FB_SERIALIZER_ERROR_JSON_PARSE,
31     FB_SERIALIZER_ERROR_FLATCC_BUILDER,
32     FB_SERIALIZER_ERROR_FLATCC_VERIFIER,
33     FB_SERIALIZER_ERROR_NOT_INITIALIZED,
34     FB_SERIALIZER_ERROR_SYSTEM_CALL,
35     FB_SERIALIZER_ERROR_LMDB_ENV_CREATE,
36     FB_SERIALIZER_ERROR_LMDB_ENV_OPEN,
37     FB_SERIALIZER_ERROR_LMDB_TXN_BEGIN,
38     FB_SERIALIZER_ERROR_LMDB_TXN_COMMIT,
39     FB_SERIALIZER_ERROR_LMDB_DBI_OPEN,
40     FB_SERIALIZER_ERROR_LMDB_PUT,
41     FB_SERIALIZER_ERROR_LMDB_GET,
42     FB_SERIALIZER_ERROR_LMDB_DEL,
43     FB_SERIALIZER_ERROR_LMDB_DROP,
44     FB_SERIALIZER_ERROR_LMDB_NOT_FOUND,
45     FB_SERIALIZER_ERROR_LMDB_KEY_EXISTS,
46 } fb_serializer_status_t;
47
48 /**
49  * @enum c_field_value_type_t
50  * @brief Distinguishes the type of data
    held in the 'c_field_value_t'
    union.
51  * This must be kept in sync with the `
    FieldValue` union in `object.fbs`.
52  */
53 typedef enum {
54     C_FIELD_VALUE_TYPE_NONE = 0,
55     C_FIELD_VALUE_TYPE_INT_VALUE,
56     C_FIELD_VALUE_TYPE_BOOL_VALUE,
57     C_FIELD_VALUE_TYPE_FLOAT_VALUE,
58     C_FIELD_VALUE_TYPE_STRING_VALUE,
59     C_FIELD_VALUE_TYPE_ARRAY_VALUE,
60     C_FIELD_VALUE_TYPE_MAP_VALUE,
61     C_FIELD_VALUE_TYPE_OBJECT_VALUE,
62     C_FIELD_VALUE_TYPE_POINTER_VALUE,
63 } c_field_value_type_t;
64
65 // =====
66 // SECTION 2: C DATA STRUCTURES
67 //
68 // These structs are the in-memory,
69 // C-native representation of an
70 // object's persistent data. They
71 // are designed to be easy to work
72 // with in C code.
73 // =====
74
75 /**
76

```



```

77  * @struct c_field_value_t
78  * @brief A tagged union representing
    the value of a field.
79  */
80  struct c_field_value_t {
81      c_field_value_type_t type;
82      union {
83          int64_t int_val;
84          bool bool_val;
85          float float_val;
86          char* string_val; //
            Used for StringValue and
            PointerValue
87          c_field_t* array_values; //
            Linked list of fields for an
            array
88          c_map_pair_t* map_pairs; //
            Linked list of key-value
            pairs for a map
89          c_object_t* object_val; // A
            nested, embedded object
90      } data;
91  };
92
93  /**
94   * @struct c_field_t
95   * @brief A single named field (a key-
    value pair). Forms a linked list
    for an
96   *
    object's or an array's fields.
97   */
98  struct c_field_t {
99      char* name;
100     c_field_value_t* value;
101     c_field_t* next;
102 };
103
104 /**
105  * @struct c_map_pair_t
106  * @brief A single key-value pair within
    a map. Forms a linked list.
107  */
108  struct c_map_pair_t {
109      char* key;
110      c_field_value_t* value;
111      c_map_pair_t* next;
112 };
113
114 /**
115  * @struct c_object_t
116  * @brief The top-level C representation
    of a deserialized object.
117  */
118  struct c_object_t {
119      char* object_id; // The object's
        unique, persistent identifier
120      char* table_name; // The name of
        the database table it belongs to
121      c_field_t* fields; // Head of the
        linked list of fields
122 };
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*
    fv);
136 void free_c_map_pair_list(c_map_pair_t*
    head);
137
138 // =====
139 // SECTION 4: C-STRUCT CREATION HELPERS
140 //
141 // Convenience functions for creating
142 // `c_field_value_t` structs for each
143 // data type. These are useful for
144 // programmatically building an object
145 // in C before serialization.
146 // =====
147 c_field_value_t* create_c_string_value(
    const char* val);
148 c_field_value_t* create_c_int_value(
    int64_t val);
149 c_field_value_t* create_c_bool_value(
    bool val);
150 c_field_value_t* create_c_float_value(
    float val);
151 c_field_value_t* create_c_pointer_value(
    const char* object_id);
152 c_field_value_t* create_c_none_value(
    void);
153
154 // =====
155 // SECTION 5: CORE SERIALIZATION API
156 // =====
157
158 /**
159  * @brief Serializes a C object
    structure into a FlatBuffer binary
    representation.
160  *
161  * @param obj The C object to serialize.
162  * @param out_buffer Pointer to a void*
    that will receive the newly
    allocated buffer.
163  *
    The caller is
    responsible for freeing this memory
164  *
165  * @param out_size Pointer to a size_t
    that will be populated with the
    buffer's size.
166  * @return FB_SERIALIZER_OK on success,
    or an error code on failure.
167  */
168 fb_serializer_status_t serialize_c_
    object(const c_object_t* obj, void**
    out_buffer, size_t* out_size);
169
170 /**
171  * @brief Deserializes a FlatBuffer from
    a binary buffer into a C object
    structure.
172  *
173  * @param buffer Pointer to the
    FlatBuffer data.
174  * @param size The size of the buffer.
175  * @param out_obj Pointer to a c_object_
    t* that will receive the newly

```

```

179 *         allocated C object.
180 *         The caller is
181 *         responsible for freeing this object
182 *         with free_c_object().
183 * @return FB_SERIALIZER_OK on success,
184 *         or an error code on failure.
185 */
186 fb_serializer_status_t deserialize_to_c_
187 object(const void* buffer, size_t
188 size, c_object_t** out_obj);
189
190 /**
191 * @brief Parses a JSON string and
192 *         converts it into a C object
193 *         structure.
194 *
195 * @param json_string The null-
196 *         terminated JSON string.
197 * @param default_object_id An object ID
198 *         to use if "object_id" is not in
199 *         the JSON.
200 * @param default_table_name A table
201 *         name to use if "table_name" is not
202 *         in the JSON.
203 * @param out_c_obj Pointer to a c_
204 *         object_t* that will receive the
205 *         allocated C object.
206 *
207 *         The caller is
208 *         responsible for freeing this object
209 *
210 * @return FB_SERIALIZER_OK on success,
211 *         or an error code on failure.
212 */
213 fb_serializer_status_t fb_object_from_
214 json_string(
215     const char *json_string,
216     const char *default_object_id,
217     const char *default_table_name,
218     c_object_t **out_c_obj);
219
220 #endif // FB_SERIALIZER_H

```

Listing 15: The public API for the serialization layer (src/object_store/fb_serializer.h).

The implementation in ‘fb_serializer.c’ is a set of recursive functions that traverse the ‘c_object_t’ linked-list structures and make corresponding calls to the ‘flatcc’ builder API. A critical lesson learned during development was the necessity of adhering strictly to the builder’s state machine: ‘init’, then ‘start_buffer’, then object construction, then ‘end_buffer’ to set the root, and finally ‘finalize_aligned_buffer’ to safely extract a copy of the completed buffer. Any deviation from this sequence resulted in corrupted buffers and runtime assertion failures.

```

1 // fb_serializer.c
2
3 #include "fb_serializer.h"
4
5 #include "object_builder.h"
6 #include "object_reader.h"
7 #include "object_verifier.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

```

```

106 ---
107 c_field_value_t* create_c_string_value(
108     const char* val) {
109     c_field_value_t* fv = (c_field_value_
110         t*)malloc(sizeof(c_field_value_
111             t));
112     if (!fv) return NULL;
113     fv->type = C_FIELD_VALUE_TYPE_STRING_
114         VALUE;
115     fv->data.string_val = val ? ansi_
116         strdup(val) : NULL;
117     if (val && !fv->data.string_val) {
118         free(fv); return NULL; }
119     return fv;
120 }
121 c_field_value_t* create_c_int_value(
122     int64_t val) {
123     c_field_value_t* fv = (c_field_value_
124         t*)malloc(sizeof(c_field_value_
125             t));
126     if (!fv) return NULL;
127     fv->type = C_FIELD_VALUE_TYPE_INT_
128         VALUE;
129     fv->data.int_val = val;
130     return fv;
131 }
132 c_field_value_t* create_c_bool_value(
133     bool val) {
134     c_field_value_t* fv = (c_field_value_
135         t*)malloc(sizeof(c_field_value_
136             t));
137     if (!fv) return NULL;
138     fv->type = C_FIELD_VALUE_TYPE_BOOL_
139         VALUE;
140     fv->data.bool_val = val;
141     return fv;
142 }
143 c_field_value_t* create_c_float_value(
144     float 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_FLOAT_
150         VALUE;
151     fv->data.float_val = val;
152     return fv;
153 }
154 c_field_value_t* create_c_pointer_value(
155     const char* object_id) {
156     c_field_value_t* fv = (c_field_value_
157         t*)malloc(sizeof(c_field_value_
158             t));
159     if (!fv) return NULL;
160     fv->type = C_FIELD_VALUE_TYPE_
161         POINTER_VALUE;
162     fv->data.string_val = object_id ?
163         ansi_strdup(object_id) : NULL;
164     if (object_id && !fv->data.string_
165         val) { free(fv); return NULL; }
166     return fv;
167 }
168 c_field_value_t* create_c_none_value() {
169     c_field_value_t* fv = (c_field_value_
170         t*)malloc(sizeof(c_field_value_

```

150	t));		
151	if (!fv) return NULL;		
152	fv->type = C_FIELD_VALUE_TYPE_NONE;		
153	return fv;	186	flatcc_builder_create_string_str(B, fv->data.string_val);
154	}		objstore_PointerValue_ref_t ref = objstore_PointerValue_create(B, s);
155	// --- Serialization Implementation ---		*out_union_ref = objstore_FieldValue_as_PointerValue(ref);
156	static int create_fb_field_value_union(187	break;
	flatcc_builder_t *B, const c_field_value_t *fv, objstore_FieldValue_union_ref_t *out_union_ref) {	188	}
157	if (!fv fv->type == C_FIELD_VALUE_TYPE_NONE) {	189	case C_FIELD_VALUE_TYPE_OBJECT_VALUE: {
158	*out_union_ref = objstore_FieldValue_as_NONE();	190	objstore_Object_ref_t obj_ref = create_fb_object_recursive(B, fv->data.object_val);
159	return 0;	191	objstore_ObjectValue_ref_t ref = objstore_ObjectValue_create(B, obj_ref);
160	}	192	*out_union_ref = objstore_FieldValue_as_ObjectValue(ref);
161		193	break;
162	switch (fv->type) {	194	}
163	case C_FIELD_VALUE_TYPE_INT_VALUE: {	195	case C_FIELD_VALUE_TYPE_ARRAY_VALUE: {
164	objstore_IntValue_ref_t ref = objstore_IntValue_create(B, fv->data.int_val);	196	c_field_t *p = fv->data.array_values;
165	*out_union_ref = objstore_FieldValue_as_IntValue(ref);	197	objstore_Field_vec_start(B);
166	break;	198	while(p) {
167	}	199	objstore_Field_ref_t field_ref = create_fb_field(B, p);
168	case C_FIELD_VALUE_TYPE_BOOL_VALUE: {	200	objstore_Field_vec_push(B, field_ref);
169	objstore_BoolValue_ref_t ref = objstore_BoolValue_create(B, fv->data.bool_val);	201	p = p->next;
170	*out_union_ref = objstore_FieldValue_as_BoolValue(ref);	202	}
171	break;	203	flatbuffers_vec_ref_t vec = objstore_Field_vec_end(B);
172	}	204	objstore_ArrayValue_ref_t ref = objstore_ArrayValue_create(B, vec);
173	case C_FIELD_VALUE_TYPE_FLOAT_VALUE: {	205	*out_union_ref = objstore_FieldValue_as_ArrayValue(ref);
174	objstore_FloatValue_ref_t ref = objstore_FloatValue_create(B, fv->data.float_val);	206	break;
175	*out_union_ref = objstore_FieldValue_as_FloatValue(ref);	207	}
176	break;	208	case C_FIELD_VALUE_TYPE_MAP_VALUE: {
177	}	209	c_map_pair_t *p = fv->data.map_pairs;
178	case C_FIELD_VALUE_TYPE_STRING_VALUE: {	210	objstore_MapPair_vec_start(B);
179	flatbuffers_string_ref_t s = flatcc_builder_create_string_str(B, fv->data.string_val);	211	while(p) {
180	objstore_StringValue_ref_t ref = objstore_StringValue_create(B, s);	212	flatbuffers_string_ref_t key = flatcc_builder_create_string_str(B, p->key);
181	*out_union_ref = objstore_FieldValue_as_StringValue(ref);	213	objstore_FieldValue_union_ref_t val;
182	break;	214	create_fb_field_value_union(B, p->value, &val);
183	}	215	objstore_MapPair_ref_t
184	case C_FIELD_VALUE_TYPE_POINTER_VALUE: {	216	
185	flatbuffers_string_ref_t s =		

```

217         pair_ref = objstore_MapPair_create(B,
218         MapPair_create(B,
219         key, val);
220         objstore_MapPair_vec
221         push(B, pair_ref);
222         p = p->next;
223     }
224     flatbuffers_vec_ref_t vec =
225     objstore_MapPair_vec_end
226     (B);
227     objstore_MapValue_ref_t ref
228     = objstore_MapValue_
229     create(B, vec);
230     *out_union_ref = objstore_
231     FieldValue_as_MapValue(
232     ref);
233     break;
234 }
235 default: return -1;
236 }
237 return 0;
238 }
239 static objstore_Field_ref_t create_fb_
240 field(flatcc_builder_t *B, const c_
241 field_t *cf) {
242     if (!cf) return 0;
243     flatbuffers_string_ref_t name =
244     flatcc_builder_create_string_str
245     (B, cf->name);
246     objstore_FieldValue_union_ref_t
247     value_union;
248     create_fb_field_value_union(B, cf->
249     value, &value_union);
250     return objstore_Field_create(B, name
251     , value_union);
252 }
253 // Replace the old create_fb_object_
254 recursive with this robust version
255 static objstore_Object_ref_t create_fb_
256 object_recursive(flatcc_builder_t *B
257 , const c_object_t *obj) {
258     if (!obj) return 0;
259     // 1. Build all nested parts first (
260     strings, vectors).
261     flatbuffers_string_ref_t id_ref =
262     flatcc_builder_create_string_str
263     (B, obj->object_id);
264     flatbuffers_string_ref_t table_ref =
265     flatcc_builder_create_string_
266     str(B, obj->table_name);
267     flatbuffers_vec_ref_t fields_vec =
268     0;
269     c_field_t *p = obj->fields;
270     if (p) {
271         objstore_Field_vec_start(&
272         builder);
273         while (p) {
274             objstore_Field_vec_push(&
275             builder, create_fb_field
276             (&builder, p));
277             p = p->next;
278         }
279         fields_vec = objstore_Field_vec_
280         end(&builder);
281     }
282     objstore_Object_start_as_root(&
283     builder);
284     objstore_Object_object_id_add(&
285     builder, id_ref);
286     objstore_Object_table_name_add(&
287     builder, table_ref);
288     if (fields_vec) {
289         // 2. Manually assemble the table
290         using low-level primitives.
291         // This avoids the problematic '
292         create' convenience function.
293         objstore_Object_start(B);
294         objstore_Object_object_id_add(B, id_
295         ref);
296         objstore_Object_table_name_add(B,
297         table_ref);
298         if (fields_vec) {
299             objstore_Object_fields_add(B,
300             fields_vec);
301         }
302         return objstore_Object_end(B);
303     }
304     // The main serialize_c_object function
305     is now correct because its helper is
306     safe.
307     fb_serializer_status_t serialize_c_
308     object(const c_object_t *obj, void
309     **out_buffer, size_t *out_size) {
310         flatcc_builder_t builder;
311         if (!obj || !out_buffer || !out_size
312         ) return FB_SERIALIZER_ERROR_
313         INVALID_INPUT;
314         flatcc_builder_init(&builder);
315         // This is the only guaranteed
316         correct and safe way to build
317         the object
318         // without risking state corruption
319         that leads to silent data loss.
320         *out_buffer = 0;
321         *out_size = 0;
322         flatcc_builder_reset(&builder);
323         // Manually build the root object
324         from primitives.
325         flatbuffers_string_ref_t id_ref =
326         flatcc_builder_create_string_str
327         (&builder, obj->object_id);
328         flatbuffers_string_ref_t table_ref =
329         flatcc_builder_create_string_
330         str(&builder, obj->table_name);
331         flatbuffers_vec_ref_t fields_vec =
332         0;
333         c_field_t *p = obj->fields;
334         if (p) {
335             objstore_Field_vec_start(&
336             builder);
337             while (p) {
338                 objstore_Field_vec_push(&
339                 builder, create_fb_field
340                 (&builder, p));
341                 p = p->next;
342             }
343             fields_vec = objstore_Field_vec_
344             end(&builder);
345         }
346         objstore_Object_start_as_root(&
347         builder);
348         objstore_Object_object_id_add(&
349         builder, id_ref);
350         objstore_Object_table_name_add(&
351         builder, table_ref);
352         if (fields_vec) {

```



```

301         objstore_Object_fields_add(&
            builder, fields_vec);
302     }
303     objstore_Object_end_as_root(&builder
        );
304
305     *out_buffer = flatcc_builder_
        finalize_aligned_buffer(&builder
            , out_size);
306     flatcc_builder_clear(&builder);
307
308     if (!*out_buffer) {
309         return FB_SERIALIZER_ERROR_
            ALLOCATION;
310     }
311     return FB_SERIALIZER_OK;
312 }
313
314 // --- Deserialization Implementation
    ---
315
316 static fb_serializer_status_t parse_fb_
    field_value_union(
317     objstore_FieldValue_union_t fb_union_
        struct,
318     flatbuffers_union_type_t fb_union_
        type_tag,
319     c_field_value_t **out_fv) {
320
321     *out_fv = (c_field_value_t*)calloc
        (1, sizeof(c_field_value_t));
322     if (!*out_fv) return FB_SERIALIZER_
        ERROR_ALLOCATION;
323
324     (*out_fv)->type = (c_field_value_
        type_t)fb_union_type_tag;
325     const void* fb_union_table_ptr = fb_
        union_struct.value;
326
327     objstore_Object_table_t nested_fb_
        obj_temp;
328     fb_serializer_status_t status_temp;
329     c_field_t *head_field = NULL, *tail_
        field = NULL;
330     c_map_pair_t *head_map_pair = NULL,
        *tail_map_pair = NULL;
331
332     if (fb_union_type_tag != objstore_
        FieldValue_NONE && !fb_union_
        table_ptr) {
333         free(*out_fv); *out_fv = NULL;
            return FB_SERIALIZER_ERROR_
                INVALID_INPUT;
334     }
335
336     switch (fb_union_type_tag) {
337     case objstore_FieldValue_NONE:
338         break;
339     case objstore_FieldValue_
        IntValue:
340         (*out_fv)->data.int_val =
            objstore_IntValue_value
            ((objstore_IntValue_
                table_t)fb_union_table_
                ptr); break;
341     case objstore_FieldValue_
        BoolValue:
342         (*out_fv)->data.bool_val =
            objstore_BoolValue_value
            ((objstore_BoolValue_
                table_t)fb_union_table_
                ptr); break;
343     case objstore_FieldValue_
        FloatValue:
344         (*out_fv)->data.float_val =
            objstore_FloatValue_
            value((objstore_
                FloatValue_table_t)fb_
                union_table_ptr); break;
345     case objstore_FieldValue_
        StringValue:
346         (*out_fv)->data.string_val =
            ansi_strdup(objstore_
                StringValue_value((
                objstore_StringValue_
                table_t)fb_union_table_
                ptr));
            if (!(*out_fv)->data.string_
                val) { free(*out_fv); *
                out_fv = NULL; return FB
                SERIALIZER_ERROR_
                ALLOCATION; } break;
347     case objstore_FieldValue_
        PointerValue:
348         (*out_fv)->data.string_val =
            ansi_strdup(objstore_
                PointerValue_object_id((
                objstore_PointerValue_
                table_t)fb_union_table_
                ptr));
            if (!(*out_fv)->data.string_
                val) { free(*out_fv); *
                out_fv = NULL; return FB
                SERIALIZER_ERROR_
                ALLOCATION; } break;
349     case objstore_FieldValue_
        ObjectValue:
350         nested_fb_obj_temp =
            objstore_ObjectValue_
            object((objstore_
                ObjectValue_table_t)fb_
                union_table_ptr);
            if (nested_fb_obj_temp) {
351                 status_temp = parse_fb_
                    object_recursive(
                        nested_fb_obj_temp,
                        &((*out_fv)->data.
                            object_val));
                    if (status_temp != FB_
                        SERIALIZER_OK) {
352                        free(*out_fv); *out_
                            fv = NULL; return
                            status_temp; }
353            }
            break;
354     case objstore_FieldValue_
        ArrayValue:
355         {
356             objstore_Field_vec_t vec =
                objstore_ArrayValue_
                values((objstore_
                    ArrayValue_table_t)fb_
                    union_table_ptr);
357             if (vec) {
358                 for (size_t i = 0; i <
                    objstore_Field_vec_
                    len(vec); ++i) {
359                     c_field_t *new_item
                        = NULL;
                        status_temp = parse_

```

	fb_field(objstore_Field_ vec_at(vec, i), &new_item);	383	_ALLOCATION; }	
364	if (status_temp != FB_SERIALIZER_OK) { free_c_field_ _list(head_field_); free(*out_fv) ; *out_fv = NULL ; return status_ temp; }	384	objstore_FieldValue_ union_t u = objstore_MapPair_ _value_union(pair_table);	
	if (!head_field)	385	status_temp = parse_ fb_field_value_ union(u, u.type, &new_pair-> value);	
365	head_field = tail_field = new _item;	386	if (status_temp != FB_SERIALIZER_OK) { free(new_ pair->key); free (new_pair); free _c_map_pair_list (head_map_pair); free(*out_fv); *out_fv = NULL; return status_ temp; }	
366	else { tail_field-> next = new_item; tail_field = new_item; }			
367	}			
368	}			
369	(*out_fv)->data.array_values = head_field;	387		
370	break;	388	if (!head_map_pair)	
371	}		head_map_pair = tail_map_pair = new_pair;	
372	case objstore_FieldValue_ MapValue:		else { tail_map_pair ->next = new_ pair; tail_map_ pair = new_pair; }	
373	{	389		
374	objstore_MapPair_vec_t vec = objstore_MapValue_pairs ((objstore_MapValue_ table_t)fb_union_table_ ptr);	390	}	
375	if (vec) {	391	}	
376	for (size_t i = 0; i < objstore_MapPair_vec _len(vec); ++i) {	392	(*out_fv)->data.map_pairs = head_map_pair;	
377	objstore_MapPair_ table_t pair_ table = objstore _MapPair_vec_at(vec, i);	393	break;	
	c_map_pair_t* new_ pair = (c_map_ pair_t*)calloc (1, sizeof(c_map_ pair_t));	394	}	
378	if (!new_pair) { free_c_map_pair_ list(head_map_ pair); free(*out_ fv); *out_fv = NULL; return FB_ SERIALIZER_ERROR_ _ALLOCATION; }	395	default: free(*out_fv); *out_fv = NULL; return FB_SERIALIZER_ _ERROR_TYPE_MISMATCH;	
379		396	}	
	new_pair->key = ansi_ strdup(objstore _MapPair_key(pair_table));	397	return FB_SERIALIZER_OK;	
380	if (!new_pair->key) { free(new_pair) ; free_c_map_ pair_list(head_ map_pair); free (*out_fv); *out_ fv = NULL; return FB_ SERIALIZER_ERROR	398	}	
381		399	}	
382		400	static fb_serializer_status_t parse_fb_ field(objstore_Field_table_t fb_ field_table, c_field_t **out_c_field) {	
		401	flatbuffers_union_type_t val_type_ tag;	
		402	objstore_FieldValue_union_t val_ union_struct;	
		403	fb_serializer_status_t status;	
		404		
		405	*out_c_field = (c_field_t*)calloc(1, sizeof(c_field_t));	
		406	if (!*out_c_field) return FB_ SERIALIZER_ERROR_ALLOCATION;	
		407		
		408	(*out_c_field)->name = ansi_strdup(objstore_Field_name(fb_field_ table));	
		409	if (!(*out_c_field)->name) { free(* out_c_field); *out_c_field = NULL; return FB_SERIALIZER_ERROR_ _ALLOCATION; }	
		410		
		411	val_union_struct = objstore_Field_ _value_union(pair_table);	

```

412     value_union(fb_field_table);
413     val_type_tag = val_union_struct.type;
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     }
486
487     (*out_obj_param)->fields = head;
488     return FB_SERIALIZER_OK;
489 }
490
491 fb_serializer_status_t deserialize_to_c_
492 object(const void *buffer, size_t
493 size, c_object_t **out_obj) {
494     if (!buffer || size == 0 || !out_obj
495     ) {
496         return FB_SERIALIZER_ERROR_
497         INVALID_INPUT;
498     }
499
500     int verify_ret = objstore_Object_
501     verify_as_root_with_identifier(
502     buffer, size, 0);
503     if (verify_ret != flatcc_verify_ok)
504     {
505         fprintf(stderr, "FlatBuffer_
506         verification_failed: %s\n",
507         flatcc_verify_error_string(
508         verify_ret));
509         return FB_SERIALIZER_ERROR_
510         FLATCC_VERIFIER;
511     }
512
513     objstore_Object_table_t root_table =
514     objstore_Object_as_root(buffer)
515     ;
516     if (!root_table) {
517         return FB_SERIALIZER_ERROR_
518         INVALID_INPUT;
519     }
520
521     return parse_fb_object_recursive(
522     root_table, out_obj);
523 }
524
525 // --- JSON Transformation Logic ---
526 static fb_serializer_status_t c_object_
527 from_cjson_node(cJSON *json_node,
528 const char *default_object_id, const
529 char *default_table_name, c_object_
530 t **out_obj) {
531     if (!cJSON_IsObject(json_node))
532         return FB_SERIALIZER_ERROR_TYPE_
533         MISMATCH;
534     *out_obj = NULL;
535
536     c_object_t *obj = (c_object_t*)
537     calloc(1, sizeof(c_object_t));
538     if (!obj) return FB_SERIALIZER_ERROR_
539     ALLOCATION;
540
541     fb_serializer_status_t status = FB_
542     SERIALIZER_OK;
543     cJSON *json_val;
544
545     json_val = cJSON_
546     GetObjectItemCaseSensitive(json_
547     node, "object_id");
548     if (cJSON_IsString(json_val) && json_
549     _val->valuestring) {
550         obj->object_id = ansi_strdup(
551         json_val->valuestring);
552     } else if (default_object_id) {
553         obj->object_id = ansi_strdup(

```

```

487         default_object_id);
488     }
489     if (!obj->object_id) { status = FB_
519         } else {
520             current_c_field_tail->
521                 next = new_c_field;
522             }
523             current_c_field_tail = new_c_
524                 _field;
525         }
526         child_item = child_item->next;
527     }
528     *out_obj = obj;
529     return FB_SERIALIZER_OK;
530
531     DYNAMIC_JSON_PARSE_ERROR_CLEANUP:
532     free_c_object(obj);
533     *out_obj = NULL;
534     return status;
535 }
536
537 static c_field_value_t* c_field_value_
538     from_cjson_recursive(cJSON *item,
539         const char* parent_key_for_list_
540         items, fb_serializer_status_t *
541         current_status) {
542     if (*current_status != FB_SERIALIZER_
543         OK) return NULL;
544     if (!item) { *current_status = FB_
545         SERIALIZER_ERROR_INVALID_INPUT;
546         return NULL; }
547
548     c_field_value_t *fv = (c_field_value_
549         t*)calloc(1, sizeof(c_field_
550         value_t));
551     if (!fv) { *current_status = FB_
552         SERIALIZER_ERROR_ALLOCATION;
553         return NULL; }
554
555     if (cJSON_IsNull(item)) {
556         fv->type = C_FIELD_VALUE_TYPE_
557             NONE;
558     } else if (cJSON_IsBool(item)) {
559         fv->type = C_FIELD_VALUE_TYPE_
560             BOOL_VALUE;
561         fv->data.bool_val = cJSON_IsTrue
562             (item) ? true : false;
563     } else if (cJSON_IsNumber(item)) {
564         double num = item->valuedouble;
565         if (num == floor(num) && num >=
566             -9007199254740991.0 && num
567             <= 9007199254740991.0) {
568             fv->type = C_FIELD_VALUE_
569                 TYPE_INT_VALUE;
570             fv->data.int_val = (int64_t)
571                 num;
572         } else {
573             fv->type = C_FIELD_VALUE_
574                 TYPE_FLOAT_VALUE;
575             fv->data.float_val = (float)
576                 num;
577         }
578     } else if (cJSON_IsString(item)) {
579         size_t key_len = parent_key_for_
580             list_items ? strlen(parent_
581             key_for_list_items) : 0;
582         if (key_len >= 4 && strcmp(
583             parent_key_for_list_items +
584             key_len - 4, "_ref") == 0) {
585             fv->type = C_FIELD_VALUE_
586                 TYPE_POINTER_VALUE;
587         } else {
588             fv->type = C_FIELD_VALUE_
589                 TYPE_STRING_VALUE;
590             fv->data.string_val = cJSON_
591                 GetStringValue(item);
592         }
593     }
594     return fv;
595 }
596
597 cJSON *fb_serializer_serialize(fb_objec
598     t *obj, fb_serializer_status_t *cur
599     rent_status) {
600     cJSON *root = cJSON_CreateObject();
601     if (!root) { *current_status = FB_
602         SERIALIZER_ERROR_ALLOCATION;
603         return NULL; }
604     if (!obj->object_id) { *current_statu
605         s = FB_SERIALIZER_ERROR_INVALID_
606         INPUT; return NULL; }
607     obj->object_id = cJSON_GetIntKeyVa
608         lue(root, "id");
609     if (!obj->table_name) { *current_stat
610         us = FB_SERIALIZER_ERROR_INVALID_
611         INPUT; return NULL; }
612     obj->table_name = cJSON_GetStringValue
613         (root, "table_name");
614     if (!obj->fields) { *current_status =
615         FB_SERIALIZER_ERROR_INVALID_INPU
616         T; return NULL; }
617     cJSON *fields = cJSON_CreateArray();
618     if (!fields) { *current_status = FB_
619         SERIALIZER_ERROR_ALLOCATION;
620         return NULL; }
621     for (c_field_t *field = obj->fields;
622         field; field = field->next) {
623         cJSON *field_obj = cJSON_CreateOb
624             j();
625         if (!field_obj) { *current_statu
626             s = FB_SERIALIZER_ERROR_ALLO
627             CATION; continue; }
628         field_obj->string = field->name;
629         if (!field->value) { *current_statu
630             s = FB_SERIALIZER_ERROR_INVA
631             LID_INPUT; continue; }
632         field_obj->value = c_field_value_
633             from_cjson_recursive(field->
634             value, field->name, current_
635             status);
636         if (*current_status != FB_SERIALI
637             ZER_OK) { cJSON_Delete(field_
638                 obj); continue; }
639         cJSON_AddItemToArray(fields, field_
640             obj);
641     }
642     return root;
643 }
644
645 void fb_serializer_free(fb_object *obj) {
646     if (!obj) return;
647     cJSON_Delete(obj->root);
648     free(obj->fields);
649     free(obj->table_name);
650     free(obj->object_id);
651     free(obj);
652 }

```

```

562         fv->type = C_FIELD_VALUE_
                    TYPE_STRING_VALUE;
563     }
564     fv->data.string_val = ansi_
                    strdup(item->valstring);
565     if (!fv->data.string_val && item
566         ->valstring) {
567         free(fv); *current_status =
                    FB_SERIALIZER_ERROR_
                    ALLOCATION; return NULL;
568     }
569     } else if (cJSON_IsArray(item)) {
570         fv->type = C_FIELD_VALUE_TYPE_
                    ARRAY_VALUE;
571         c_field_t *head = NULL, *tail =
                    NULL;
572         cJSON *elem = NULL;
573         int index = 0;
574         cJSON_ArrayForEach(elem, item) {
575             char synthetic_name[32];
576             snprintf(synthetic_name,
577                     sizeof(synthetic_name),
578                     "item_%d", index++);
579
580             c_field_value_t* item_fv = c_
                    _field_value_from_cjson_
                    recursive(elem,
581                             synthetic_name, current_
                    status);
582             if (*current_status != FB_
                    SERIALIZER_OK) { free_c_
                    _field_list(head); free
                    (fv); return NULL; }
583
584             c_field_t* new_item_field =
                    (c_field_t*)calloc(1,
585                                         sizeof(c_field_t));
586             if(!new_item_field) { free_c_
                    _field_value(item_fv);
587             free_c_field_list(head);
588             free(fv); *current
                    status = FB_SERIALIZER_
                    ERROR_ALLOCATION; return
                    NULL; }
589
590             new_item_field->name = ansi_
                    strdup(synthetic_name);
591             if(!new_item_field->name) {
592                 free(new_item_field);
593                 free_c_field_value(item_
                    fv); free_c_field_list(
                    head); free(fv); *
                    current_status = FB_
                    SERIALIZER_ERROR_
                    ALLOCATION; return NULL
                    ; }
594
595             new_item_field->value = item_
                    _fv;
596
597             if (!head) head = tail = new
                    _item_field;
598             else { tail->next = new_item_
                    field; tail = new_item_
                    field; }
599         }
600         fv->data.array_values = head;
601     } else if (cJSON_IsObject(item)) {
602         cJSON *obj_id_json = cJSON_
                    GetObjectItemCaseSensitive(

```

```

                    item, "object_id");
603         cJSON *tbl_name_json = cJSON_
                    GetObjectItemCaseSensitive(
                    item, "table_name");
604
605         if (cJSON_IsString(obj_id_json)
606             && obj_id_json->valstring
607             &&
608             cJSON_IsString(tbl_name_json
609                             ) && tbl_name_json->
                    valstring) {
610             fv->type = C_FIELD_VALUE_
                    TYPE_OBJECT_VALUE;
611             *current_status = c_object_
                    from_cjson_node(item,
                    NULL, NULL, &fv->data.
                    object_val);
612             if (*current_status != FB_
                    SERIALIZER_OK) {
613                 free(fv); return NULL;
614             }
615         } else {
616             fv->type = C_FIELD_VALUE_
                    TYPE_MAP_VALUE;
617             c_map_pair_t *head = NULL, *
                    tail = NULL;
618             cJSON *pair_item_json = item
                    ->child;
619             while(pair_item_json) {
620                 if (!pair_item_json->
                    string) {
621                     pair_item_json =
                    pair_item_json->
                    next;
622                     continue;
623                 }
624                 c_map_pair_t *new_pair =
                    (c_map_pair_t*)
                    calloc(1, sizeof(c_
                    map_pair_t));
625                 if(!new_pair) { free_c_
                    map_pair_list(head);
626                 free(fv); *current_
                    status = FB_
                    SERIALIZER_ERROR_
                    ALLOCATION; return
                    NULL; }
627
628                 new_pair->key = ansi_
                    strdup(pair_item_
                    json->string);
629                 if(!new_pair->key) {free
                    (new_pair); free_c_
                    map_pair_list(head);
630                 free(fv); *current_
                    status = FB_
                    SERIALIZER_ERROR_
                    ALLOCATION; return
                    NULL; }
631
632                 new_pair->value = c_
                    _field_value_from_
                    cJSON_recursive(pair_
                    item_json, pair_
                    item_json->string,
                    current_status);
633                 if(*current_status != FB_
                    SERIALIZER_OK) {
634                     free(new_pair->key);
635                     free(new_pair);

```



```

        free_c_map_pair
        list(head);
        free(fv);
        return NULL;
    }
    if (!head) head = tail =
        new_pair;
    else { tail->next = new_
        pair; tail = new_
        pair; }
    pair_item_json = pair_
        item_json->next;
    }
    fv->data.map_pairs = head;
}
} else {
    fprintf(stderr, "Unsupported_
    cJSON_type_for_field_value:_
    %d\n", item->type);
    free(fv);
    *current_status = FB_SERIALIZER_
        ERROR_TYPE_MISMATCH;
    return NULL;
}
return fv;
}
fb_serializer_status_t fb_object_from_
    json_string(const char *json_string,
    const char *default_object_id,
    const char *default_table_name, c_
    object_t **out_c_obj) {
    if (!json_string || !out_c_obj)
        return FB_SERIALIZER_ERROR_
            INVALID_INPUT;
    *out_c_obj = NULL;
    cJSON *root_json = cJSON_Parse(json_
        string);
    if (!root_json) {
        return FB_SERIALIZER_ERROR_JSON_
            PARSE;
    }
    if (!cJSON_IsObject(root_json)){
        cJSON_Delete(root_json);
        return FB_SERIALIZER_ERROR_TYPE_
            MISMATCH;
    }
    fb_serializer_status_t status = c_
        object_from_cjson_node(root_json
        , default_object_id, default_
        table_name, out_c_obj);
    cJSON_Delete(root_json);
    return status;
}

```

Listing 16: The implementation of the serialization and deserialization logic (src/object_store/core/fb_serializer.c)

The Object Store API ('object_store'): This module provides the high-level, user-friendly API for database interaction. The 'object_store.h' header declares the primary functions for manag-

ing the database and performing CRUD operations.

```

1 #ifndef OBJECT_STORE_H
2 #define OBJECT_STORE_H
3
4 #include "fb_serializer.h" // For status
    enum and c_object_t
5 #include <lmdb.h> // For MDB_
    env, MDB_dbi
6 #include <stddef.h>
7 #include <stdint.h>
8 #include <stdbool.h>
9
10 // --- Portability Defines for Threading
    ---
11 // Selects the appropriate threading
    primitive based on the operating
    system.
12 #if defined(_WIN32) && !defined(__CYGWIN
    )
13 #include <windows.h>
14 #define OS_USE_WINDOWS_CRITICAL_
    SECTION
15 #elif defined(__linux__) || defined(__
    APPLE__) || defined(__FreeBSD__) ||
    defined(__NetBSD__) || defined(__
    OpenBSD__) || defined(__sun) ||
    defined(__QNXNTO__)
16 #include <pthread.h>
17 #define OS_USE_PTHREADS
18 #endif
19
20 // =====
21 // SECTION 1: CONSTANTS
22 // =====
23
24 // The fixed size of a key in the
    database.
25 // 8 bytes (inverted timestamp) + 16
    bytes (MD5 hash) + 8 bytes (nonce)
26 #define OS_MAX_KEY_SIZE 32
27 // Default size for the LMDB memory map
    (1 GiB).
28 #define OS_DEFAULT_MAPSIZE (1024 * 1024
    * 1024)
29 // Default maximum number of named
    databases (tables) within the
    environment.
30 #define OS_DEFAULT_MAX_DBS 128
31
32 // =====
33 // SECTION 2: CORE STRUCTURES
34 // =====
35
36 /**
37  * @struct os_table_info_t
38  * @brief Caches the name and LMDB
    handle (DBI) of an opened table.
39  */
40 typedef struct {
41     char* name;
42     MDB_dbi dbi;
43     bool valid;
44 } os_table_info_t;
45
46 /**
47  * @struct object_store_t
48  * @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
78 int object_store_create(object_store_t
**store_ptr);
79
80 /**
81  * @brief Closes the database and frees
all resources associated with an
object store handle.
82  * @param store_ptr A pointer to the
object_store_t pointer to be freed
and set to NULL.
83  */
84 void object_store_destroy(object_store_t
**store_ptr);
85
86 /**
87  * @brief Initializes the LMDB
environment for the object store.
88  * @param store The object store handle.
89  * @param path The filesystem path where
the database will be stored.
90  * @param mapsize The maximum size of
the database file. Pass 0 for
default.
91  * @param maxdbs The maximum number of
tables. Pass 0 for default.
92  * @return FB_SERIALIZER_OK on success,
or an error code on failure.
93  */
94 fb_serializer_status_t object_store_init
db(object_store_t *store, const
char *path, size_t mapsize, unsigned
int maxdbs);
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
139 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);
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  *        if the key did not exist.
178  */
179 fb_serializer_status_t object_store_get_
180 object(object_store_t *store, const
181 char *table_name, const unsigned
182 char key[OS_MAX_KEY_SIZE], c_object_
183 t **out_value);
184
185 /**
186  * @brief Deletes an object from the
187  *        database.
188  * @param store The object store handle.
189  * @param table_name The name of the
190  *        table.
191  * @param key The key of the object to
192  *        delete.
193  * @return FB_SERIALIZER_OK on success,
194  *        FB_SERIALIZER_ERROR_LMDB_NOT_FOUND
195  *        if the key did not exist.
196  */
197 fb_serializer_status_t object_store_
198 delete_object(object_store_t *store,
199 const char *table_name, const
200 unsigned char key[OS_MAX_KEY_SIZE]);
201
202 /**
203  * @brief Parses a JSON string and puts
204  *        the resulting object into the
205  *        database.
206  * @param store The object store handle.
207  * @param table_name The name of the
208  *        table.
209  * @param key The unique key for the
210  *        object.
211  * @param object_json_string The null-
212  *        terminated JSON string to parse and
213  *        store.
214  * @return FB_SERIALIZER_OK on success,
215  *        or an error code.
216  */
217 fb_serializer_status_t object_store_put_
218 object_from_json(
219 object_store_t *store,
220 const char *table_name,
221 const unsigned char key[OS_MAX_KEY_
222 SIZE],
223 const char *object_json_string);
224
225 #endif // OBJECT_STORE_H

```

```

175 * @param object_json_string The null-
176 *        terminated JSON string to parse and
177 *        store.
178 * @return FB_SERIALIZER_OK on success,
179 *        or an error code.
180 */
181 fb_serializer_status_t object_store_put_
182 object_from_json(
183 object_store_t *store,
184 const char *table_name,
185 const unsigned char key[OS_MAX_KEY_
186 SIZE],
187 const char *object_json_string);
188
189 #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(__APPLE__) || defined(__FreeBSD__) || defined(__NetBSD__) || defined(__OpenBSD__) || defined(__sun) || defined(__QNXNTO__)
11 #ifndef POSIX_C_SOURCE
12 #define POSIX_C_SOURCE 200809L
13 #endif
14 #endif
15
16 #include "object_store.h"
17 #include "fb_serializer.h"
18 #include "md5.h"
19
20 #include <stdio.h>
21 #include <stdlib.h>
22 #include <string.h>
23 #include <time.h>
24 #include <errno.h>
25
26 #if defined(_WIN32) && !defined(__CYGWIN__)
27 #include <windows.h>
28 #include <wincrypt.h>
29 #pragma comment(lib, "Advapi32.lib")
30 #include <direct.h>

```

```

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 -\n
41             malloc failed");
42         return NULL;
43     }
44     memcpy(new_s, s, len);
45     return new_s;
46 }
47
48 // --- Helper Function for LMDB Errors ---
49 static void print_lmdb_error(const char
50     *func_name, int rc) {
51     fprintf(stderr, "LMDB ERROR in %s: %s\n", func_name, mdb_strerror(rc), rc);
52 }
53
54 // --- Table DBI Management (Thread-Safe Cache Access) ---
55 static fb_serializer_status_t get_table_dbi(object_store_t *store, const char *name, MDB_dbi *dbi_out, bool create_if_not_exists) {
56     if (!store || !store->env || !name || !dbi_out) return FB_SERIALIZER_ERROR_INVALID_INPUT;
57
58     #if defined(OS_USE_PTHREADS)
59         pthread_mutex_lock(&store->table_cache_mutex);
60     #elif defined(OS_USE_WINDOWS_CRITICAL_SECTION)
61         EnterCriticalSection(&store->table_cache_mutex);
62     #endif
63
64     for (size_t i = 0; i < store->num_tables; ++i) {
65         if (store->tables[i].valid && strcmp(store->tables[i].name, name) == 0) {
66             *dbi_out = store->tables[i].dbi;
67             #if defined(OS_USE_PTHREADS)
68                 pthread_mutex_unlock(&store->table_cache_mutex);
69             #elif defined(OS_USE_WINDOWS_CRITICAL_SECTION)
70                 LeaveCriticalSection(&store->table_cache_mutex);
71             #endif
72             return FB_SERIALIZER_OK;
73         }
74     }

```

```

74     #if defined(OS_USE_PTHREADS)
75         pthread_mutex_unlock(&store->table_cache_mutex);
76     #elif defined(OS_USE_WINDOWS_CRITICAL_SECTION)
77         LeaveCriticalSection(&store->table_cache_mutex);
78     #endif
79
80     MDB_txn *txn = NULL;
81     int rc = mdb_txn_begin(store->env, NULL, create_if_not_exists ? 0 : MDB_RDONLY, &txn);
82     if (rc != MDB_SUCCESS) {
83         print_lmdb_error("get_table_dbi (mdb_txn_begin)", rc);
84         return FB_SERIALIZER_ERROR_LMDB_TXN_BEGIN;
85     }
86
87     rc = mdb_dbi_open(txn, name, create_if_not_exists ? MDB_CREATE : 0, dbi_out);
88     if (rc != MDB_SUCCESS) {
89         mdb_txn_abort(txn);
90         if (rc == MDB_NOTFOUND && !create_if_not_exists) return FB_SERIALIZER_ERROR_LMDB_NOT_FOUND;
91         print_lmdb_error("get_table_dbi (mdb_dbi_open)", rc);
92         return FB_SERIALIZER_ERROR_LMDB_DBI_OPEN;
93     }
94
95     if (create_if_not_exists) {
96         rc = mdb_txn_commit(txn);
97         if (rc != MDB_SUCCESS) {
98             print_lmdb_error("get_table_dbi (mdb_txn_commit_for_create)", rc);
99             return FB_SERIALIZER_ERROR_LMDB_TXN_COMMIT;
100         }
101     } else {
102         mdb_txn_abort(txn);
103     }
104
105     #if defined(OS_USE_PTHREADS)
106         pthread_mutex_lock(&store->table_cache_mutex);
107     #elif defined(OS_USE_WINDOWS_CRITICAL_SECTION)
108         EnterCriticalSection(&store->table_cache_mutex);
109     #endif
110
111     for (size_t i = 0; i < store->num_tables; ++i) {
112         if (store->tables[i].valid && strcmp(store->tables[i].name, name) == 0) {
113             goto get_table_dbi_cleanup;
114         }
115     }
116
117     if (store->num_tables >= store->tables_capacity) {
118         size_t new_capacity = (store->tables_capacity == 0) ? 16 :

```

```

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].
143         name = 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     }
163     get_table_dbi_cleanup:
164     #if defined(OS_USE_PTHREADS)
165         pthread_mutex_unlock(&store->table_
166             cache_mutex);
167     #elif defined(OS_USE_WINDOWS_CRITICAL_
168         SECTION)
169         LeaveCriticalSection(&store->table_
170             cache_mutex);
171     #endif
172     return FB_SERIALIZER_OK;
173 }
174
175 int object_store_create(object_store_t
176     **store_ptr) {
177     if (!store_ptr) return -1;
178     *store_ptr = (object_store_t*)calloc
179         (1, sizeof(object_store_t));
180     if (!*store_ptr) return -1;
181     #if defined(OS_USE_PTHREADS)
182         if (pthread_mutex_init(&(*store_ptr)
183             ->table_cache_mutex, NULL) != 0)
184         {
185             perror("Failed to initialize
186                 table_cache_mutex");
187         }
188     #endif
189     free(*store_ptr);
190     *store_ptr = NULL;
191     return -1;
192 }
193
194 #if defined(OS_USE_WINDOWS_CRITICAL_
195     SECTION)
196     InitializeCriticalSection(&(*store_
197         ptr)->table_cache_mutex);
198 #endif
199 return 0;
200 }
201
202 void object_store_destroy(object_store_t
203     **store_ptr) {
204     if (!store_ptr || !*store_ptr)
205         return;
206     object_store_t *store = *store_ptr;
207     if (store->env) {
208         mdb_env_close(store->env);
209     }
210     if (store->tables) {
211         for (size_t i = 0; i < store->
212             num_tables; ++i) {
213             free(store->tables[i].name);
214         }
215         free(store->tables);
216     }
217     #if defined(OS_USE_PTHREADS)
218         pthread_mutex_destroy(&store->table_
219             cache_mutex);
220     #elif defined(OS_USE_WINDOWS_CRITICAL_
221         SECTION)
222         DeleteCriticalSection(&store->table_
223             cache_mutex);
224     #endif
225     free(store);
226     *store_ptr = NULL;
227 }
228
229 fb_serializer_status_t object_store_init
230     _db(object_store_t *store, const
231     char *path, size_t mapsize, unsigned
232     int maxdbs) {
233     if (!store || !path) return FB_
234         SERIALIZER_ERROR_INVALID_INPUT;
235     if (store->env) return FB_SERIALIZER_
236         ERROR_INVALID_INPUT;
237     #if defined(WIN32) && !defined(_
238         CYGWIN_)
239         if (!CreateDirectoryA(path, NULL
240             ) && GetLastError() != ERROR_
241             _ALREADY_EXISTS) {
242             return FB_SERIALIZER_ERROR_
243                 SYSTEM_CALL;
244         }
245     #else
246         if (mkdir(path, 0755) != 0 &&
247             errno != EEXIST) {
248             return FB_SERIALIZER_ERROR_
249                 SYSTEM_CALL;
250         }
251     #endif
252     int rc = mdb_env_create(&store->env)
253         ;
254     if (rc != MDB_SUCCESS) {
255         print_lmdb_error("mdb_env_create
256             ", rc);
257     }

```



```

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

```

```

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         RDONLY);
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
330 fb_serializer_status_t object_store_
331 generate_key(unsigned char key_
332     buffer[OS_MAX_KEY_SIZE]) {
333     if (!key_buffer) return FB_SERIALIZER_
334         ERROR_INVALID_INPUT;
335
336     uint64_t inverted_timestamp = UINT64
337         _MAX - os_time_nanoseconds();
338     for (int i = 0; i < 8; ++i) {
339         key_buffer[i] = (unsigned char)
340             ((inverted_timestamp >> (56
341                 - i * 8)) & 0xFF);
342     }
343
344     unsigned char random_data[16];
345     generate_random_bytes(random_data,
346         16);
347     md5_state_t md5_ctx;
348     md5_init(&md5_ctx);
349     md5_append(&md5_ctx, random_data,
350         16);
351     md5_finish(&md5_ctx, key_buffer + 8)
352         ;
353
354     generate_random_bytes(key_buffer +
355         24, 8);
356
357     return FB_SERIALIZER_OK;
358 }
359
360 static fb_serializer_status_t do_put(
361     object_store_t *store, const char *
362     table_name, const unsigned char key[
363     OS_MAX_KEY_SIZE], const c_object_t *
364     value, unsigned int flags) {
365     if (!store || !table_name || !key ||
366         !value) return FB_SERIALIZER_
367         ERROR_INVALID_INPUT;
368     if (!store->env) return FB_
369
370     SERIALIZER_ERROR_NOT_INITIALIZED
371     ;
372
373     MDB_dbi dbi;
374     fb_serializer_status_t status = get_
375         table_dbi(store, table_name, &
376         dbi, (flags & MDB_NOOVERWRITE) ?
377         false : true);
378     if (status != FB_SERIALIZER_OK)
379         return status;
380
381     void *fb_buffer = NULL;
382     size_t fb_size = 0;
383     status = serialize_c_object(value, &
384         fb_buffer, &fb_size);
385     if (status != FB_SERIALIZER_OK) {
386         free(fb_buffer);
387         return status;
388     }
389
390     MDB_txn *txn;
391     MDB_val lmbd_key, lmbd_data;
392     int rc = mdb_txn_begin(store->env,
393         NULL, 0, &txn);
394     if (rc != MDB_SUCCESS) { free(fb_
395         buffer); print_lmbd_error("do_
396         put_␣(mdb_txn_begin)", rc);
397         return FB_SERIALIZER_ERROR_LMDB_
398         TXN_BEGIN; }
399
400     lmbd_key.mv_size = OS_MAX_KEY_SIZE;
401     lmbd_key.mv_data = (void*)key;
402     lmbd_data.mv_size = fb_size;
403     lmbd_data.mv_data = fb_buffer;
404
405     rc = mdb_put(txn, dbi, &lmbd_key, &
406         lmbd_data, flags);
407     if (rc != MDB_SUCCESS) {
408         mdb_txn_abort(txn);
409         free(fb_buffer);
410         if (rc == MDB_KEYEXIST) return
411             FB_SERIALIZER_ERROR_LMDB_KEY_
412             EXISTS;
413         print_lmbd_error("do_put_␣(mdb_
414             put)", rc);
415         return FB_SERIALIZER_ERROR_LMDB_
416             PUT;
417     }
418
419     rc = mdb_txn_commit(txn);
420     free(fb_buffer);
421     if (rc != MDB_SUCCESS) {
422         print_lmbd_error("do_put_␣(mdb_
423             txn_commit)", rc);
424         return FB_SERIALIZER_ERROR_LMDB_
425             TXN_COMMIT;
426     }
427     return FB_SERIALIZER_OK;
428 }
429
430 fb_serializer_status_t object_store_
431 create_object(object_store_t *store,
432     const char *table_name, const
433     unsigned char key[OS_MAX_KEY_SIZE],
434     const c_object_t *value) {
435     return do_put(store, table_name, key
436         , value, MDB_NOOVERWRITE);
437 }
438
439 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) {
389     return do_put(store, table_name, key
        , value, 0);
390 }
391
392 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) {
393     if (!store || !table_name || !key ||
        !out_value) return FB_
        SERIALIZER_ERROR_INVALID_INPUT;
394     *out_value = NULL;
395     if (!store->env) return FB_
        SERIALIZER_ERROR_NOT_INITIALIZED
        ;
396
397     MDB_dbi dbi;
398     fb_serializer_status_t status = get_
        table_dbi(store, table_name, &
        dbi, false);
399     if (status != FB_SERIALIZER_OK)
        return status;
400
401     MDB_txn *txn;
402     MDB_val lmbd_key, lmbd_data;
403     int rc = mdb_txn_begin(store->env,
        NULL, MDB_RDONLY, &txn);
404     if (rc != MDB_SUCCESS) { print_lmbd_
        error("get_object_(mdb_txn_begin
        )", rc); return FB_SERIALIZER_
        ERROR_LMDB_TXN_BEGIN; }
405
406     lmbd_key.mv_size = OS_MAX_KEY_SIZE;
407     lmbd_key.mv_data = (void*)key;
408
409     rc = mdb_get(txn, dbi, &lmbd_key, &
        lmbd_data);
410     mdb_txn_abort(txn);
411
412     if (rc == MDB_NOTFOUND) return FB_
        SERIALIZER_ERROR_LMDB_NOT_FOUND;
413     if (rc != MDB_SUCCESS) { print_lmbd_
        error("get_object_(mdb_get)", rc
        ); return FB_SERIALIZER_ERROR_
        LMDB_GET; }
414
415     return deserialize_to_c_object(lmbd_
        data.mv_data, lmbd_data.mv_size,
        out_value);
416 }
417
418 fb_serializer_status_t object_store_
delete_object(object_store_t *store,
const char *table_name, const
unsigned char key[OS_MAX_KEY_SIZE])
{
419     if (!store || !table_name || !key)
        return FB_SERIALIZER_ERROR_
        INVALID_INPUT;
420     if (!store->env) return FB_
        SERIALIZER_ERROR_NOT_INITIALIZED
        ;
421
422     MDB_dbi dbi;
423     fb_serializer_status_t status = get_
        table_dbi(store, table_name, &
        dbi, false);
        table_dbi(store, table_name, &
        dbi, false);
        if (status != FB_SERIALIZER_OK)
            return status;
424
425         return status;
426
427     MDB_txn *txn;
428     MDB_val lmbd_key;
429     int rc = mdb_txn_begin(store->env,
        NULL, 0, &txn);
430     if (rc != MDB_SUCCESS) { print_lmbd_
        error("delete_object_(mdb_txn_
        begin)", rc); return FB_
        SERIALIZER_ERROR_LMDB_TXN_BEGIN;
        }
431
432     lmbd_key.mv_size = OS_MAX_KEY_SIZE;
433     lmbd_key.mv_data = (void*)key;
434
435     rc = mdb_del(txn, dbi, &lmbd_key,
        NULL);
436     if (rc != MDB_SUCCESS) {
437         mdb_txn_abort(txn);
438         if (rc == MDB_NOTFOUND) return
            FB_SERIALIZER_ERROR_LMDB_NOT_
            FOUND;
439         print_lmbd_error("delete_object_
            (mdb_del)", rc);
440         return FB_SERIALIZER_ERROR_LMDB_
            DEL;
441     }
442
443     rc = mdb_txn_commit(txn);
444     if (rc != MDB_SUCCESS) {
445         print_lmbd_error("delete_object_
            (mdb_txn_commit)", rc);
446         return FB_SERIALIZER_ERROR_LMDB_
            TXN_COMMIT;
447     }
448     return FB_SERIALIZER_OK;
449 }
450
451 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) {
452
453     #ifdef CJSON_ENABLED
454     if (!store || !table_name || !key ||
        !object_json_string) return FB_
        SERIALIZER_ERROR_INVALID_INPUT;
455
456     c_object_t *temp_c_obj = NULL;
457     fb_serializer_status_t status = fb_
        object_from_json_string(object_
        json_string, NULL, NULL, &temp_c_
        obj);
458
459     if (status != FB_SERIALIZER_OK)
        return status;
460     if (!temp_c_obj) return FB_
        SERIALIZER_ERROR_JSON_PARSE;
461
462     status = object_store_put_object(
        store, table_name, key, temp_c_
        obj);
463
464 }

```

```

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_
        INPUT; // JSON support not
        compiled in
476 #endif
477 }

```

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__) ||
   defined(__NetBSD__) || defined(__OpenBSD__) || defined(__sun) ||
   defined(__QNXNTO__)
3     #ifndef _POSIX_C_SOURCE
4         #define _POSIX_C_SOURCE 200809L
5     #endif
6 #endif
7
8 #include "object_store.h"
9 #include "fb_serializer.h"
10 #include <stdio.h>
11 #include <string.h>
12 #include <stdlib.h>
13 #include <stdbool.h>
14 #include <math.h>
15 #include <time.h>
16
17 #ifdef WIN32
18 #include <direct.h>
19 #else
20 #include <sys/stat.h>
21 #include <unistd.h>
22 #endif
23
24 #if defined(OS_USE_PTHREADS)
25 #include <pthread.h>
26 #endif
27
28 #define TEST_DB_PATH "./test_os_db_final

```

```

29 // --- Forward Declarations ---
30 void print_c_object(const c_object_t *
   obj, int indent);
31 bool compare_c_object(const c_object_t *
   o1, const c_object_t * o2);
32 c_object_t* create_test_user_object(
   const char* id_suffix, const char*
   name, int age);
33
34 // --- Test Assertion Macro ---
35 static int tests_passed = 0;
36 static int tests_total = 0;
37 #define ASSERT(condition, message) do {
   \
   tests_total++; \
   if (condition) { \
   tests_passed++; \
   printf("[PASS] %s\n", message);
   \
   } else { \
   printf("[FAIL] %s\n", message);
   \
   } \
   } while (0)
38
39 // --- Test Helper: Cleanup ---
40 void cleanup_test_db_dir() {
41     #if defined(WIN32)
42         _rmdir(TEST_DB_PATH);
43     #else
44         char command[256];
45         sprintf(command, "rm -rf %s",
   TEST_DB_PATH);
46         system(command);
47     #endif
48 }
49
50 // --- Main Test Functions ---
51 void run_basic_crud_tests(object_store_t
   *store) {
52     printf("\n--- Running Basic CRUD
   Tests ---\n");
53     const char* table = "users";
54     ASSERT(object_store_create_table(
   store, table) == FB_SERIALIZER_
   OK, "Create table 'users'");
55
56     unsigned char key1[OS_MAX_KEY_SIZE];
57     ASSERT(object_store_generate_key(
   key1) == FB_SERIALIZER_OK, "
   Generate key1");
58
59     c_object_t* user1 = create_test_user
   _object("001", "Alice", 30);
60     ASSERT(object_store_create_object(
   store, table, key1, user1) == FB
   _SERIALIZER_OK, "Create object
   user1");
61
62     c_object_t* retrieved_user = NULL;
63     ASSERT(object_store_get_object(store
   , table, key1, &retrieved_user)
   == FB_SERIALIZER_OK, "Get object
   user1");
64     ASSERT(retrieved_user != NULL, "
   Retrieved user1 is not null");
65     if (retrieved_user) {
66         ASSERT(compare_c_object(user1,

```

```

    retrieved_user), "Compare_
    original_and_retrieved_user1
    ");
77 }
78 free_c_object(retrieved_user);
79 retrieved_user = NULL;
80
81 c_object_t* user1_updated = create_
    test_user_object("001_upd", "
    Alice_Updated", 31);
82 ASSERT(object_store_put_object(store
    , table, key1, user1_updated) ==
    FB_SERIALIZER_OK, "Update_
    object_user1");
83
84 ASSERT(object_store_get_object(store
    , table, key1, &retrieved_user)
    == FB_SERIALIZER_OK, "Get_
    updated_user1");
85 ASSERT(retrieved_user != NULL, "
    Retrieved_updated_user1_is_not_
    null");
86 if (retrieved_user) {
87     ASSERT(compare_c_object(user1_
        updated, retrieved_user), "
        Compare_updated_and_
        retrieved_user1");
88 }
89 free_c_object(retrieved_user);
90
91 ASSERT(object_store_delete_object(
    store, table, key1) == FB_
    SERIALIZER_OK, "Delete_object_
    user1");
92 ASSERT(object_store_get_object(store
    , table, key1, &retrieved_user)
    == FB_SERIALIZER_ERROR_LMDB_NOT_
    FOUND, "Verify_user1_is_deleted"
    );
93
94 free_c_object(user1);
95 free_c_object(user1_updated);
96 }
97
98 void run_error_condition_tests(object_
    store_t *store) {
99     printf("\n---_Running_Error_
        Condition_Tests_---\n");
100     unsigned char key[OS_MAX_KEY_SIZE];
101     object_store_generate_key(key);
102     c_object_t* dummy_obj = create_test_
        user_object("dummy", "dummy", 0)
        ;
103     c_object_t* retrieved = NULL;
104
105     // FIX: The 'put' command is
        designed to be an "upsert",
        which includes creating
106     // the table if it doesn't exist.
        Therefore, this operation SHOULD
        succeed.
107     ASSERT(object_store_put_object(store
        , "non_existent_table", key,
        dummy_obj) == FB_SERIALIZER_OK,
        "Put_to_non-existent_table_
        should_succeed_by_creating_it");
108
109     // FIX: The 'get' command should
        correctly fail with NOT_FOUND on
        a non-existent table.
110     ASSERT(object_store_get_object(store
        , "another_non_existent_table",
        key, &retrieved) == FB_
        SERIALIZER_ERROR_LMDB_NOT_FOUND,
        "Get_from_non-existent_table_
        fails_with_NOT_FOUND");
111
112     ASSERT(object_store_put_object(NULL,
        "users", key, dummy_obj) == FB_
        SERIALIZER_ERROR_INVALID_INPUT,
        "Put_with_NULL_store_fails");
113
114     free_c_object(dummy_obj);
115 }
116
117 int main() {
118     srand((unsigned int)time(NULL));
119     cleanup_test_db_dir();
120
121     object_store_t *store = NULL;
122     ASSERT(object_store_create(&store)
        == 0, "Object_store_creation");
123     ASSERT(store != NULL, "Store_handle_
        is_not_null");
124
125     ASSERT(object_store_init_db(store,
        TEST_DB_PATH, OS_DEFAULT_MAPSIZE
        , OS_DEFAULT_MAX_DBS) == FB_
        SERIALIZER_OK, "DB_
        Initialization");
126
127     run_basic_crud_tests(store);
128     run_error_condition_tests(store);
129
130     object_store_destroy(&store);
131     ASSERT(store == NULL, "Store_handle_
        is_null_after_destruction");
132
133     printf("\n-----\n
        ");
134     printf("Test_Suite_Finished._Passed:
        _%d/_%d\n", tests_passed, tests_
        total);
135     printf("-----\n
        ");
136
137     cleanup_test_db_dir();
138     return (tests_passed == tests_total
        ? 0 : 1);
139 }
140
141 // Helper Implementations
142 c_object_t* create_test_user_object(
    const char* id_suffix, const char*
    name, int age) {
143     c_object_t* obj = (c_object_t*)
        calloc(1, sizeof(c_object_t));
144     char buf[128];
145     snprintf(buf, sizeof(buf), "user_%s"
        , id_suffix);
146     obj->object_id = strdup(buf);
147     obj->table_name = strdup("users");
148     c_field_t* tail = NULL;
149
150     c_field_t* name_field = (c_field_t*)
        calloc(1, sizeof(c_field_t));
151     name_field->name = strdup("name");
152     name_field->value = create_c_string_
        value(name);
153     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     return obj;
163 }
164 bool compare_c_field_value(const c_field_
165     _value_t* v1, const c_field_value_t*
166     v2);
167 bool compare_c_fields(const c_field_t*
168     f1, const c_field_t* f2) {
169     while(f1 && f2) {
170         if (strcmp(f1->name, f2->name)
171             != 0) return false;
172         if (!compare_c_field_value(f1->
173             value, f2->value)) return
174             false;
175         f1 = f1->next;
176         f2 = f2->next;
177     }
178     return f1 == NULL && f2 == NULL; //
179     Both must be null
180 }
181
182 bool compare_c_field_value(const c_field_
183     _value_t* v1, const c_field_value_t*
184     v2) {
185     if (!v1 || !v2 || v1->type != v2->
186         type) return false;
187     switch(v1->type) {
188         case C_FIELD_VALUE_TYPE_INT_
189             VALUE: return v1->data.int_
190                 val == v2->data.int_val;
191         case C_FIELD_VALUE_TYPE_STRING_
192             VALUE: return strcmp(v1->
193                 data.string_val, v2->data.
194                 string_val) == 0;
195         default: return false;
196     }
197 }
198
199 bool compare_c_object(const c_object_t*
200     o1, const c_object_t* o2) {
201     if (!o1 || !o2) return false;
202     if (strcmp(o1->object_id, o2->object
203         _id) != 0) return false;
204     if (strcmp(o1->table_name, o2->table
205         _name) != 0) return false;
206     return compare_c_fields(o1->fields,
207         o2->fields);
208 }

```

Listing 19: The Validation suite for the Object Store (src/object_store/test/test_object_store.c)

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 too 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

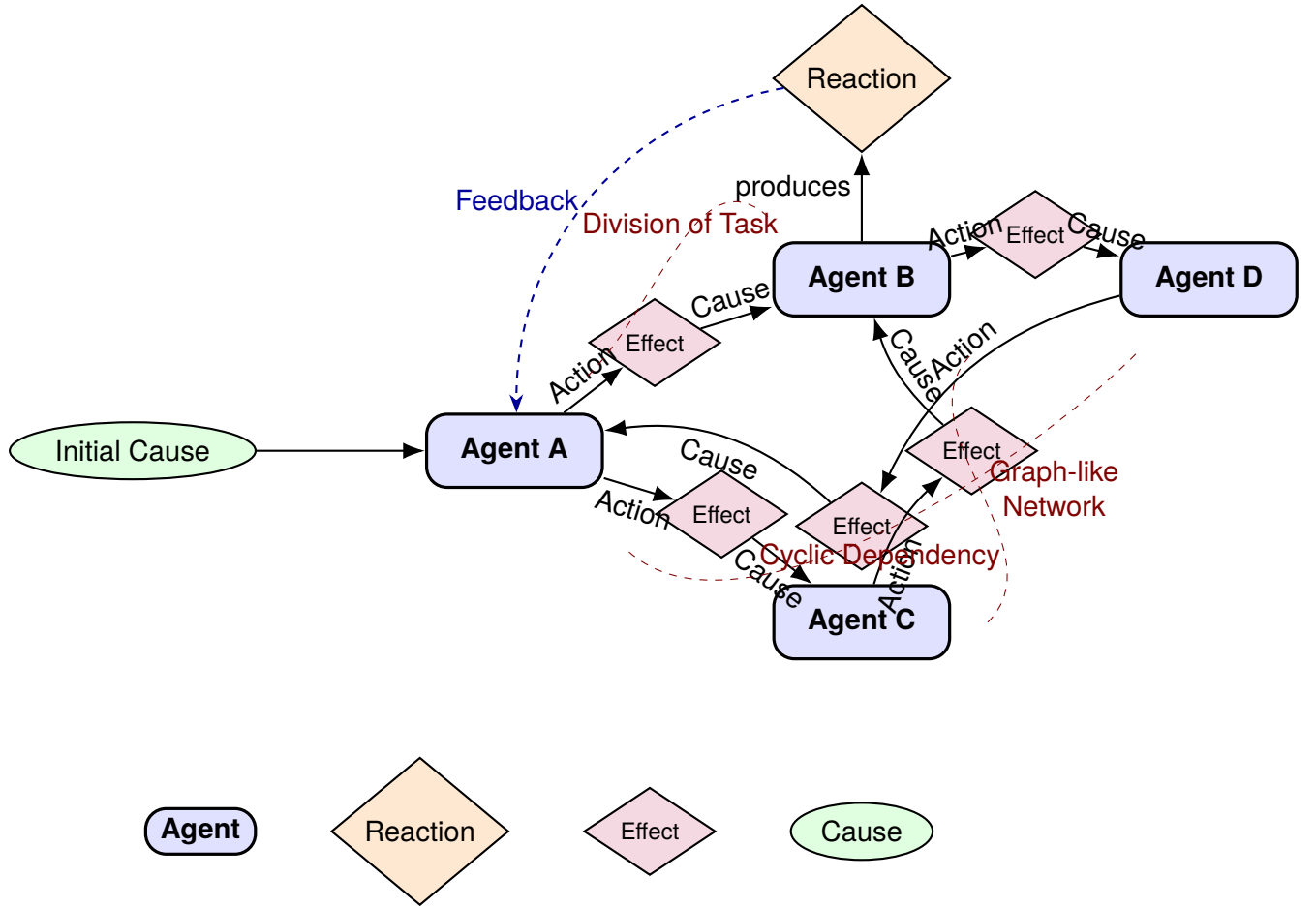


Figure 1: A comprehensive model of the agent interaction graph.

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 AGENK 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 η 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 vector 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
  primitive data types
2 typedef enum {
3     AGENK_DATA_TYPE_FLOAT32, // For
      embeddings, saliency maps
4     AGENK_DATA_TYPE_UINT8,   // For
      standard images, raw bytes
5     AGENK_DATA_TYPE_INT16,   // For raw
      audio (PCM)
6     AGENK_DATA_TYPE_UINT16,  // For
      token grids
7 } AgenkDataType;
8
9 // The core in-memory tensor object
10 typedef struct AgenkTensor {
11     AgenkDataType data_type; // The type
      of each element
12     size_t element_size;     // Size of
      one element in bytes
13
14     void* data;              // Pointer
      to the raw, contiguous data
      buffer
15     size_t data_size_bytes;  // Total
      size of the data buffer
16
17     size_t ndim;             // Number
      of dimensions
18     size_t* shape;          // Array of
      dimension sizes

```

```

19     size_t* strides;         // Array of
      byte-strides for each dimension
20 } 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
  data in the Memory Subsystem
2 typedef struct AgenkTensorView {
3     // Reference to the persistent
      source of data
4     uint64_t source_memory_id;
5     uint32_t layer_hash_id; // Hash
      of the layer name (e.g., "raw_
      sensor_data")
6
7     // The logical structure of this
      specific view
8     AgenkDataType data_type;
9     size_t ndim;
10    size_t* shape;
11    size_t* strides;
12
13    // The offset within the source blob
      where this view begins
14    size_t offset_bytes;
15 } 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
  represents
2 typedef enum {
3     VIEW_OP_SOURCE,          // Base case:
                               // as defined above
4     VIEW_OP_SLICE,           // Structural:
                               // A sub-region of a parent view
5     VIEW_OP_CONCATENATE,     // Structural:
                               // Concatenation of multiple
                               // parents
6     VIEW_OP_ADD,              // Arithmetic:
                               // Element-wise addition of
                               // parents
7     // ... other primitive operations
8 } AgenkViewOperationType;
9
10 // The unified structure for a node in
   the computational graph
11 typedef struct AgenkTensorView {
12     AgenkViewOperationType op_type;
13
14     // Logical properties of the tensor
       this view describes
15     AgenkDataType data_type;
16     size_t ndim;
17     size_t* shape;
18     size_t* strides;
19
20     // Links to parent nodes in the
       computational graph
21     struct AgenkTensorView** parents;
22     size_t num_parents;
23
24     // Operation-specific parameters
25     void* op_params;
26 } 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 'typedef'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 typedef enum {
```

```

38     AGENK_DATA_TYPE_UNDEFINED = 0,
39     AGENK_DATA_TYPE_FLOAT32, // For
        embeddings, saliency maps,
        continuous data
40     AGENK_DATA_TYPE_UINT8, // For
        standard 8-bit images, raw bytes
41     AGENK_DATA_TYPE_INT16, // For 16-
        bit raw audio (PCM)
42     AGENK_DATA_TYPE_UINT16, // For
        token grids from the Pattern
        Engine
43 } AgenkDataType;
44
45 /**
46  * @enum AgenkContentType
47  * @brief Defines the LOGICAL
        interpretation of the data within a
        tensor.
48  * This is orthogonal to the physical
        data type.
49  */
50 typedef enum {
51     AGENK_CONTENT_TYPE_UNDEFINED = 0, //
        Content is unknown or generic
52     AGENK_CONTENT_TYPE_RAW_BYTES, //
        Generic byte data
53     AGENK_CONTENT_TYPE_IMAGE_PIXELS, //
        Data represents image pixels
54     AGENK_CONTENT_TYPE_AUDIO_PCM, //
        Data represents raw audio
        samples
55     AGENK_CONTENT_TYPE_UTF8_TEXT, //
        Data is a sequence of UTF-8
        characters
56     AGENK_CONTENT_TYPE_TOKEN_IDS, //
        Data is a sequence of learned
        token IDs
57     AGENK_CONTENT_TYPE_EMBEDDING //
        Data is a semantic embedding
        vector
58 } AgenkContentType;
59
60 /**
61  * @enum AgenkViewOperationType
62  * @brief Defines the operation a
        TensorView node represents in the
        computational graph.
63  */
64 typedef enum {
65     VIEW_OP_SOURCE, // Base case:
        A view of data in persistent
        memory
66     VIEW_OP_SLICE, // Structural:
        A sub-region of a parent view
67     VIEW_OP_CONCATENATE, // Structural:
        Concatenation of multiple
        parents
68     VIEW_OP_TRANSPOSE, // Structural:
        Permutation of axes
69     VIEW_OP_BROADCAST, // Structural:
        Expands a dimension of size 1
70     VIEW_OP_ADD, // Arithmetic:
        Element-wise addition of
        parents
71     VIEW_OP_SUBTRACT, // Arithmetic:
        Element-wise subtraction
72     VIEW_OP_MULTIPLY, // Arithmetic:
        Element-wise multiplication
73     VIEW_OP_SCALE // Arithmetic:
        Multiplication by a scalar
74 } AgenkViewOperationType;
75
76 /**
77  * @brief Opaque forward declarations
        for the core data structures.
78  * The full definitions are hidden in
        private headers to enforce a clean
        API.
79  * Users of this API should only ever
        interact with pointers to these
        structs.
80  */
81 typedef struct AgenkTensor AgenkTensor;
82 typedef struct AgenkTensorView
        AgenkTensorView;
83
84 //=====
85 // SECTION 2: TENSOR
86 // LIFECYCLE MANAGEMENT
87 //=====
88
89 /**
90  * @brief Creates a new, contiguous,
        zero-initialized in-memory tensor.
91  *
92  * This function performs a single
        memory allocation for the tensor
        struct, its
93  * metadata (shape, strides), and the
        data buffer itself to improve data
        locality
94  * and reduce allocation overhead.
95  *
96  * @param shape An array of sizes for
        each dimension. Must not be NULL.
97  * @param ndim The number of dimensions
        (the length of the shape array).
        Must be > 0.
98  * @param type The primitive data type
        of each element in the tensor.
99  * @return A pointer to the new
        AgenkTensor, or NULL on allocation
        failure or invalid input.
100  * The caller is responsible for
        freeing the tensor with tensor_
        free().
101  */
102 AgenkTensor* tensor_create(const size_t*
        shape, size_t ndim, AgenkDataType
        type, AgenkContentType content);
103
104 /**
105  * @brief Frees all memory associated
        with an AgenkTensor created by
        tensor_create().
106  * @param tensor A pointer to the tensor
        to be freed. Can safely be called
        on NULL.
107  */
108 void tensor_free(AgenkTensor* tensor);
109
110 //=====
111 // SECTION 3: TENSOR
112 // VIEW MANAGEMENT
113 //=====
114
115 /**
116  * @brief Creates a base-case "source"
        view, a logical pointer to data in

```



```

118     persistent memory.
119     * This is the root node for many
120       computational graphs.
121     * @param memory_id The unique ID of the
122       MemoryObject in the database.
123     * @param layer_name A string identifier
124       for the layer within the
125       MemoryObject (e.g., "raw_sensor_
126       data").
127     * @return A pointer to the new
128       AgenkTensorView, or NULL on failure
129     *
130       The caller is responsible for
131       freeing the view with view_free().
132     */
133     AgenkTensorView* view_create_source(
134         uint64_t memory_id, const char*
135         layer_name);
136
137 /**
138  * @brief Frees a TensorView object and
139  * any resources it owns.
140  * Note: This does NOT recursively free
141  * parent views. Each view must be
142  * managed independently.
143  * @param view A pointer to the view to
144  * be freed. Can safely be called on
145  * NULL.
146  */
147 void view_free(AgenkTensorView* view);
148
149 // --- Primitive Operations for building
150 // the computational graph ---
151
152 /**
153  * @brief Creates a new view
154  * representing an element-wise
155  * addition of two parent views.
156  * The shapes of the parent views must
157  * be compatible for broadcasting.
158  * @param view_a The first parent view.
159  * @param view_b The second parent view.
160  * @return A new AgenkTensorView
161  * representing the lazy addition, or
162  * NULL on failure.
163  */
164     AgenkTensorView* view_create_add(const
165     AgenkTensorView* view_a, const
166     AgenkTensorView* view_b);
167
168 /**
169  * @brief Creates a new view that
170  * represents a slice of a parent view
171  *
172  * This is a lazy, metadata-only
173  * operation.
174  * @param parent The parent view to
175  * slice.
176  * @param offsets An array of starting
177  * indices for the slice in each
178  * dimension.
179  * @param shape An array of sizes for
180  * the slice in each dimension.
181  * @return A new AgenkTensorView, or
182  * NULL on failure.
183  */
184     AgenkTensorView* view_create_slice(const
185     AgenkTensorView* parent, const size
186     _t* offsets, const size_t* shape);
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

196 //=====
197
198 /**
199  * @brief Retrieves the content type of
200  * the specified tensor.
201  *
202  * This function returns the content
203  * type associated with the given
204  * tensor,
205  * which indicates the type of data
206  * stored within the tensor (e.g.,
207  * float, int).
208  *
209  * @param tensor Pointer to the
210  * AgenkTensor whose content type is
211  * to be retrieved.
212  * @return The content type of the
213  * tensor as an AgenkContentType value
214  *
215  */
216 AgenkContentType tensor_get_content_type
217 (const AgenkTensor* tensor);
218
219 /**
220  * @brief Gets the number of dimensions
221  * of a materialized tensor.
222  * @param tensor A non-NULL,
223  * materialized tensor.
224  * @return The number of dimensions.
225  */
226 size_t tensor_get_ndim(const AgenkTensor
227 * tensor);
228
229 /**
230  * @brief Gets a pointer to the shape
231  * array of a materialized tensor.
232  * @param tensor A non-NULL,
233  * materialized tensor.
234  * @return A read-only pointer to the
235  * shape array. The lifetime of this
236  * pointer
237  * is tied to the lifetime of
238  * the tensor. Do not free it.
239  */
240 const size_t* tensor_get_shape(const
241 AgenkTensor* tensor);
242
243 /**
244  * @brief Retrieves the data type of the
245  * specified tensor.
246  *
247  * This function returns the data type
248  * associated with the given tensor.
249  *
250  * @param tensor Pointer to the
251  * AgenkTensor whose data type is to
252  * be retrieved.
253  * @return The data type of the tensor
254  * as an AgenkDataType enum value.
255  */
256 AgenkDataType tensor_get_data_type(const
257 AgenkTensor* tensor);
258
259 /**
260  * @brief Gets a pointer to a specific
261  * element within a materialized
262  * tensor.
263  *
264  * This is typically implemented as a
265  * static inline function for maximum
266
267 performance,
268 * allowing the compiler to eliminate
269 function call overhead.
270 *
271 * @param tensor A non-NULL,
272 materialized tensor.
273 * @param coords An array of coordinates
274 , one for each dimension.
275 * @return A void pointer to the
276 requested element. The caller must
277 cast this to the
278 correct type (e.g., `float*`,
279 `uint8_t*`). Returns NULL if
280 inputs
281 are invalid or coordinates
282 are out of bounds.
283 */
284 void* tensor_get_element_ptr(const
285 AgenkTensor* tensor, const size_t*
286 coords);
287
288 /**
289  * @brief Gets a read-only pointer to
290  * the raw data buffer of a
291  * materialized tensor.
292  * @param tensor A non-NULL,
293  * materialized tensor.
294  * @return A const void pointer to the
295  * start of the data buffer.
296 */
297 const void* tensor_get_data_ptr(const
298 AgenkTensor* tensor);
299
300 /**
301  * @brief Gets the total size in bytes
302  * of the tensor's data buffer.
303  * @param tensor A non-NULL,
304  * materialized tensor.
305  * @return The size of the data buffer
306  * in bytes.
307 */
308 size_t tensor_get_data_size_bytes(const
309 AgenkTensor* tensor);
310
311 bool tensor_op_copy(AgenkTensor* dest,
312 const AgenkTensor* src);
313 bool tensor_op_copy_from_view(
314 AgenkTensor* dest, const AgenkTensor
315 * src_view);
316 bool tensor_op_concatenate(AgenkTensor*
317 dest, const AgenkTensor** srcs, size
318 _t num_srcs, size_t axis);
319
320 #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  *        Aucta Sapience
21  */
22
23 #ifndef TENSOR_PRIVATE_H
24 #define TENSOR_PRIVATE_H
25
26 #include "../include/agenk_tensor.h" //
27     Include the public API
28
29 #include <assert.h> //
30     For debug assertions
31 #include <stdio.h> //
32     For logging/error messages
33
34 //=====
35 // SECTION 1: FULL STRUCT DEFINITIONS
36 //=====
37
38 #define AGENK_TENSOR_MAGIC 0xDEADBEEF
39 #define AGENK_VIEW_MAGIC 0xCAFEBAFE
40
41 struct AgenkTensor {
42     AgenkDataType data_type;
43     AgenkContentType content_type;
44     size_t element_size;
45     void* data;
46     size_t data_size_bytes;
47     size_t ndim;
48     size_t* shape;
49     size_t* strides;
50     uint32_t magic;
51 };
52
53 struct AgenkTensorView {
54     AgenkViewOperationType op_type;
55     AgenkDataType data_type;
56     AgenkContentType content_type;
57     size_t ndim;
58     size_t* shape;
59     size_t* strides;
60     struct AgenkTensorView** parents;
61     size_t num_parents;
62     void* op_params;
63     uint32_t magic;
64     uint64_t source_memory_id;
65     uint32_t layer_hash_id;
66     size_t offset_bytes;
67 };
68
69 //=====
70 // SECTION 2: INTERNAL HELPER
71 // FUNCTION PROTOTYPES
72 //=====
73
74 size_t internal_get_element_size(
75     AgenkDataType type);
76 bool internal_calculate_num_elements(
77     const size_t* shape, size_t ndim,
78     size_t* out_num_elements);
79 uint32_t internal_hash_string(const char
80     * str);
81
82 //=====
83 // SECTION 3: INTERNAL COMPUTATIONAL
84 // KERNEL PROTOTYPES
85 //=====
86
87 // --- Arithmetic Kernels ---
88 bool tensor_op_add(AgenkTensor* dest,
89     const AgenkTensor* src_a, const
90     AgenkTensor* src_b);
91 bool tensor_op_subtract(AgenkTensor*
92     dest, const AgenkTensor* src_a,
93     const AgenkTensor* src_b);
94 bool tensor_op_multiply(AgenkTensor*
95     dest, const AgenkTensor* src_a,
96     const AgenkTensor* src_b);
97 bool tensor_op_scale(AgenkTensor* dest,
98     const AgenkTensor* src, float scalar

```

```

    );
75
76 // --- Structural Kernels ---
77 bool tensor_op_copy_from_view(
    AgenkTensor* dest, const AgenkTensor
    * src_view);
78 bool tensor_op_concatenate(AgenkTensor*
    dest, const AgenkTensor** srcs, size
    _t num_srcs, size_t axis);
79
80 // --- Non-Owning Slice Helpers ---
81 AgenkTensor* tensor_slice(const
    AgenkTensor* parent, const size_t*
    offsets, const size_t* shape);
82 void tensor_slice_free(AgenkTensor*
    slice);
83
84 //=====
85 // SECTION 4: DEBUGGING AND
86 // ERROR HANDLING MACROS
87 //=====
88
89 #ifndef NDEBUG
90 #define TENSOR_LOG_ERROR(format,
    ...) \
91     fprintf(stderr, "[TENSOR_ERROR]
    %s:%d: " format "\n", __FILE
    __, __LINE__, ##__VA_ARGS__)
92
93 #define TENSOR_ASSERT(condition,
    message) \
94     do { \
95         if (!(condition)) { \
96             TENSOR_LOG_ERROR("
    Assertion failed: %s
    ", message); \
97             assert(condition); \
98         } \
99     } while (0)
100
101 #define TENSOR_VALIDATE(tensor) \
102     do { \
103         TENSOR_ASSERT(tensor != NULL
    , "Tensor pointer is
    NULL."); \
104         TENSOR_ASSERT(((AgenkTensor
    *)tensor)->magic ==
    AGENK_TENSOR_MAGIC, "
    Invalid tensor pointer (
    bad magic number or use-
    after-free)."); \
105     } while (0)
106
107 #define VIEW_VALIDATE(view) \
108     do { \
109         TENSOR_ASSERT(view != NULL,
    "View pointer is NULL."); \
110         TENSOR_ASSERT(((
    AgenkTensorView*)view)->
    magic == AGENK_VIEW_
    MAGIC, "Invalid view
    pointer (bad magic
    number or use-after-free
    )."); \
111     } while (0)
112
113 #else // Release build
114 #define TENSOR_LOG_ERROR(format,

```

```

    ...) ((void)0)
116 #define TENSOR_ASSERT(condition,
    message) ((void)0)
117 #define TENSOR_VALIDATE(tensor) ((
    void)0)
118 #define VIEW_VALIDATE(view) ((void)
    0)
119 #endif // NDEBUG
120
121 #endif // TENSOR_PRIVATE_H
122

```

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:
 1. **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 pre-fetch the beginning of its raw data.
 2. **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
    destruction of materialized
    AgenkTensors.
4  *
5  * This file contains the core memory
    management logic for the Tensor
    Substrate.

```

```

6  * The primary design goal is
   * performance and data locality,
   * achieved through
7  * a single-block allocation strategy.
   * All public functions are defined in
   * agenk_tensor.h.
8  *
9  *
10 * @version 1.0.0
11 * @author Ankush Yadav, Ankit Yadav,
   * Aucta Sapience
12 */
13
14 #include "private/tensor_private.h" //
   Includes public header, asserts,
   helpers
15 #include <stdlib.h> //
   For malloc, calloc, free
16 #include <string.h> //
   For memcpy
17
18 //=====
19 // SECTION 1: INTERNAL HELPER
20 // FUNCTION IMPLEMENTATIONS
21 //=====
22
23 // This function is declared in tensor_
   private.h and used by multiple files
24 // in this module.
25 size_t internal_get_element_size(
   AgenkDataType type) {
26     switch (type) {
27         case AGENK_DATA_TYPE_FLOAT32:
28             return sizeof(float);
29         case AGENK_DATA_TYPE_UINT8:
30             return sizeof(uint8_t);
31         case AGENK_DATA_TYPE_INT16:
32             return sizeof(int16_t);
33         case AGENK_DATA_TYPE_UINT16:
34             return sizeof(uint16_t);
35         default: return 0; // Invalid
   type
36     }
37 }
38
39 // This function is also declared in
   tensor_private.h
40 bool internal_calculate_num_elements(
   const size_t* shape, size_t ndim,
   size_t* out_num_elements) {
41     size_t num_elements = 1;
42     for (size_t i = 0; i < ndim; ++i) {
43         // Check for potential overflow
   before multiplication.
44         // If shape[i] is 0, the total
   will be 0, which is fine.
45         if (shape[i] > 0 && num_elements
   > (SIZE_MAX / shape[i])) {
46             *out_num_elements = 0;
47             return false; // Overflow
   detected
48         }
49         num_elements *= shape[i];
50     }
51     *out_num_elements = num_elements;
52     return true;
53 }
54
55 //=====
56 // SECTION 2: PUBLIC API
57 // FUNCTION IMPLEMENTATIONS

```

```

54 //=====
55
56 AgenkTensor* tensor_create(const size_t*
   shape, size_t ndim, AgenkDataType
   type, AgenkContentType content) {
57     // --- 1. Input Validation ---
58     if (!shape || ndim == 0) {
59         TENSOR_LOG_ERROR("Invalid_
   arguments:_shape_is_NULL_or_
   ndim_is_0.");
60         return NULL;
61     }
62
63     size_t element_size = internal_get_
   element_size(type);
64     if (element_size == 0) {
65         TENSOR_LOG_ERROR("Invalid_data_
   type_specified:_%d", type);
66         return NULL;
67     }
68
69     // --- 2. Calculate Memory Layout
   ---
70     size_t num_elements;
71     if (!internal_calculate_num_elements
   (shape, ndim, &num_elements)) {
72         TENSOR_LOG_ERROR("Shape_
   calculation_would_overflow_
   size_t.");
73         return NULL;
74     }
75
76     // Calculate the size of each part
   of the single memory block
77     const size_t tensor_struct_bytes =
   sizeof(AgenkTensor);
78     const size_t shape_bytes = ndim *
   sizeof(size_t);
79     const size_t strides_bytes = ndim *
   sizeof(size_t);
80     const size_t data_bytes = num_
   elements * element_size;
81
82     // Check for overflow in total size
   calculation
83     size_t total_bytes = tensor_struct_
   bytes + shape_bytes + strides_
   bytes + data_bytes;
84     if (total_bytes < tensor_struct_
   bytes) { // A simple overflow
   check
85         TENSOR_LOG_ERROR("Total_
   allocation_size_would_
   overflow_size_t.");
86         return NULL;
87     }
88
89     // --- 3. Single Block Allocation
   ---
90     // Use calloc to ensure all memory,
   including the data buffer, is
   zero-initialized.
91     char* block = (char*)calloc(1, total
   bytes);
92     if (!block) {
93         TENSOR_LOG_ERROR("Failed_to_
   allocate_%zu_bytes_for_
   tensor.", total_bytes);
94         return NULL;
95     }

```



```

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

```

```

138     freeing.
139 // This helps catch double-free
140 // bugs and other memory
141 // corruption issues.
142 TENSOR_VALIDATE(tensor);
143 // Because we used a single-
144 // block allocation, we only
145 // need one call to free().
146 free(tensor);
147 }
148 }

```

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,
    AuctaSapience
15  */
16
17  #include "private/tensor_private.h"
18  #include <stdlib.h>
19  #include <string.h>
20
21  //=====
22  // SECTION 1: INTERNAL
23  // HELPER FUNCTIONS
24  //=====
25
26  // Stub function for database
    interaction.
27  // This will be replaced by the real
    Memory subsystem.
28  static bool get_metadata_from_memory(
    uint64_t memory_id, const char*
    layer_name, AgenkDataType* out_type,
    AgenkContentType* out_content, size
    _t* out_ndim, size_t** out_shape) {
29  // --- STUB IMPLEMENTATION for
    testing ---
30  if (memory_id == 123 && strcmp(layer
    _name, "raw_sensor_data") == 0)
    {
31      *out_type = AGENK_DATA_TYPE_
        UINT8;
32      *out_content = AGENK_CONTENT_
        TYPE_IMAGE_PIXELS; // Assign
        a logical type
33      *out_ndim = 3;
34      *out_shape = (size_t*)malloc(3 *
        sizeof(size_t));
35      if (!*out_shape) return false;
36      (*out_shape)[0] = 100; (*out_
        shape)[1] = 200; (*out_shape
        )[2] = 3;
37      return true;
38  }
39  // Test cases for arithmetic tests
40  if (memory_id == 1 || memory_id ==
    2) {
41      *out_type = AGENK_DATA_TYPE_
        FLOAT32;
42      *out_content = AGENK_CONTENT_
        TYPE_EMBEDDING; // Assign a
        logical type
43      *out_ndim = 2;
44      *out_shape = (size_t*)malloc(2 *
        sizeof(size_t));
45      if (!*out_shape) return false;
46      (*out_shape)[0] = 10; (*out_
        shape)[1] = 20;
47      return true;
48  }
49  TENSOR_LOG_ERROR("MEMORY_STUB:
    Unknown_memory_ID_%llu_or_layer_
    '%s'", (unsigned long long)
    memory_id, layer_name);
50  return false;
51 }
52
53 // Simple string hashing function (djb2)
54 uint32_t internal_hash_string(const char
    * str) {
55     unsigned long hash = 5381;
56     int c;

```

```

57     if (!str) return 0;
58     while ((c = *str++)) {
59         hash = ((hash << 5) + hash) + c;
60     }
61     return (uint32_t)hash;
62 }
63
64 /**
65  * @brief A generic helper to allocate a
    new view and its metadata arrays.
66  * Uses a single-block allocation for
    efficiency.
67  */
68 static AgenkTensorView* allocate_view(
    size_t ndim) {
69     size_t shape_bytes = ndim * sizeof(
        size_t);
70     size_t strides_bytes = ndim * sizeof
        (size_t);
71     size_t total_bytes = sizeof(
        AgenkTensorView) + shape_bytes +
        strides_bytes;
72
73     char* block = (char*)calloc(1, total
        _bytes);
74     if (!block) {
75         TENSOR_LOG_ERROR("Failed_to_
        allocate_%zu_bytes_for_view.
        ", total_bytes);
76         return NULL;
77     }
78
79     AgenkTensorView* view = (
        AgenkTensorView*)block;
80     view->shape = (size_t*)(block +
        sizeof(AgenkTensorView));
81     view->strides = (size_t*)(block +
        sizeof(AgenkTensorView) + shape_
        bytes);
82     view->magic = AGENK_VIEW_MAGIC;
83     view->ndim = ndim;
84
85     return view;
86 }
87
88 /**
89  * @brief The core broadcasting logic.
90  * Determines if two shapes are
    broadcast-compatible and calculates
    the resulting shape.
91  */
92 static bool get_broadcast_shape(const
    size_t* shape_a, size_t ndim_a,
    const size_t* shape_b, size_t ndim_b
    , size_t** out_shape, size_t* out_
    ndim) {
93     *out_ndim = (ndim_a > ndim_b) ? ndim
        _a : ndim_b;
94     *out_shape = (size_t*)malloc(*out_
        ndim * sizeof(size_t));
95     if (!*out_shape) return false;
96
97     for (size_t i = 0; i < *out_ndim; ++
        i) {
98         size_t idx_a = ndim_a > i ? ndim
            _a - 1 - i : (size_t)-1;
99         size_t idx_b = ndim_b > i ? ndim
            _b - 1 - i : (size_t)-1;
100
101         size_t dim_a = (idx_a != (size_t)

```

```

102         )-1) ? shape_a[idx_a] : 1;
103         size_t dim_b = (idx_b != (size_t)-1) ? shape_b[idx_b] : 1;
104
105         if (dim_a != dim_b && dim_a != 1
106             && dim_b != 1) {
107             TENSOR_LOG_ERROR("Shapes are
108                 not broadcast-
109                 compatible.");
110             free(*out_shape);
111             *out_shape = NULL;
112             return false;
113         }
114         (*out_shape)[*out_ndim - 1 - i]
115             = (dim_a > dim_b) ? dim_a :
116               dim_b;
117     }
118     return true;
119 }
120
121 //=====
122 // SECTION 2: PUBLIC API
123 // - VIEW LIFECYCLE
124 //=====
125
126 AgenkTensorView* view_create_source(
127     uint64_t memory_id, const char*
128     layer_name) {
129     if (!layer_name) return NULL;
130
131     AgenkDataType data_type;
132     AgenkContentType content_type; //
133     <<-- Get the content type
134     size_t ndim;
135     size_t* shape = NULL;
136     if (!get_metadata_from_memory(memory_id, layer_name, &data_type, &content_type, &ndim, &shape)) {
137         return NULL;
138     }
139
140     AgenkTensorView* 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             shape[i + 1];
169         }
170     }
171     return view;
172 }
173
174 void view_free(AgenkTensorView* view) {
175     if (view) {
176         VIEW_VALIDATE(view);
177         if (view->op_params) free(view->
178             op_params);
179         if (view->parents) free(view->
180             parents);
181         free(view);
182     }
183 }
184
185 //=====
186 // SECTION 3: PUBLIC API -
187 // COMPUTATIONAL GRAPH PRIMITIVES
188 //=====
189
190 AgenkTensorView* view_create_slice(const
191     AgenkTensorView* parent, const size
192     _t* offsets, const size_t* shape) {
193     if (!parent || !offsets || !shape)
194         return NULL;
195     VIEW_VALIDATE(parent);
196
197     #ifndef NDEBUG
198     for (size_t i = 0; i < parent->ndim;
199         ++i) {
200         if ((offsets[i] + shape[i]) >
201             parent->shape[i]) {
202             TENSOR_LOG_ERROR("Slice is
203                 out of bounds for
204                 dimension %zu.", i);
205             return NULL;
206         }
207     }
208     #endif
209
210     AgenkTensorView* view = allocate_
211     view(parent->ndim);
212     if (!view) return NULL;
213
214     view->op_type = VIEW_OP_SLICE;
215     view->data_type = parent->data_type;
216     view->content_type = parent->content
217     _type; // Inherit content type
218     memcpy(view->shape, shape, parent->
219     ndim * sizeof(size_t));
220     memcpy(view->strides, parent->
221     strides, parent->ndim * sizeof(
222     size_t));
223
224     view->op_params = malloc(parent->
225     ndim * sizeof(size_t));
226     if (!view->op_params) { free(view);
227         return NULL; }
228     memcpy(view->op_params, offsets,
229     parent->ndim * sizeof(size_t));
230
231     view->parents = (AgenkTensorView**)
232     malloc(sizeof(AgenkTensorView*))
233     ;
234     if (!view->parents) { free(view->op_
235     params); free(view); return NULL
236     ; }

```

200	view->parents[0] = (AgenkTensorView		malloc(2 * sizeof(
)parent;		AgenkTensorView));
201	view->num_parents = 1;	245	if (!view->parents) { free(view);
202			return NULL; }
203	return view;	246	view->parents[0] = (AgenkTensorView
204	}		*)view_a;
205		247	view->parents[1] = (AgenkTensorView
206	static AgenkTensorView* view_create_		*)view_b;
	binary_op(const AgenkTensorView*	248	view->num_parents = 2;
	view_a, const AgenkTensorView* view_	249	
	b, AgenkViewOperationType op_type) {	250	return view;
207	if (!view_a !view_b) return NULL;	251	}
208	VIEW_VALIDATE(view_a); VIEW_VALIDATE	252	AgenkTensorView* view_create_add(const
	(view_b);	253	AgenkTensorView* view_a, const
209			AgenkTensorView* view_b) {
210	if (view_a->data_type != view_b->	254	return view_create_binary_op(view_a,
	data_type) {		view_b, VIEW_OP_ADD);
211	TENSOR_LOG_ERROR("Operands for	255	}
	binary_op must have same	256	
	data_type.");	257	AgenkTensorView* view_create_subtract(
212	return NULL;		const AgenkTensorView* view_a, const
213	}		AgenkTensorView* view_b) {
214		258	return view_create_binary_op(view_a,
215	size_t* result_shape = NULL;		view_b, VIEW_OP_SUBTRACT);
216	size_t result_ndim = 0;	259	}
217	if (!get_broadcast_shape(view_a->	260	AgenkTensorView* view_create_multiply(
	shape, view_a->ndim, view_b->	261	const AgenkTensorView* view_a, const
	shape, view_b->ndim,	262	AgenkTensorView* view_b) {
218	&result_	263	return view_create_binary_op(view_a,
	shape,		view_b, VIEW_OP_MULTIPLY);
	&result	264	}
	_ndim))	265	AgenkTensorView* view_create_scale(const
	{		AgenkTensorView* parent, float
219	return NULL;	266	scalar) {
220	}	267	if (!parent) return NULL;
221		268	VIEW_VALIDATE(parent);
222	AgenkTensorView* view = allocate_	269	
	view(result_ndim);	270	AgenkTensorView* view = allocate_
223	if (!view) {		view(parent->ndim);
224	free(result_shape);	271	if (!view) return NULL;
225	return NULL;	272	view->op_type = VIEW_OP_SCALE;
226	}	273	view->data_type = parent->data_type;
227		274	view->content_type = parent->content
228	view->op_type = op_type;		_type; // Scaling preserves
229	view->data_type = view_a->data_type;	275	content type
230	// THE FIX: For arithmetic, the		memcpy(view->shape, parent->shape,
	result is usually just raw	276	parent->ndim * sizeof(size_t));
	numerical data,		memcpy(view->strides, parent->
231	// unless a more sophisticated rule	277	strides, parent->ndim * sizeof(
	is defined.	278	size_t));
232	view->content_type = AGENK_CONTENT_	279	
	TYPE_RAW_BYTES;	280	view->op_params = malloc(sizeof(
233	memcpy(view->shape, result_shape,		float));
	result_ndim * sizeof(size_t));	281	if (!view->op_params) { free(view);
234	free(result_shape);	282	return NULL; }
235		283	*(float*)view->op_params = scalar;
236	size_t element_size = internal_get_		
	element_size(view->data_type);	284	view->parents = (AgenkTensorView**)
237	if (result_ndim > 0) {		malloc(sizeof(AgenkTensorView*))
238	view->strides[result_ndim - 1] =		;
	element_size;	285	if (!view->parents) { free(view->op
239	for (int i = (int)result_ndim -		params); free(view); return NULL
	2; i >= 0; --i) {		;
240	view->strides[i] = view->		view->parents[0] = (AgenkTensorView
	strides[i + 1] * view->		*)parent;
	shape[i + 1];		view->num_parents = 1;
241	}		
242	}		
243			
244	view->parents = (AgenkTensorView**)		

```

286
287     return view;
288 }

```

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 #include "private/tensor_private.h"
25 #include <stdlib.h>
26 #include <string.h>
27
28 //=====
29 // SECTION 1: GENERIC N-DIMENSIONAL
30 // TENSOR ITERATOR (INTERNAL HELPER)
31 //=====
32 // The iterator is a powerful internal
33 // tool that encapsulates the complex
34 // logic
35 // of N-dimensional, broadcast-aware
36 // iteration. Its implementation is
37 // kept
38 // static within this file, as it is an
39 // implementation detail of the kernels
40 *
41 /**
42 * @struct TensorIterator
43 * @brief An iterator for traversing one
44 * or more tensors simultaneously,
45 * with full
46 * broadcasting support. This is
47 * the core engine for all element-
48 * wise ops.
49 */
50 typedef struct {
51     size_t ndim; // The
52                 // number of dimensions of the
53                 // broadcasted result
54     const size_t* shape; // The
55                         // shape of the broadcasted result
56                         // (owned by dest tensor)
57
58     void** ptrs; // Array of
59                 // current pointers for each
60                 // tensor being iterated
61     size_t num_tensors; // Number
62                         // of tensors
63
64     // Internal state for iteration
65     // logic
66     size_t* coords; // Current
67                     // logical coordinates in the
68                     // broadcasted shape
69     size_t** b_strides; // Pre-
70                         // calculated broadcast strides for
71                         // each tensor
72     const AgenkTensor** tensors;
73 } TensorIterator;

```



```

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 is out of
190                 bounds for dimension
191                 with size %zu.",
192                 i, coords[i],
193                 tensor->
194                 shape[i])
195             ;
196             return NULL;
197         }
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 }

```

```

189     return (void*)ptr;
190 }
191
192 // --- Computational Kernels ---
193
194 /**
195  * @brief Core engine for element-wise
196  * binary operations, using the
197  * generic iterator.
198  */
199 static bool tensor_op_binary_broadcast(
200     AgenkTensor* dest, const AgenkTensor
201     * src_a, const AgenkTensor* src_b,
202     char op) {
203     TENSOR_VALIDATE(dest); TENSOR_
204     VALIDATE(src_a); TENSOR_VALIDATE
205     (src_b);
206     if (dest->data_type != src_a->data_
207         type || src_a->data_type != src_
208         b->data_type) return false;
209
210     size_t* coords = (size_t*)calloc(
211         dest->ndim, sizeof(size_t));
212     if (!coords) return false;
213
214     do {
215         // Map destination coordinates
216         // to source coordinates for
217         // broadcasting
218         size_t coords_a_stack[32],
219             coords_b_stack[32];
220         size_t* coords_a = coords_a_
221             stack;
222         size_t* coords_b = coords_b_
223             stack;
224
225         for(size_t i = 0; i < src_a->
226             ndim; ++i) {
227             size_t dest_idx = dest->ndim
228                 - src_a->ndim + i;
229             coords_a[i] = (src_a->shape[
230                 i] == 1) ? 0 : coords[
231                 dest_idx];
232         }
233         for(size_t i = 0; i < src_b->
234             ndim; ++i) {
235             size_t dest_idx = dest->ndim
236                 - src_b->ndim + i;
237             coords_b[i] = (src_b->shape[
238                 i] == 1) ? 0 : coords[
239                 dest_idx];
240         }
241
242         void* p_dest = tensor_get_
243             element_ptr(dest, coords);
244         const void* p_src_a = tensor_get
245             _element_ptr(src_a, coords_a
246             );
247         const void* p_src_b = tensor_get
248             _element_ptr(src_b, coords_b
249             );
250         if(!p_dest || !p_src_a || !p_src
251             _b) continue;
252
253         // Perform the type-specific
254         // operation
255         switch(dest->data_type) {
256             case AGENK_DATA_TYPE_FLOAT32
257                 : {
258                 float v_a = *(const

```

```

228         float*)p_src_a;
229         float v_b = *(const
230         float*)p_src_b;
231         if (op == '+') *(float*)
232         p_dest = v_a + v_b;
233         else if (op == '-')
234         *(float*)p_dest = v_
235         a - v_b; else if (op
236         == '*') *(float*)p_
237         dest = v_a * v_b;
238         break;
239     }
240     case AGENK_DATA_TYPE_UINT8:
241     {
242         uint8_t v_a = *(const
243         uint8_t*)p_src_a;
244         uint8_t v_b = *(
245         const uint8_t*)p_src
246         b;
247         if (op == '+') *(uint8_t
248         *)p_dest = v_a + v_b
249         ; else if (op == '-')
250         *(uint8_t*)p_dest
251         = v_a - v_b; else if
252         (op == '*') *(uint8
253         _t*)p_dest = v_a * v
254         _b;
255         break;
256     }
257     case AGENK_DATA_TYPE_INT16:
258     {
259         int16_t v_a = *(const
260         int16_t*)p_src_a;
261         int16_t v_b = *(
262         const int16_t*)p_src
263         b;
264         if (op == '+') *(int16_t
265         *)p_dest = v_a + v_b
266         ; else if (op == '-')
267         *(int16_t*)p_dest
268         = v_a - v_b; else if
269         (op == '*') *(int16
270         _t*)p_dest = v_a * v
271         _b;
272         break;
273     }
274     case AGENK_DATA_TYPE_UINT16:
275     {
276         uint16_t v_a = *(const
277         uint16_t*)p_src_a;
278         uint16_t v_b = *(
279         const uint16_t*)p_
280         src_b;
281         if (op == '+') *(uint16_
282         t*)p_dest = v_a + v_
283         b; else if (op == '-')
284         *(uint16_t*)p_
285         dest = v_a - v_b;
286         else if (op == '*')
287         *(uint16_t*)p_dest =
288         v_a * v_b;
289         break;
290     }
291     default: break;
292 }
293
294 int current_dim = (int)dest->
295 ndim - 1;
296 while(current_dim >= 0) {
297     coords[current_dim]++;
298
299     if (coords[current_dim] <
300         dest->shape[current_dim]
301         ) break;
302     coords[current_dim] = 0;
303     current_dim--;
304 }
305 if (current_dim < 0) break;
306 } while(true);
307
308 free(coords);
309 return true;
310 }
311
312 bool tensor_op_add(AgenkTensor* dest,
313 const AgenkTensor* src_a, const
314 AgenkTensor* src_b) {
315     return tensor_op_binary_broadcast(
316         dest, src_a, src_b, '+');
317 }
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
326 bool tensor_op_multiply(AgenkTensor*
327 dest, const AgenkTensor* src_a,
328 const AgenkTensor* src_b) {
329     return tensor_op_binary_broadcast(
330         dest, src_a, src_b, '*');
331 }
332
333 bool tensor_op_scale(AgenkTensor* dest,
334 const AgenkTensor* src, float scalar
335 ) {
336     TENSOR_VALIDATE(dest); TENSOR_
337     VALIDATE(src);
338     AgenkDataType type = src->data_type;
339     size_t shape[] = {1};
340
341     // THE FIX: Provide the fourth
342     // argument to tensor_create.
343     // We use UNDEFINED as this
344     // temporary tensor has no specific
345     // logical content.
346     AgenkTensor* scalar_tensor = tensor_
347     create(shape, 1, type, AGENK_
348     CONTENT_TYPE_UNDEFINED);
349     if(!scalar_tensor) return false;
350
351     switch(type) {
352         case AGENK_DATA_TYPE_FLOAT32: *(
353         float*)scalar_tensor->data =
354         scalar; break;
355         case AGENK_DATA_TYPE_UINT8: *(
356         uint8_t*)scalar_tensor->data
357         = (uint8_t)scalar; break;
358         case AGENK_DATA_TYPE_INT16: *(
359         int16_t*)scalar_tensor->data
360         = (int16_t)scalar; break;
361         case AGENK_DATA_TYPE_UINT16: *(
362         uint16_t*)scalar_tensor->
363         data = (uint16_t)scalar;
364         break;
365         default: tensor_free(scalar_
366         tensor); return false;
367     }
368
369     // Reuse our powerful broadcast-
370     // aware multiplication function

```

```

292     bool success = tensor_op_multiply(
293         dest, src, scalar_tensor);
294     tensor_free(scalar_tensor);
295     return success;
296 }
297 bool tensor_op_copy_from_view(
298     AgenkTensor* dest, const AgenkTensor
299     * src_view) {
300     TENSOR_VALIDATE(dest); TENSOR_
301     VALIDATE(src_view);
302     if (dest->ndim != src_view->ndim ||
303         dest->element_size != src_view->
304         element_size) return false;
305     for(size_t i=0; i<dest->ndim; ++i)
306         if(dest->shape[i] != src_view->
307             shape[i]) return false;
308
309     bool is_src_contiguous = true;
310     size_t expected_stride = src_view->
311     element_size;
312     for (int i = (int)src_view->ndim -
313         1; i >= 0; --i) {
314         if (src_view->shape[i] == 0)
315             continue;
316         if (src_view->strides[i] !=
317             expected_stride) { is_src_
318             contiguous = false; break; }
319         if(src_view->shape[i] > 1)
320             expected_stride *= src_view
321             ->shape[i];
322     }
323
324     if (is_src_contiguous) {
325         memcpy(dest->data, src_view->
326             data, dest->data_size_bytes)
327         ;
328         return true;
329     }
330
331     size_t* coords = (size_t*)calloc(
332         dest->ndim, sizeof(size_t));
333     if(!coords) return false;
334     char* dest_ptr = (char*)dest->data;
335     do {
336         const void* p_src = tensor_get_
337         element_ptr(src_view, coords
338         );
339         memcpy(dest_ptr, p_src, dest->
340         element_size);
341         dest_ptr += dest->element_size;
342         int current_dim = (int)dest->
343         ndim - 1;
344         while(current_dim >= 0) {
345             coords[current_dim]++;
346             if (coords[current_dim] <
347                 dest->shape[current_dim]
348                 ) break;
349             coords[current_dim] = 0;
350             current_dim--;
351         }
352         if (current_dim < 0) break;
353     } while(true);
354     free(coords);
355     return true;
356 }
357
358 bool tensor_op_concatenate(AgenkTensor*

```

```

dest, const AgenkTensor** srcs, size
_t num_srcs, size_t axis) {
    TENSOR_VALIDATE(dest);
    if (axis >= dest->ndim) return false
    ;

    size_t dest_axis_offset = 0;

    for (size_t i = 0; i < num_srcs; ++i
    ) {
        const AgenkTensor* src = srcs[i
        ];
        TENSOR_VALIDATE(src);

        size_t* slice_offsets = (size_t
        *)calloc(dest->ndim, sizeof(
        size_t));
        if(!slice_offsets) return false;
        slice_offsets[axis] = dest_axis_
        offset;

        AgenkTensor* dest_slice_view =
        tensor_slice(dest, slice_
        offsets, src->shape);
        free(slice_offsets);
        if (!dest_slice_view) return
        false;

        tensor_op_copy_from_view(dest_
        slice_view, src);
        tensor_slice_free(dest_slice_
        view);

        dest_axis_offset += src->shape[
        axis];
    }

    return true;
}

// --- Non-Owning Slice Helpers ---

/**
 * @brief Creates a new tensor header
 * that is a "view" into a slice of a
 * parent tensor.
 *
 * * THIS IS A METADATA-ONLY OPERATION. It
 * does not copy any of the
 * underlying
 * * tensor data. The returned tensor's
 * data pointer will point into the
 * parent
 * * tensor's data buffer.
 *
 * * IMPORTANT: The returned tensor is a "
 * non-owning" view. The caller must
 * NOT
 * * call tensor_free() on it. The memory
 * is still managed by the original
 * parent
 * * tensor. A separate `view_free()`
 * function would be needed for a
 * complete view system.
 *
 * * For now, this function is for
 * demonstrating the power of strides.
 *
 * * A full implementation would track
 * memory ownership to prevent double-
 * frees.
 *
 *

```

```

379 * @param parent The source tensor to
    slice from.
380 * @param offsets An array of starting
    indices for the slice in each
    dimension.
381 * @param shape An array of sizes for
    the slice in each dimension.
382 * @return A new AgenkTensor struct
    configured as a view, or NULL on
    failure.
383 */
384 AgenkTensor* tensor_slice(const
    AgenkTensor* parent, const size_t*
    offsets, const size_t* shape) {
385     if (!parent || !offsets || !shape) {
386         TENSOR_LOG_ERROR("Invalid
            arguments: parent, offsets,
            or shape is NULL.");
387         return NULL;
388     }
389     TENSOR_VALIDATE(parent);
390
391     #ifndef NDEBUG
392     for (size_t i = 0; i < parent->ndim;
        ++i) {
393         if ((offsets[i] + shape[i]) >
            parent->shape[i]) {
394             TENSOR_LOG_ERROR("Slice [%zu
                , %zu] is out of bounds
                for dimension %zu with
                size %zu.",
395                 offsets[i], offsets[i] +
                shape[i], i, parent
                ->shape[i]);
396             return NULL;
397         }
398     }
399     #endif
400
401     AgenkTensor* slice = (AgenkTensor*)
        malloc(sizeof(AgenkTensor));
402     if (!slice) return NULL;
403
404     slice->shape = (size_t*)malloc(
        parent->ndim * sizeof(size_t));
405     if (!slice->shape) { free(slice);
        return NULL; }
406
407     slice->strides = (size_t*)malloc(
        parent->ndim * sizeof(size_t));
408     if (!slice->strides) { free(slice->
        shape); free(slice); return NULL
        ; }
409
410     slice->magic = AGENK_TENSOR_MAGIC;
411     slice->ndim = parent->ndim;
412     slice->data_type = parent->data_type
        ;
413     slice->element_size = parent->
        element_size;
414
415     memcpy(slice->shape, shape, parent->
        ndim * sizeof(size_t));
416     memcpy(slice->strides, parent->
        strides, parent->ndim * sizeof(
        size_t));
417
418     slice->data = tensor_get_element_ptr
        (parent, offsets);
419

```

```

420     size_t num_elements;
421     internal_calculate_num_elements(
        slice->shape, slice->ndim, &num_
        elements);
422     slice->data_size_bytes = num_
        elements * slice->element_size;
423
424     return slice;
425 }
426
427 void tensor_slice_free(AgenkTensor*
    slice) {
428     if (slice) {
429         TENSOR_VALIDATE(slice);
430         free(slice->shape);
431         free(slice->strides);
432         free(slice);
433     }
434 }

```

Listing 27: Implementation of the optimized computational kernels.

```

1 /**
2  * @file tensor_materialize.c
3  * @brief Implements the full execution
    engine for lazy AgenkTensorView
    graphs.
4
5  * This file contains the view_
    materialize() function, which is
    the "compiler"
6  * and "executor" for the computational
    graphs built by the view management
    API.
7  * It uses a recursive approach to
    traverse the graph, materialize the
    necessary
8  * source data, and then apply the
    specified structural and arithmetic
    operations.
9
10 * @version 1.0.0
11 * @author Ankush Yadav, Ankit Yadav,
    AuctaSapience
12 */
13
14 #include "private/tensor_private.h"
15 #include <stdlib.h>
16 #include <string.h>
17
18 //=====
19 // SECTION 1: FORWARD
20 // DECLARATIONS & STUBS
21 //=====
22
23 static AgenkTensor* materialize_
    recursive(const AgenkTensorView*
    view);
24
25 // --- STUB for the Memory Subsystem (
    Updated for comprehensive testing)
    ---
26 static void* memory_stub_get_blob(uint64
    _t memory_id, uint32_t layer_hash_id
    , size_t* out_size) {
27     (void)layer_hash_id;
28     char filename[256];
29

```



```

30 // Logic to handle multiple test
    files based on memory_id
31 if (memory_id == 1) {
32     snprintf(filename, sizeof(
        filename), "test_blob_a.bin"
    );
33 } else if (memory_id == 2) {
34     snprintf(filename, sizeof(
        filename), "test_blob_b.bin"
    );
35 } else if (memory_id == 123) {
36     snprintf(filename, sizeof(
        filename), "test_blob_123.
        bin");
37 } else {
38     TENSOR_LOG_ERROR("MEMORY_STUB:
        Unknown memory ID %llu", (
        unsigned long long)memory_id
    );
39     return NULL;
40 }
41
42 FILE* f = fopen(filename, "rb");
43 if (!f) {
44     TENSOR_LOG_ERROR("MEMORY_STUB:
        Failed to open dummy data
        file '%s'", filename);
45     return NULL;
46 }
47
48 fseek(f, 0, SEEK_END);
49 long size = ftell(f);
50 fseek(f, 0, SEEK_SET);
51 if (size <= 0) { fclose(f); return
    NULL; }
52
53 char* buffer = (char*)malloc(size);
54 if (!buffer) { fclose(f); return
    NULL; }
55
56 if (fread(buffer, 1, size, f) != (
    size_t)size) {
57     free(buffer);
58     fclose(f);
59     return NULL;
60 }
61
62 fclose(f);
63 *out_size = size;
64 return buffer;
65 }
66
67 //=====
68 // SECTION 2: PUBLIC
69 // API FUNCTION
70 //=====
71
72
73 AgenkTensor* view_materialize(const
    AgenkTensorView* view) {
74     if (!view) {
75         TENSOR_LOG_ERROR("Cannot
            materialize a NULL view.");
76         return NULL;
77     }
78     VIEW_VALIDATE(view);
79     return materialize_recursive(view);
80 }
81
82

```

```

83 //=====
84 // SECTION 3: THE RECURSIVE
85 // MATERIALIZATION ENGINE
86 //=====
87
88 static AgenkTensor* materialize_
    recursive(const AgenkTensorView*
    view) {
89     // --- Base Case: SOURCE node ---
90     if (view->op_type == VIEW_OP_SOURCE)
91     {
92         size_t blob_size;
93         void* data_blob = memory_stub_
            get_blob(view->source_memory
            id, view->layer_hash_id, &
            blob_size);
94         if (!data_blob) return NULL;
95
96         // THE FIX: Pass the view's
            content_type to tensor_
            create
97         AgenkTensor* materialized_tensor
            = tensor_create(view->shape
            , view->ndim, view->data_
            type, view->content_type);
98         if (!materialized_tensor) { free
            (data_blob); return NULL; }
99
100         if (materialized_tensor->data_
            size_bytes != blob_size) {
101             TENSOR_LOG_ERROR("Data blob
                size mismatch for memory
                ID %llu. Expected %zu,
                got %zu.", (unsigned long
                long)view->source_
                memory_id, materialized_
                tensor->data_size_bytes,
                blob_size);
102             tensor_free(materialized_
                tensor);
103             free(data_blob);
104             return NULL;
105         }
106
107         memcpy(materialized_tensor->data
            , data_blob, blob_size);
108         free(data_blob);
109         return materialized_tensor;
110     }
111
112     // --- Recursive Step: Materialize
        all parent nodes first. ---
113     if (view->num_parents == 0) {
114         TENSOR_LOG_ERROR("Non-source
            view has zero parents.");
115         return NULL;
116     }
117
118     AgenkTensor** parent_tensors = (
        AgenkTensor**)calloc(view->num_
        parents, sizeof(AgenkTensor*));
119     if (!parent_tensors) return NULL;
120
121     for (size_t i = 0; i < view->num_
        parents; ++i) {
122         parent_tensors[i] = materialize_
            recursive(view->parents[i]);
123         if (!parent_tensors[i]) {
            for (size_t j = 0; j < i; ++
            j) tensor_free(parent_

```

<pre> 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 case VIEW_OP_SLICE: { 173 // A materialized slice 174 // requires creating a non- 175 // owning header and then 176 // performing a deep copy 177 // from it. 178 const AgenkTensor* parent = 179 parent_tensors[0]; 180 const size_t* offsets = (181 const size_t*)view->op_ 182 params; 183 AgenkTensor* slice_view_ 184 header = tensor_slice(185 parent, offsets, view-> 186 shape); 187 if (slice_view_header) { 188 success = tensor_op_copy 189 _from_view(dest_ 190 tensor, slice_view_ 191 header); </pre>	<pre> 163 tensor_slice_free(slice_ 164 view_header); // 165 // Free the temporary 166 // header 167 } 168 break; 169 } 170 case VIEW_OP_TRANSPOSE: 171 case VIEW_OP_BROADCAST: { 172 // Transpose and Broadcast 173 // views are materialized 174 // by performing a deep, 175 // strided copy from the 176 // materialized parent, 177 // which already has the 178 // correct (but non- 179 // contiguous) memory 180 // layout. 181 success = tensor_op_copy_ 182 _from_view(dest_tensor, 183 parent_tensors[0]); 184 break; 185 } 186 case VIEW_OP_CONCATENATE: 187 if (view->op_params) { 188 size_t axis = *(size_t*) 189 view->op_params; 190 success = tensor_op_ 191 concatenate(dest_ 192 tensor, (const 193 AgenkTensor**)parent_ 194 tensors, view->num_ 195 parents, axis); 196 } 197 break; 198 default: 199 TENSOR_LOG_ERROR(" 200 Unsupported view_ 201 operation type in_ 202 materializer: %d", view 203 ->op_type); 204 success = false; 205 break; 206 } 207 208 // --- Cleanup and Return --- 209 for (size_t i = 0; i < view->num_ 210 parents; ++i) { 211 tensor_free(parent_tensors[i]); 212 } 213 free(parent_tensors); 214 215 if (!success) { 216 tensor_free(dest_tensor); 217 return NULL; 218 } 219 220 return dest_tensor; </pre>
---	--

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 lifecycle 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("_[TEST]
46     _%s\n", name)
47
48 #define ASSERT(condition)
49
50     \
51     do {
52
53         \
54         g_tests_run++;
55
56         \
57         if (condition) {

```

```

39         \
40         g_tests_passed++;
41
42     } else {
43
44         \
45         fprintf(stderr, COLOR_RED "[
46             FAIL]_s:%d:_Assertion_
47             failed:_%s\n"
48             \
49             COLOR_RESET, __FILE_
50             \
51             __LINE__, #
52             condition);
53
54         \
55     }
56
57 } while (0)
58
59 #define ASSERT_FLOAT_EQ(a, b) ASSERT(
60     fabs((a) - (b)) < 1e-6)
61
62 // Helper to create a dummy data file
63 // with a predictable pattern for a
64 // given type.
65 void create_dummy_blob_file(const char*
66     filename, size_t size_bytes, uint8_t
67     start_val, AgenkDataType type) {
68     FILE* f = fopen(filename, "wb");
69     if (!f) { perror("Failed_to_create_
70         dummy_blob_file"); return; }
71
72     if (type == AGENK_DATA_TYPE_FLOAT32)
73     {
74         size_t num_elements = size_bytes
75             / sizeof(float);
76         for (size_t i = 0; i < num_
77             elements; ++i) {
78             float val = (float)(start_
79                 val + i);
80             fwrite(&val, sizeof(float),
81                 1, f);
82         }
83     } else { // Default to byte-wise for
84         UINT8 and others
85         for (size_t i = 0; i < size_
86             bytes; ++i) {
87             fputc((start_val + i) % 256,
88                 f);
89         }
90     }
91     fclose(f);
92 }
93
94 //=====
95 // SECTION 2: TEST SUITES
96 //=====
97
98 void test_suite_lifecycle_and_metadata()
99 {
100     TEST_SUITE_START("Lifecycle_&_
101         Metadata");
102
103     TEST_CASE("Basic_2D_tensor_creation_
104         (like_a_grayscale_image)");
105     size_t shape2d[] = {10, 20};
106     // THE FIX: Add the fourth argument
107     // for content_type
108     AgenkTensor* t2d = tensor_create(

```

```

118     shape2d, 2, AGENK_DATA_TYPE
119     UINT8, AGENK_CONTENT_TYPE_IMAGE_
120     PIXELS);
121     ASSERT(t2d != NULL);
122     if (t2d) {
123         ASSERT(t2d->content_type ==
124             AGENK_CONTENT_TYPE_IMAGE_
125             PIXELS);
126         ASSERT(tensor_get_ndim(t2d) ==
127             2);
128         const size_t* shape_ptr = tensor
129             get_shape(t2d);
130         ASSERT(shape_ptr[0] == 10 &&
131             shape_ptr[1] == 20);
132         ASSERT(t2d->data_type == AGENK_
133             DATA_TYPE_UINT8);
134         ASSERT(t2d->data_size_bytes ==
135             10 * 20 * sizeof(uint8_t));
136         ASSERT(t2d->strides[0] == 20 *
137             sizeof(uint8_t));
138         ASSERT(t2d->strides[1] == 1 *
139             sizeof(uint8_t));
140         tensor_free(t2d);
141     }
142     TEST_CASE("Complex_4D_tensor_
143         creation_(like_a_batch_of_videos
144         )");
145     size_t shape4d[] = {8, 10, 192,
146         108};
147     // THE FIX: Add the fourth argument
148     AgenkTensor* t4d = tensor_create(
149         shape4d, 4, AGENK_DATA_TYPE_
150         FLOAT32, AGENK_CONTENT_TYPE_RAW_
151         BYTES);
152     ASSERT(t4d != NULL);
153     if (t4d) {
154         ASSERT(tensor_get_ndim(t4d) ==
155             4);
156         ASSERT(t4d->data_type == AGENK_
157             DATA_TYPE_FLOAT32);
158         ASSERT(t4d->strides[0] == 10 *
159             192 * 108 * sizeof(float));
160         ASSERT(t4d->strides[1] == 192 *
161             108 * sizeof(float));
162         ASSERT(t4d->strides[2] == 108 *
163             sizeof(float));
164         ASSERT(t4d->strides[3] == 1 *
165             sizeof(float));
166         tensor_free(t4d);
167     }
168     TEST_CASE("Tensor_with_a_zero-sized_
169         dimension");
170     size_t shape_zero[] = {10, 0, 20};
171     // THE FIX: Add the fourth argument
172     AgenkTensor* t_zero = tensor_create(
173         shape_zero, 3, AGENK_DATA_TYPE_
174         INT16, AGENK_CONTENT_TYPE_AUDIO_
175         PCM);
176     ASSERT(t_zero != NULL);
177     if (t_zero) {
178         ASSERT(t_zero->data_size_bytes
179             == 0);
180         tensor_free(t_zero);
181     }
182 }
183 void test_suite_error_handling() {
184     TEST_SUITE_START("Error_Handling");
185     size_t shape[] = {10, 20};
186     TEST_CASE("Creation_with_NULL_shape
187         ");
188     // THE FIX: Add the fourth argument
189     ASSERT(tensor_create(NULL, 2, AGENK_
190         DATA_TYPE_UINT8, AGENK_CONTENT_
191         TYPE_RAW_BYTES) == NULL);
192     TEST_CASE("Creation_with_zero_
193         dimensions");
194     // THE FIX: Add the fourth argument
195     ASSERT(tensor_create(shape, 0, AGENK
196         DATA_TYPE_UINT8, AGENK_CONTENT_
197         TYPE_RAW_BYTES) == NULL);
198     TEST_CASE("Creation_with_invalid_
199         data_type");
200     // THE FIX: Add the fourth argument
201     ASSERT(tensor_create(shape, 2, AGENK
202         DATA_TYPE_UNDEFINED, AGENK_
203         CONTENT_TYPE_RAW_BYTES) == NULL)
204         ;
205     ASSERT(tensor_create(shape, 2, (
206         AgenkDataType)99, AGENK_CONTENT_
207         TYPE_RAW_BYTES) == NULL);
208     TEST_CASE("Creation_that_would_
209         overflow_size_t");
210     size_t huge_shape[] = {SIZE_MAX / 2,
211         4};
212     // THE FIX: Add the fourth argument
213     ASSERT(tensor_create(huge_shape, 2,
214         AGENK_DATA_TYPE_UINT8, AGENK_
215         CONTENT_TYPE_RAW_BYTES) == NULL)
216         ;
217     TEST_CASE("Freeing_a_NULL_pointer");
218     tensor_free(NULL);
219     ASSERT(1);
220 }
221 void test_suite_element_access() {
222     TEST_SUITE_START("Element_Access");
223     TEST_CASE("Accessing_elements_in_a_3
224         D_tensor");
225     size_t shape3d[] = {2, 3, 4};
226     // THE FIX: Add the fourth argument
227     AgenkTensor* t3d = tensor_create(
228         shape3d, 3, AGENK_DATA_TYPE_
229         INT16, AGENK_CONTENT_TYPE_RAW_
230         BYTES);
231     ASSERT(t3d != NULL);
232     if (!t3d) return;
233     int16_t* data_ptr = (int16_t*)t3d->
234         data;
235     for (size_t z = 0; z < 2; ++z) {
236         for (size_t y = 0; y < 3; ++y) {
237             for (size_t x = 0; x < 4; ++
238                 x) {
239                 data_ptr[z * (3*4) + y *
240                     4 + x] = (int16_t)(
241                     z * 100 + y * 10 + x
242                     );
243             }
244         }
245     }
246     for (size_t z = 0; z < 2; ++z) {
247         for (size_t y = 0; y < 3; ++y) {

```

```

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
174 TEST_CASE("Out-of-bounds_access_should_return_NULL_(in_Debug_builds)");
175 #ifndef NDEBUG
176     size_t oob_coords[] = {2, 0, 0};
177     ASSERT(tensor_get_element_ptr(t3d, oob_coords) == NULL);
178 #endif
179     tensor_free(t3d);
180 }
181
182 void test_suite_arithmetic_ops() {
183     TEST_SUITE_START("Arithmetic_Operations");
184     size_t shape[] = {10, 10};
185     // THE FIX: Add the fourth argument
186     AgenkTensor* t_a = tensor_create(shape, 2, AGENK_DATA_TYPE_FLOAT32, AGENK_CONTENT_TYPE_RAW_BYTES);
187     AgenkTensor* t_b = tensor_create(shape, 2, AGENK_DATA_TYPE_FLOAT32, AGENK_CONTENT_TYPE_RAW_BYTES);
188     AgenkTensor* t_dest = tensor_create(shape, 2, AGENK_DATA_TYPE_FLOAT32, AGENK_CONTENT_TYPE_RAW_BYTES);
189     ASSERT(t_a && t_b && t_dest);
190     if (!t_a || !t_b || !t_dest) {
191         tensor_free(t_a); tensor_free(t_b); tensor_free(t_dest);
192         return;
193     }
194
195     for (size_t i = 0; i < 100; ++i) {
196         ((float*)t_a->data)[i] = (float)i;
197         ((float*)t_b->data)[i] = (float)(i * 2);
198     }
199
200     TEST_CASE("Element-wise_ADD_operation");
201     ASSERT(tensor_op_add(t_dest, t_a, t_b));
202     for (size_t i = 0; i < 100; ++i) {
203         ASSERT_FLOAT_EQ(((float*)t_dest->data)[i], (float)i + (float)(i * 2));
204     }
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); tensor_free(t_dest);
212 }
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_(stubbed)");
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 /**
251  * @brief Test suite for the full
252  *        computational graph pipeline (
253  *        integration tests).
254  */
255 void test_suite_computational_graphs() {
256     TEST_SUITE_START("Computational_
257                     Graph_(Integration_Tests)");
258
259     size_t blob_size_floats = 10 * 20 *
260         sizeof(float);
261     create_dummy_blob_file("test_blob_a.
262                           bin", blob_size_floats, 0, AGENK
263                           DATA_TYPE_FLOAT32);
264     create_dummy_blob_file("test_blob_b.
265                           bin", blob_size_floats, 100,
266                           AGENK_DATA_TYPE_FLOAT32);
267
268     AgenkTensorView* v_a = view_create_
269         source(1, "source_a");
270     AgenkTensorView* v_b = view_create_
271         source(2, "source_b");
272     ASSERT(v_a && v_b);
273     if (!v_a || !v_b) return;
274
275     TEST_CASE("Materializing_a_simple_
276             ADD_graph");
277     AgenkTensorView* v_add = view_create_
278         add(v_a, v_b);
279     ASSERT(v_add != NULL);
280     if (v_add) {
281         AgenkTensor* t_add = view_
282             materialize(v_add);
283         ASSERT(t_add != NULL);
284         if (t_add) {
285             size_t coords[] = {5, 5};
286             float* val_ptr = (float*)
287                 tensor_get_element_ptr(t_
288                     add, coords);
289             float expected = (5.0f *
290                 20.0f + 5.0f) + (100.0f
291                 + 5.0f * 20.0f + 5.0f);
292             ASSERT_FLOAT_EQ(*val_ptr,
293                 expected);
294             tensor_free(t_add);
295         }
296         view_free(v_add);
297     }
298
299     TEST_CASE("Materializing_a_complex_
300             nested_graph:(slice(A)_+_slice(
301             B))*_2.0");
302     size_t offsets[] = {2, 3};
303     size_t slice_shape[] = {5, 5};
304     AgenkTensorView* v_a_slice = view_
305         create_slice(v_a, offsets, slice_
306             shape);
307     AgenkTensorView* v_b_slice = view_
308         create_slice(v_b, offsets, slice_
309             shape);
310     AgenkTensorView* v_add_slices = view_
311         create_add(v_a_slice, v_b_slice
312             );
313     float scalar = 2.0f;
314     AgenkTensorView* v_final = view_
315         create_scale(v_add_slices,

```

```

316         scalar);
317     ASSERT(v_a_slice && v_b_slice && v_
318         add_slices && v_final);
319
320     if (v_final) {
321         AgenkTensor* t_final = view_
322             materialize(v_final);
323         ASSERT(t_final != NULL);
324         if (t_final) {
325             ASSERT(tensor_get_ndim(t_
326                 final) == 2);
327             ASSERT(tensor_get_shape(t_
328                 final)[0] == 5 && tensor_
329                 get_shape(t_final)[1]
330                 == 5);
331
332             size_t final_coords[] =
333                 {1,1}; // Corresponds to
334                 original (3,4)
335             float* val_ptr = (float*)
336                 tensor_get_element_ptr(t_
337                     final, final_coords);
338             float val_a_orig = (3.0f *
339                 20.0f + 4.0f);
340             float val_b_orig = 100.0f +
341                 val_a_orig;
342             ASSERT_FLOAT_EQ(*val_ptr, (
343                 val_a_orig + val_b_orig)
344                 * 2.0f);
345
346             tensor_free(t_final);
347         }
348         view_free(v_final);
349     }
350
351     view_free(v_a); view_free(v_b);
352     view_free(v_a_slice); view_free(v_b_
353         slice);
354     view_free(v_add_slices);
355 }
356
357 /**
358  * @brief Tests the correct handling of
359  *        logical content types.
360  */
361 void test_suite_content_types() {
362     TEST_SUITE_START("Logical_Content_
363                     Types");
364
365     TEST_CASE("Creating_a_tensor_to_
366             represent_a_UTF-8_string");
367     const char* text = "hello";
368     size_t text_len = strlen(text);
369     size_t shape[] = {text_len}; // A 1D
370     tensor
371
372     // A text string is physically a
373     // sequence of UINT8, but logically
374     // it's UTF8_TEXT
375     AgenkTensor* text_tensor = tensor_
376         create(shape, 1, AGENK_DATA_TYPE_
377             UINT8, AGENK_CONTENT_TYPE_UTF8_
378             TEXT);
379     ASSERT(text_tensor != NULL);
380
381     if (text_tensor) {
382         ASSERT(text_tensor->data_type ==
383             AGENK_DATA_TYPE_UINT8);
384         ASSERT(text_tensor->content_type
385             == AGENK_CONTENT_TYPE_UTF8_

```

```

332         TEXT);
333         // Copy the string data in and
334         // verify
335         memcpy(text_tensor->data, text,
336                text_len);
337         ASSERT(memcmp(text_tensor->data,
338                        text, text_len) == 0);
339     }
340 }
341
342 //=====
343 // SECTION 3: TEST RUNNER
344 //=====
345
346 int main() {
347     printf("=====\n");
348     printf("Running_AGENK_Tensor_
349           Substrate_Test_Suite\n");
350     printf("=====\n");
351
352     test_suite_lifecycle_and_metadata();
353     printf("\n");
354     test_suite_error_handling();
355     printf("\n");
356     test_suite_element_access();
357     printf("\n");
358     test_suite_content_types();
359     printf("\n");
360     test_suite_arithmetic_ops();
361     printf("\n");
362     test_suite_views_and_materialization
363     ();
364     printf("\n");
365
366     printf("=====\n");
367     printf("==TEST_SUMMARY==\n");
368     printf("=====\n");
369     if (g_tests_passed == g_tests_run) {
370         printf(COLOR_GREEN "SUCCESS: All
371               %d tests passed.\n" COLOR_
372               RESET, g_tests_run);
373         return 0;
374     } else {
375         fprintf(stderr, COLOR_RED "
376               FAILURE: %d out of %d tests
377               failed.\n" COLOR_RESET,
378               g_tests_run - g_tests_
379               passed, g_tests_run);
380
381         return 1;
382     }
383 }

```

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 AGENT
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(AGENTK 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 ${
19   CMAKE_BINARY_DIR}/lib)
20 set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${
21   CMAKE_BINARY_DIR}/bin)
```

```
21 if(NOT CMAKE_BUILD_TYPE)
22   set(CMAKE_BUILD_TYPE Debug)
23 endif()
24 message(STATUS "Build type set to: ${
25   CMAKE_BUILD_TYPE}")
26
27 option(AGENTK_ENABLE_PYTHON "Enable
28   Python runtime and native bridge" ON
29 )
30
31 option(AGENTK_ENABLE_NODEJS "Enable
32   NodeJS/NPM runtime" OFF) #
33   Placeholder for the future
34
35 # =====
36 # SECTION 2: VENDORED &
37 # SYSTEM DEPENDENCY MANAGEMENT
38 # =====
39 # --- System Libraries ---
40 find_package(Threads REQUIRED)
41 find_library(MATH_LIBRARY m)
42
43 # --- Dependency Discovery ---
44 find_package(Threads REQUIRED)
45 if(AGENTK_ENABLE_PYTHON)
46   message(STATUS "Python support is
47     ENABLED.")
48   find_package(Python3 COMPONENTS
49     Development NumPy REQUIRED)
50 else()
51   message(STATUS "Python support is
52     DISABLED.")
53 endif()
54
55 # --- mbed TLS (vendored from deps/) ---
56 set(ENABLE_TESTING OFF CACHE BOOL "
57   Disable mbed TLS tests")
58 set(ENABLE_PROGRAMS OFF CACHE BOOL "
59   Disable mbed TLS programs")
60 add_subdirectory(deps/mbedtls)
61 set(MBEDTLS_LIBRARIES mbedtls mbedx509
62   mbedcrypto)
63 set(MBEDTLS_LIBRARY_DIR ${CMAKE_ARCHIVE_
64   OUTPUT_DIRECTORY})
65 set(MBEDTLS_INCLUDE_DIR ${CMAKE_CURRENT_
66   SOURCE_DIR}/deps/mbedtls/include)
67
68 # --- libcurl (vendored from deps/,
69   built with custom command) ---
70 ExternalProject_Add(
71   curl_local_build
72   SOURCE_DIR ${CMAKE_CURRENT_SOURCE_
73     DIR}/deps/curl
74   BINARY_DIR ${CMAKE_BINARY_DIR}/curl-
75     build
76   # Use bash -c to create a reliable
77   # shell environment for the
78   # configure script
79   CONFIGURE_COMMAND bash -c
80     "LD_FLAGS='-L${MBEDTLS_LIBRARY_
81       DIR}' LIBS='-lmbedtls -
82       lmbedx509 -lmbedcrypto -
83       lpthread' CPPFLAGS='-I${
84       MBEDTLS_INCLUDE_DIR}'<
85       SOURCE_DIR>/configure --
86       disable-shared --with-ssl --
87       without-zlib"
88   BUILD_COMMAND make
89   INSTALL_COMMAND ""
90   DEPENDS ${MBEDTLS_LIBRARIES}
```

```

66 )
67 add_library(libcurl STATIC IMPORTED
  GLOBAL)
68 set_target_properties(libcurl PROPERTIES
69   IMPORTED_LOCATION "${CMAKE_BINARY_
    DIR}/curl-build/lib/.libs/
    libcurl.a"
70 )
71 add_dependencies(libcurl curl_local_
  build)
72
73 # --- Other Vended Libraries ---
74 set(FLATCC_TEST OFF CACHE BOOL "")
75 set(FLATCC_SAMPLES OFF CACHE BOOL "")
76 add_subdirectory(deps/flatcc)
77 add_library(lmdb STATIC deps/lmdb/mdb.c
  deps/lmdb/midl.c)
78 target_include_directories(lmdb PUBLIC $
  {CMAKE_CURRENT_SOURCE_DIR}/deps/lmdb
  )
79 add_library(md5 STATIC deps/md5/md5.c)
80 target_include_directories(md5 PUBLIC ${
  CMAKE_CURRENT_SOURCE_DIR}/deps/md5)
81 set(CJSON_LIB "")
82 if(EXISTS ${CMAKE_CURRENT_SOURCE_DIR}/
  deps/cJSON/cJSON.c)
83   add_library(cjson STATIC deps/cJSON/
    cJSON.c)
84   target_include_directories(cjson
    PUBLIC ${CMAKE_CURRENT_SOURCE_
      DIR}/deps/cJSON)
85   set(CJSON_LIB cjson)
86   set(CJSON_FOUND TRUE)
87 endif()
88 ExternalProject_Add(tcc_external_project
  SOURCE_DIR ${CMAKE_CURRENT_SOURCE_
    DIR}/deps/tcc CONFIGURE_COMMAND ""
  BUILD_COMMAND "" INSTALL_COMMAND "")
89 add_custom_target(build_tcc_library ALL
  COMMAND ${CMAKE_COMMAND} -E chdir ${
    CMAKE_CURRENT_SOURCE_DIR}/deps/tcc
  ./configure --libdir=${CMAKE_CURRENT
    SOURCE_DIR}/deps/tcc/lib --extra-
    cflags="-fPIC" COMMAND ${CMAKE_
    COMMAND} -P ${CMAKE_CURRENT_BINARY_
    DIR}/cmake_script_patch_tcc_makefile
  .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/lib
  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/
  DEPENDS tcc_external_project)
90 file(WRITE ${CMAKE_CURRENT_BINARY_DIR}/
  cmake_script_patch_tcc_makefile.
  cmake "set(TCC_LIB_MAKEFILE_PATH \"${
    CMAKE_CURRENT_SOURCE_DIR}/deps/tcc/
    lib/Makefile\")\nfile(READ ${TCC_
    LIB_MAKEFILE_PATH} TCC_LIB_MAKEFILE_
    CONTENT)\nstring(REPLACE \"bcheck.o
    \" \"\" #bcheck.o) \"TCC_LIB_MAKEFILE_
    PATCHED\" \"${TCC_LIB_MAKEFILE_
    CONTENT}\")\nfile(WRITE ${TCC_LIB_
    MAKEFILE_PATH} \"${TCC_LIB_MAKEFILE

```

```

    PATCHED}\")\nmessage(STATUS \"
    Patched TCC lib/Makefile to disable
    bcheck.\")\n")
91 add_library(libtcc STATIC IMPORTED
  GLOBAL)
92 set_target_properties(libtcc PROPERTIES
  IMPORTED_LOCATION "${CMAKE_BINARY_
    DIR}/lib/libtcc.a")
93 add_dependencies(libtcc build_tcc_
  library)
94
95 # =====
96 # SECTION 3: AUTOMATED CODE
97 # GENERATION (from FlatBuffers)
98 # =====
99 set(FBS_SCHEMA ${CMAKE_CURRENT_SOURCE_
  DIR}/src/object_store/core/object.
  fbs)
100 set(GENERATED_INCLUDE_DIR ${CMAKE_
  CURRENT_BINARY_DIR}/generated/object_
  store)
101 file(MAKE_DIRECTORY ${GENERATED_INCLUDE_
  DIR})
102 set(GENERATED_FILES
103   ${GENERATED_INCLUDE_DIR}/object_
    builder.h
104   ${GENERATED_INCLUDE_DIR}/object_
    reader.h
105   ${GENERATED_INCLUDE_DIR}/object_
    verifier.h
106   ${GENERATED_INCLUDE_DIR}/flatbuffers
    _common.h
107 )
108 add_custom_command(
109   OUTPUT ${GENERATED_FILES}
110   COMMAND $<TARGET_FILE:flatcc_cli> -a
    -o ${GENERATED_INCLUDE_DIR} ${
    FBS_SCHEMA}
111   DEPENDS ${FBS_SCHEMA} flatcc_cli
112   COMMENT "Generating C headers from
    FlatBuffers schema"
113 )
114 add_custom_target(generate_object_
  headers DEPENDS ${GENERATED_FILES})
115
116 # =====
117 # SECTION 4: AGENK CORE
118 # LIBRARY DEFINITIONS
119 # =====
120
121 # --- LIBRARY 1: object store (The
  DataBase) ---
122 # --- NEW LIBRARY: object_store (The
  Persistence Layer) ---
123
124 add_library(object_store STATIC
125   src/object_store/core/fb_serializer.
  c
126   src/object_store/core/object_store.c
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     /include
204 )
205 if(AGENK_ENABLE_PYTHON)
206     target_include_directories(agenk_
207         core PUBLIC
208         ${Python3_INCLUDE_DIRS}
209         ${Python3_NumPy_INCLUDE_DIRS}
210     )
211 endif()
212 target_link_libraries(agenk_core PUBLIC
213     agenk_snippet agenk_tensor class_
214     framework
215 )
216 if(AGENK_ENABLE_PYTHON)
217     target_link_libraries(agenk_core
218         PRIVATE Python3::Python Python3
219         ::NumPy)
220 endif()
221 add_library(agenk_sensation STATIC
222     src/senses/core/sense_manager.c
223     src/senses/core/sense_registry.c
224     # ADD THESE MISSING SOURCE FILES
225     src/senses/builder/build_controller.
226     c
227     src/senses/builder/tool_check.c
228 )
229 target_include_directories(agenk_
230     sensation PUBLIC
231     ${CMAKE_CURRENT_SOURCE_DIR}/src/
232     senses/include
233 )
234 target_compile_definitions(agenk_
235     sensation PRIVATE AGENK_SOURCE_DIR="
236     ${CMAKE_SOURCE_DIR}")
237 target_link_libraries(agenk_sensation
238     PRIVATE
239     agenk_core
240     md5
241     libtcc
242     object_store # This must be here
243 )
244 # --- LIBRARY 6: agenk_execution (The "
245 # CPU" / Execution Engine) ---
246 set(EXECUTION_SOURCES
247     src/execution/core/engine.c
248     src/execution/builder/dependency_
249     manager.c

```



```

234     src/execution/runtimes/runtime_c.c
235     src/execution/runtimes/runtime_shell
        .c
236 )
237 if(AGENK_ENABLE_PYTHON)
238     list(APPEND EXECUTION_SOURCES src/
        execution/runtimes/runtime_
        python.c)
239 endif()
240 add_library(agenk_execution STATIC ${
    EXECUTION_SOURCES})
241
242 target_include_directories(agenk_
    execution PUBLIC
243     ${CMAKE_CURRENT_SOURCE_DIR}/src/
    execution/include
244     PRIVATE
245     ${CMAKE_CURRENT_SOURCE_DIR}/src/
    execution
246 )
247
248 target_compile_definitions(agenk_
    execution PRIVATE
249     AGENK_SOURCE_DIR="${CMAKE_SOURCE_DIR
    }"
250     AGENK_BINARY_DIR="${CMAKE_BINARY_DIR
    }"
251     AGENK_TCC_PATH="${CMAKE_CURRENT_
    SOURCE_DIR}/deps/tcc"
252     # THE FIX: Provide a reliable path
    to the *built* TCC libraries
253     AGENK_TCC_LIB_PATH="${CMAKE_ARCHIVE_
    OUTPUT_DIRECTORY}"
254     # Provide include directories for
    runtime compilation
255     AGENK_CORE_INCLUDE_DIR="${CMAKE_
    CURRENT_SOURCE_DIR}/src/core/
    include"
256     AGENK_SNIPPET_INCLUDE_DIR="${CMAKE_
    CURRENT_SOURCE_DIR}/src/snippets
    /include"
257 )
258
259 if(AGENK_ENABLE_PYTHON)
260     # Add compile definitions needed for
    the Python runtime.
261     # - AGENK_PYTHON_SUPPORT enables all
    Python-related code blocks.
262     # - PYTHON_SITE_PACKAGES tells the C
    code where to find system
    packages.
263     target_compile_definitions(agenk_
        execution PRIVATE
264         AGENK_PYTHON_SUPPORT
265         "PYTHON_SITE_PACKAGES=\"${CMAKE_
            PYTHON3_SITELIB}\"")
266 )
267 endif()
268
269 target_link_libraries(agenk_execution
    PRIVATE
270     agenk_core agenk_snippet libtcc
        Threads::Threads md5
271 )
272 if(AGENK_ENABLE_PYTHON)
273     target_link_libraries(agenk_
        execution PRIVATE Python3::
        Python)
274 endif()
275
276 # =====
277 # SECTION 5: EXECUTABLE
278 # TARGETS (TESTS & EXAMPLES)
279 # =====
280
281 # --- EXECUTABLES ---
282 add_executable(test_tensors src/tensors/
    test/test_tensors.c)
283 target_link_libraries(test_tensors
    PRIVATE agenk_tensor)
284
285 add_executable(test_sensation src/senses
    /test/test_sensation.c)
286 target_link_libraries(test_sensation
    PRIVATE agenk_sensation)
287
288 add_executable(test_sensation_build src/
    senses/test/test_sensation_build.c)
289 target_link_libraries(test_sensation_
    build PRIVATE agenk_sensation)
290
291 add_executable(test_sense_registry src/
    senses/test/test_sense_registry.c)
292 target_link_libraries(test_sense_
    registry PRIVATE agenk_sensation ${
    MATH_LIBRARY})
293
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/

```

```

318 snippets/test/test_snippets.c)
target_link_libraries(test_snippets
PRIVATE
319 agenk_snippet
320 class_framework # For CLASS_INIT
321 object_store # For the registry's
backend
322 )
323
324 add_executable(test_execution src/
execution/test/test_execution.c)
325
326 if(AGENK_ENABLE_PYTHON)
327 target_compile_definitions(test
execution PRIVATE AGENK_PYTHON_
SUPPORT) # <-- ADD THIS LINE
BACK
328 # THE DEFINITIVE FIX for systems
using GNU ld or clang ld.
329 # This consolidates all linking into
a single, ordered command.
330 if(CMAKE_SYSTEM_NAME STREQUAL "Linux"
OR CMAKE_SYSTEM_NAME STREQUAL
"Darwin")
331 target_link_libraries(test_
execution PRIVATE
332 # --- Libraries used directly by
the test C code ---
333 agenk_execution
334 class_framework
335
336 # --- Force-load the python
bridge, THEN provide its
dependencies ---
337 "-Wl,--whole-archive"
338 agenk_core
339 "-Wl,--no-whole-archive"
340 agenk_snippet
341 agenk_tensor
342
343 # --- Provide all other system/
vendored dependencies ---
344 lmbd
345 Threads::Threads
346 Python3::Python
347 Python3::NumPy
348 )
349 else()
350 # Fallback for other systems (e.g
., Windows).
351 target_link_libraries(test_
execution PRIVATE
352 agenk_execution class_
framework agenk_core agenk
snippet
353 agenk_tensor lmbd Threads::
Threads Python3::Python
354 )
355 endif()
356 else()
357 # Link normally if Python is
disabled
358 target_link_libraries(test_execution
PRIVATE
359 agenk_execution class_framework
agenk_core agenk_snippet
360 agenk_tensor lmbd Threads::
Threads
361 )
362 endif()

```

```

363 if(CMAKE_SYSTEM_NAME STREQUAL "Linux")
364 target_link_options(test_execution
PRIVATE "-Wl,--export-dynamic")
365 endif()
366
367
368 # =====
369 # SECTION 6: CTEST
370 # INTEGRATION
371 # =====
372 enable_testing()
373
374 # --- TEST ---
375 add_test(NAME Snippets.
LifecycleAndRegistry COMMAND test_
snippets)
376
377 add_test(NAME Execution.Substrate
COMMAND test_execution)
378
379 add_test(NAME Tensor.Substrate
COMMAND test_tensors)
380
381 add_test(NAME Sensation.Subsystem.Core
COMMAND test_sensation)
382
383 add_test(NAME Sensation.Subsystem.Build
COMMAND test_sensation_build)
384
385 add_test(NAME Sensation.Registry COMMAND
test_sense_registry)
386
387 add_test(NAME ObjectStore.Core
COMMAND test_object_store)
388
389 add_test(NAME Framework.Concurrency
COMMAND test_concurrency)
390
391 add_test(NAME Framework.ErrorHandling
COMMAND test_errors)
392
393 add_test(NAME Framework.Hierarchy
COMMAND test_hierarchy)
394
395 add_test(NAME Framework.Stress
COMMAND test_stress)

```

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”: `AgentkSnippet`, `AgentkDependency`, `AgentkCode`, `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 "
8  * nouns" of the AGENK agent's
9  * capabilities.
10 * It is part of the foundational '
11 * snippets' library, ensuring that all
12 * other
13 * subsystems have a common, stable set
14 * of data formats to work with.
15 *
16 * @version 1.0.0
17 * @author Ankush Yadav, Ankit Yadav,
18 *       AuctaSpience
19 */
20 #ifndef AGENK_SNIPPET_H
21 #define AGENK_SNIPPET_H
22
23 #include <stddef.h>
24 #include <stdint.h>
25 #include <stdbool.h>
26
27 // --- Forward Declarations for Opaque
28 //       Structs ---
29 typedef struct AgenkTensor AgenkTensor;
30 typedef struct AgenkSnippet AgenkSnippet;
31
32 typedef struct AgenkDependency
33     AgenkDependency;
34 typedef struct AgenkCode AgenkCode;
35 typedef struct CodeVariant CodeVariant;
36 typedef struct ApiRequestContext
37     ApiRequestContext;
38 typedef struct ApiRequestParam
39     ApiRequestParam;
40 typedef struct ExecutionResult
41     ExecutionResult;
42 typedef struct ExecutionContext
43     ExecutionContext;
44
45 // --- Core Data Structures ---
46
47 /** @brief Describes a single external
48 *       software dependency. */
49 struct AgenkDependency {
50     enum { DEP_SOURCE, DEP_PIP, DEP_NPM,
51           DEP_SYSTEM } type;
52     char* uri; // THE FIX: Was char, is
53               // now char*
54     char** build_commands;
55     size_t num_build_commands;
56     char** required_artifacts;
57     size_t num_required_artifacts;
58 };

```

```

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

```

```

97 };
98
99 // --- Snippet Registry API ---
100
101 // Forward declare the object store
102 // handle for the init function
103 typedef struct object_store_t object_
104 store_t;
105
106 /**
107  * @brief Initializes the snippet
108  * registry with a handle to the
109  * object store.
110  * @param store A valid, initialized
111  * object_store_t handle.
112  * @return True on success, false on
113  * failure (e.g., failed to create
114  * table).
115  */
116 bool snippet_registry_init(object_store_
117 t* store);
118
119 /**
120  * @brief Shuts down the snippet
121  * registry. (Currently a no-op).
122  */
123 void snippet_registry_shutdown(void);
124
125 /**
126  * @brief Adds a new snippet to the
127  * registry or updates an existing one
128  * .
129  * The object_id of the internal c_
130  * object_t is used as the key.
131  * @param snippet The snippet to save.
132  * @return True on success, false on
133  * failure.
134  */
135 bool snippet_registry_add(const
136 AgenkSnippet* snippet);
137
138 /**
139  * @brief Retrieves a snippet from the
140  * registry by its unique ID.
141  * @param snippet_id The ID of the
142  * snippet to retrieve.
143  * @return A pointer to a new, fully
144  * reconstructed AgenkSnippet. The
145  * caller
146  * is responsible for freeing
147  * this with snippet_free(). Returns
148  * NULL
149  * if not found or on error.
150  */
151 AgenkSnippet* snippet_registry_get_by_id
152 (const char* snippet_id);
153
154 /**
155  * @brief Deletes a snippet from the
156  * registry.
157  * @param snippet_id The ID of the
158  * snippet to delete.
159  * @return True on success or if the
160  * snippet didn't exist, false on
161  * error.
162  */
163 bool snippet_registry_delete(const char*
164 snippet_id);
165

```

```

140 // --- Memory Management APIs ---
141
142 // Snippet Builder
143 AgenkSnippet* snippet_create(const char*
144 description);
145 void snippet_free(AgenkSnippet* s);
146 AgenkDependency* dependency_create(int
147 type, const char* uri);
148 void dependency_add_build_command(
149 AgenkDependency* d, const char*
150 command);
151 void dependency_add_required_artifact(
152 AgenkDependency* d, const char*
153 artifact_path);
154 AgenkCode* code_create(int language,
155 const char* entry_point);
156 void code_add_variant(AgenkCode* c, int
157 os, const char* source);
158 void snippet_add_dependency(AgenkSnippet
159 * s, AgenkDependency* d);
160 void snippet_set_execution_code(
161 AgenkSnippet* s, AgenkCode* c);
162 void snippet_set_test_code(AgenkSnippet*
163 s, AgenkCode* c, const char*
164 expected_output);
165
166 // API Request Context Builder
167 ApiRequestContext* api_request_context_
168 create(const char* url, const char*
169 http_method);
170 void api_request_context_free(
171 ApiRequestContext* ctx);
172 bool api_request_context_add_header(
173 ApiRequestContext* ctx, const char*
174 key, const char* value);
175 bool api_request_context_add_param(
176 ApiRequestContext* ctx, const char*
177 key, const char* value);
178 void api_request_context_set_body(
179 ApiRequestContext* ctx, AgenkTensor*
180 body);
181
182 // Execution Result
183 void execution_result_free(
184 ExecutionResult* result);
185
186 #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 *        AuctaSapiencie
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*

```

```

command) {
46     if (!d || !command) return;
47     d->num_build_commands++;
48     d->build_commands = (char**)realloc(
49         d->build_commands, d->num_build_
50         commands * sizeof(char*));
51     if (!d->build_commands) { d->num_
52         build_commands = 0; return; }
53     d->build_commands[d->num_build_
54         commands - 1] = safe_strdup(
55         command);
56 }
57
58 void dependency_add_required_artifact(
59     AgenkDependency* d, const char*
60     artifact_path) {
61     if (!d || !artifact_path) return;
62     d->num_required_artifacts++;
63     d->required_artifacts = (char**)
64         realloc(d->required_artifacts, d
65         ->num_required_artifacts *
66         sizeof(char*));
67     if (!d->required_artifacts) { d->num
68         _required_artifacts = 0; return;
69     }
70     d->required_artifacts[d->num_
71         required_artifacts - 1] = safe_
72         strdup(artifact_path);
73 }
74
75 static void dependency_free(
76     AgenkDependency* d) {
77     if (!d) return;
78     free(d->uri);
79     for (size_t i = 0; i < d->num_build_
80         commands; ++i) free(d->build_
81         commands[i]);
82     free(d->build_commands);
83     for (size_t i = 0; i < d->num_
84         required_artifacts; ++i) free(d
85         ->required_artifacts[i]);
86     free(d->required_artifacts);
87     free(d);
88 }
89
90 // --- Memory Management for Code ---
91 AgenkCode* code_create(int language,
92     const char* entry_point) {
93     AgenkCode* c = (AgenkCode*)calloc(1,
94         sizeof(AgenkCode));
95     if (!c) return NULL;
96     c->language = language;
97     c->entry_point = safe_strdup(entry_
98         point);
99     return c;
100 }
101
102 void code_add_variant(AgenkCode* c, int
103     os, const char* source) {
104     if (!c || !source) return;
105     c->num_variants++;
106     c->variants = (struct CodeVariant*)
107         realloc(c->variants, c->num_
108         variants * sizeof(struct
109         CodeVariant));
110     if (!c->variants) { c->num_variants
111         = 0; return; }
112     c->variants[c->num_variants - 1].
113         target_os = os;
114     c->variants[c->num_variants - 1].

```



```

        source_code = safe_strdup(source
    );
82 }
83
84 static void code_free(AgenkCode* c) {
85     if (!c) return;
86     free(c->entry_point);
87     for (size_t i = 0; i < c->num_
        variants; ++i) free(c->variants[
            i].source_code);
88     free(c->variants);
89     free(c);
90 }
91
92 // --- Memory Management for Snippets
93
94 ---
95 AgenkSnippet* snippet_create(const char*
    description) {
96     AgenkSnippet* s = (AgenkSnippet*)
        calloc(1, sizeof(AgenkSnippet));
97     if (!s) return NULL;
98     s->description = safe_strdup(
        description);
99     return s;
100 }
101
102 void snippet_add_dependency(AgenkSnippet
    * s, AgenkDependency* d) {
103     if (!s || !d) return;
104     s->num_dependencies++;
105     s->dependencies = (AgenkDependency
        *)realloc(s->dependencies, s->
        num_dependencies * sizeof(
            AgenkDependency));
106     if (!s->dependencies) { s->num_
        dependencies = 0; return; }
107     s->dependencies[s->num_dependencies
        - 1] = d;
108 }
109
110 void snippet_set_execution_code(
    AgenkSnippet* s, AgenkCode* c) {
111     if (s) s->execution_code = c;
112 }
113
114 void snippet_set_test_code(AgenkSnippet*
    s, AgenkCode* c, const char*
    expected_output) {
115     if (s) {
116         s->test_code = c;
117         s->expected_test_output = safe_
            strdup(expected_output);
118     }
119 }
120
121 void snippet_free(AgenkSnippet* s) {
122     if (!s) return;
123     free(s->snippet_id);
124     free(s->description);
125     for (size_t i = 0; i < s->num_
        dependencies; ++i) dependency_
        free(s->dependencies[i]);
126     free(s->dependencies);
127     code_free(s->execution_code);
128     code_free(s->test_code);
129     free(s->expected_test_output);
130     free(s);
131 }
132
133 // --- Memory Management for
    ExecutionResult ---
134 void execution_result_free(
    ExecutionResult* result) {
135     if (!result) return;
136     free(result->stdout_buffer);
137     free(result->stderr_buffer);
138     free(result);
139 }
140
141 // --- API Request Context Builder
    Implementation ---
142
143 ApiRequestContext* api_request_context_
    create(const char* url, const char*
    http_method) {
144     ApiRequestContext* ctx = (
        ApiRequestContext*)calloc(1,
        sizeof(ApiRequestContext));
145     if (!ctx) return NULL;
146     ctx->url = safe_strdup(url);
147     if (!ctx->url) { free(ctx); return
        NULL; }
148     ctx->http_method = safe_strdup(http_
        method);
149     if (!ctx->http_method) { free(ctx->
        url); free(ctx); return NULL; }
150     return ctx;
151 }
152
153 void api_request_context_free(
    ApiRequestContext* ctx) {
154     if (!ctx) return;
155     free(ctx->url);
156     free(ctx->http_method);
157     for (size_t i = 0; i < ctx->num_
        headers; ++i) {
158         free(ctx->headers[i].key);
159         free(ctx->headers[i].value);
160     }
161     free(ctx->headers);
162     for (size_t i = 0; i < ctx->num_
        params; ++i) {
163         free(ctx->query_params[i].key);
164         free(ctx->query_params[i].value)
            ;
165     }
166     free(ctx->query_params);
167     if (ctx->body) {
168         // This is the call that creates
            the dependency
169         tensor_free(ctx->body);
170     }
171     free(ctx);
172 }
173
174 static bool add_param_to_list(
    ApiRequestParam** list, size_t*
    count, size_t* capacity, const char*
    key, const char* value) {
175     if (*count >= *capacity) {
176         *capacity = (*capacity == 0) ? 8
            : *capacity * 2;
177         ApiRequestParam* new_list = (
            ApiRequestParam*)realloc(*
            list, *capacity * sizeof(
                ApiRequestParam));
178         if (!new_list) return false;
179         *list = new_list;
180     }
181     (*list)[*count].key = safe_strdup(
        key);

```

```

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
191 bool 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
215 bool 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
225 void 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 // =====
36 // SECTION 1: PUBLIC API
37 // =====
38
39 bool snippet_registry_init(object_store_
40     t* store) {
41     if (!store) return false;
42     g_store = store;
43     return (object_store_create_table(g_
44         store, SNIPPET_TABLE_NAME) == FB_
45         _SERIALIZER_OK);
46 }
47
48 void snippet_registry_shutdown(void) {
49     g_store = NULL;
50 }
51
52 bool snippet_registry_add(const
53     AgenkSnippet* snippet) {
54     if (!g_store || !snippet || !snippet
55         ->snippet_id) return false;
56     c_object_t* obj = snippet_to_c_
57         object(snippet);
58     if (!obj) return false;
59     unsigned char key[OS_MAX_KEY_SIZE];
60     get_key_for_snippet_id(snippet->
61         snippet_id, key);
62     fb_serializer_status_t status =
63         object_store_put_object(g_store,
64             SNIPPET_TABLE_NAME, key, obj);
65     free_c_object(obj);
66     return (status == FB_SERIALIZER_OK);
67 }
68
69 AgenkSnippet* snippet_registry_get_by_id
70     (const char* snippet_id) {
71     if (!g_store || !snippet_id) return
72         NULL;
73     unsigned char key[OS_MAX_KEY_SIZE];
74     get_key_for_snippet_id(snippet_id,
75         key);
76     c_object_t* obj = NULL;
77     if (object_store_get_object(g_store,
78         SNIPPET_TABLE_NAME, key, &obj)
79         != FB_SERIALIZER_OK) {
80         return NULL;
81     }
82     AgenkSnippet* snippet = snippet_from
83         _c_object(obj);
84     free_c_object(obj);
85     return snippet;
86 }
87
88 bool snippet_registry_delete(const char*
89     snippet_id) {
90     if (!g_store || !snippet_id) return
91         false;
92     unsigned char key[OS_MAX_KEY_SIZE];
93     get_key_for_snippet_id(snippet_id,
94         key);
95     fb_serializer_status_t status =
96         object_store_delete_object(g_
97             store, SNIPPET_TABLE_NAME, key);
98     return (status == FB_SERIALIZER_OK
99         || status == FB_SERIALIZER_ERROR
100             _LMDB_NOT_FOUND);

```

```

79 }
80
81 //
82 // =====
83 // SECTION 2: TRANSLATION LOGIC
84 // =====
85
86 // CORRECTED MACRO
87 #define ADD_FIELD(HEAD, TAIL, NEW_FIELD)
88     do { \
89         /* Use a temporary variable to avoid
90            evaluating NEW_FIELD twice */ \
91         c_field_t* __new_field = (NEW_FIELD)
92         ; \
93         if (__new_field) { /* Only proceed
94            if the new field is not NULL */
95             \
96             if (!*(HEAD)) { \
97                 *(HEAD) = __new_field; \
98             } else { \
99                 (TAIL)->next = __new_field;
100                 \
101                 (TAIL) = __new_field; /* Only
102                    update tail if a field was
103                    added */ \
104             } \
105         } while(0)
106
107 static c_field_t* create_string_array_
108     field(const char* name, char** data,
109         size_t count) {
110     if (count == 0) return NULL;
111     c_field_t* field = (c_field_t*)
112         calloc(1, sizeof(c_field_t));
113     field->name = safe_strdup(name);
114     field->value = (c_field_value_t*)
115         calloc(1, sizeof(c_field_value_t
116             ));
117     field->value->type = C_FIELD_VALUE_
118         TYPE_ARRAY_VALUE;
119     c_field_t* array_tail = NULL;
120     for (size_t i = 0; i < count; ++i) {
121         c_field_t* item = (c_field_t*)
122             calloc(1, sizeof(c_field_t))
123             ;
124         char item_name[32];
125         snprintf(item_name, sizeof(item_
126             name), "%zu", i);
127         item->name = safe_strdup(item_
128             name);
129         item->value = create_c_string_
130             value(data[i]);
131         ADD_FIELD(&field->value->data.
132             array_values, array_tail,
133             item);
134     }
135     return field;
136 }
137
138 static c_object_t* code_to_c_object(
139     const AgenkCode* code) {
140     if (!code) return NULL;
141     c_object_t* obj = calloc(1, sizeof(c_
142         object_t));
143     obj->object_id = safe_strdup("nested

```

```

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("
139         entry_point");
140         f_entry->value = create_c_string
141         _value(code->entry_point);
142         ADD_FIELD(&obj->fields, tail, f_
143         entry);
144     }
145
146     c_field_t* f_variants = calloc(1,
147     sizeof(c_field_t));
148     f_variants->name = safe_strdup("
149     variants");
150     f_variants->value = calloc(1, sizeof
151     (c_field_value_t));
152     f_variants->value->type = C_FIELD_
153     VALUE_TYPE_MAP_VALUE;
154     ADD_FIELD(&obj->fields, tail, f_
155     variants);
156     c_map_pair_t* map_tail = NULL;
157     for (size_t i = 0; i < code->num_
158     variants; i++) {
159         c_map_pair_t* pair = calloc(1,
160         sizeof(c_map_pair_t));
161         char os_key[4];
162         snprintf(os_key, sizeof(os_key),
163         "%d", code->variants[i].
164         target_os);
165         pair->key = safe_strdup(os_key);
166         pair->value = create_c_string_
167         value(code->variants[i].
168         source_code);
169         if (!f_variants->value->data.map
170         _pairs) f_variants->value->
171         data.map_pairs = pair;
172         else map_tail->next = pair;
173         map_tail = pair;
174     }
175     return obj;
176 }
177
178 static c_object_t* dependency_to_c_
179 object(const AgenkDependency* dep) {
180     if (!dep) return NULL;
181     c_object_t* obj = calloc(1, sizeof(c
182     _object_t));
183     obj->object_id = safe_strdup("nested
184     _dependency");
185     obj->table_name =
186     safe_strdup("nested");
187     c_field_t* tail = NULL;
188
189     c_field_t* f_type = calloc(1, sizeof
190     (c_field_t));
191     f_type->name = safe_strdup("type");
192     f_type->value = create_c_int_value(
193
194     dep->type);
195     ADD_FIELD(&obj->fields, tail, f_type
196     );
197
198     if (dep->uri) {
199         c_field_t* f_uri = calloc(1,
200         sizeof(c_field_t));
201         f_uri->name = safe_strdup("uri")
202         ;
203         f_uri->value = create_c_string_
204         value(dep->uri);
205         ADD_FIELD(&obj->fields, tail, f_
206         uri);
207     }
208     ADD_FIELD(&obj->fields, tail, create
209     _string_array_field("build_
210     commands", dep->build_commands,
211     dep->num_build_commands));
212     ADD_FIELD(&obj->fields, tail, create
213     _string_array_field("required_
214     artifacts", dep->required_
215     artifacts, dep->num_required_
216     artifacts));
217     return obj;
218 }
219
220 static c_object_t* snippet_to_c_object(
221 const AgenkSnippet* snippet) {
222     c_object_t* obj = calloc(1, sizeof(c
223     _object_t));
224     obj->object_id = safe_strdup(snippet
225     ->snippet_id);
226     obj->table_name = safe_strdup(
227     SNIPPET_TABLE_NAME);
228     c_field_t* tail = NULL;
229
230     if (snippet->description) {
231         c_field_t* f = calloc(1, sizeof(
232         c_field_t));
233         f->name = safe_strdup("
234         description");
235         f->value = create_c_string_value
236         (snippet->description);
237         ADD_FIELD(&obj->fields, tail, f)
238         ;
239     }
240     if (snippet->expected_test_output) {
241         c_field_t* f = calloc(1, sizeof(
242         c_field_t));
243         f->name = safe_strdup("expected_
244         test_output");
245         f->value = create_c_string_value
246         (snippet->expected_test_
247         output);
248         ADD_FIELD(&obj->fields, tail, f)
249         ;
250     }
251     if (snippet->execution_code) {
252         c_field_t* f = calloc(1, sizeof(
253         c_field_t));
254         f->name = safe_strdup("execution
255         _code");
256         f->value = calloc(1, sizeof(c_
257         field_value_t));
258         f->value->type = C_FIELD_VALUE_
259         TYPE_OBJECT_VALUE;
260         f->value->data.object_val = code
261         _to_c_object(snippet->
262         execution_code);
263         ADD_FIELD(&obj->fields, tail, f)

```

```

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 ");
209     f->value = calloc(1, sizeof(c_
210         field_value_t));
211     f->value->type = C_FIELD_VALUE_
212         TYPE_OBJECT_VALUE;
213     f->value->data.object_val = code
214         _to_c_object(snippet->test_
215         code);
216     ADD_FIELD(&obj->fields, tail, f)
217     ;
218 }
219 if (snippet->dependencies && snippet
220     ->num_dependencies > 0) {
221     c_field_t* f = calloc(1, sizeof(
222         c_field_t));
223     f->name = safe_strdup("
224         dependencies");
225     f->value = calloc(1, sizeof(c_
226         field_value_t));
227     f->value->type = C_FIELD_VALUE_
228         TYPE_ARRAY_VALUE;
229     ADD_FIELD(&obj->fields, tail, f)
230     ;
231     c_field_t* arr_tail = NULL;
232     for(size_t i = 0; i < snippet->
233         num_dependencies; i++) {
234         c_field_t* item = calloc(1,
235             sizeof(c_field_t));
236         char name_buf[32];
237         snprintf(name_buf, sizeof(
238             name_buf), "%zu", i);
239         item->name = safe_strdup(
240             name_buf);
241         item->value = calloc(1,
242             sizeof(c_field_value_t))
243         ;
244         item->value->type = C_FIELD_
245             VALUE_TYPE_OBJECT_VALUE;
246         item->value->data.object_val
247             = dependency_to_c_
248             object(snippet->
249             dependencies[i]);
250         ADD_FIELD(&f->value->data.
251             array_values, arr_tail,
252             item);
253     }
254 }
255 return obj;
256 }
257
258 static AgenkCode* code_from_c_object(
259     const c_object_t* code_obj) {
260     if(!code_obj) return NULL;
261     int lang = 0; char* entry = NULL;
262     for (c_field_t* f = code_obj->fields
263         ; f; f = f->next) {
264         if (strcmp(f->name, "language")
265             == 0) lang = (int)f->value->
266             data.int_val;
267         if (strcmp(f->name, "entry_point
268 ") == 0) entry = f->value->
269             data.string_val;
270     }
271     AgenkCode* code = code_create(lang,
272         entry);
273     for (c_field_t* f = code_obj->fields
274         ; f; f = f->next) {
275         if (strcmp(f->name, "variants")
276             == 0) {
277             for(c_map_pair_t* pair = f->
278                 value->data.map_pairs;
279                 pair; pair = pair->next)
280                 {
281                     code_add_variant(code,
282                         atoi(pair->key),
283                         pair->value->data.
284                         string_val);
285                 }
286             }
287         }
288     }
289     return code;
290 }
291
292 static AgenkDependency* dependency_from_
293     c_object(const c_object_t* dep_obj)
294     {
295     if(!dep_obj) return NULL;
296     int type = 0; char* uri = NULL;
297     for(c_field_t* f = dep_obj->fields;
298         f; f = f->next) {
299         if (strcmp(f->name, "type") ==
300             0) type = (int)f->value->
301             data.int_val;
302         if (strcmp(f->name, "uri") == 0)
303             uri = f->value->data.string_
304             _val;
305     }
306     AgenkDependency* dep = dependency_
307         create(type, uri);
308     for(c_field_t* f = dep_obj->fields;
309         f; f = f->next) {
310         if (strcmp(f->name, "build_
311             commands") == 0 && f->value
312             ->type == C_FIELD_VALUE_TYPE_
313             _ARRAY_VALUE) {
314             for(c_field_t* item = f->
315                 value->data.array_values
316                 ; item; item = item->
317                 next) {
318                 dependency_add_build_
319                     command(dep, item->
320                     value->data.string_
321                     val);
322             }
323         }
324         if (strcmp(f->name, "required_
325             artifacts") == 0 && f->value
326             ->type == C_FIELD_VALUE_TYPE_
327             _ARRAY_VALUE) {
328             for(c_field_t* item = f->
329                 value->data.array_values
330                 ; item; item = item->
331                 next) {
332                 dependency_add_required_
333                     artifact(dep, item->
334                     value->data.string_
335                     val);
336             }
337         }
338     }
339     return dep;
340 }
341
342 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, "
284             description") == 0) s->
285             description = safe_strdup(
286                 field->value->data.string_
287                 val);
288         else if (strcmp(field->name, "
289             expected_test_output") == 0)
290             s->expected_test_output =
291                 safe_strdup(field->value->
292                 data.string_val);
293         else if (strcmp(field->name, "
294             execution_code") == 0) s->
295             execution_code = code_from_c_
296             _object(field->value->data.
297             object_val);
298         else if (strcmp(field->name, "
299             test_code") == 0) s->test_
300             code = code_from_c_object(
301                 field->value->data.object_
302                 val);
303         else if (strcmp(field->name, "
304             dependencies") == 0) {
305             for (c_field_t* item = field
306                 ->value->data.array_
307                 values; item; item=item
308                 ->next) {
309                 AgenkDependency* dep =
310                     dependency_from_c_
311                     object(item->value->
312                     data.object_val);
313                 snippet_add_dependency(s
314                     , dep);
315             }
316         }
317     }
318     return s;
319 }
320
321 static void get_key_for_snippet_id(const
322     char* snippet_id, unsigned char key
323     _out[OS_MAX_KEY_SIZE]) {
324     md5_state_t pms;
325     md5_byte_t digest[16];
326     md5_init(&pms);
327     md5_append(&pms, (const md5_byte_t*)
328         snippet_id, strlen(snippet_id));
329     md5_finish(&pms, digest);
330     memcpy(key_out, digest, 16);
331     memset(key_out + 16, 0, OS_MAX_KEY_
332         SIZE - 16);
333 }

```

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

6.2.4 Validation

The correctness of this substrate is validated by a dedicated test suite, `test_snippets.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 #include <stdio.h>
11 #include <stdlib.h>
12 #include <string.h>
13
14 #include "agenk_snippet.h"
15 #include "object_store.h"
16 #include "object.h"
17
18 // --- Testing Framework Macros ---
19 static int g_tests_run = 0;
20 static int g_tests_passed = 0;
21 #define COLOR_GREEN "\x1B[32m"
22 #define COLOR_RED "\x1B[31m"
23 #define COLOR_RESET "\x1B[0m"
24 #define TEST_SUITE_START(name) printf("\
25     n---_Running_Test_Suite:_%s_---\n",
26     name)
27 #define TEST_CASE(name) printf("\
28     _%s\n", name)
29 #define ASSERT(condition, message)
30
31     do {
32
33         \
34         g_tests_run++;
35
36         \
37         if (condition) {
38
39             \
40             g_tests_passed++;
41
42             \
43             printf(COLOR_GREEN "\
44                 _%s\n" COLOR_RESET,
45                 message);
46         } else {
47
48             \
49             fprintf(stderr, COLOR_RED "\
50                 FAIL]_%s:%d:_%s_(_%s)\n"
51                 \

```

```

33         COLOR_RESET, __FILE_
34         , __LINE_,
        message, #
        condition);
        \
    }

    } while (0)

// --- Forward Declarations for Test
Helpers ---
AgenkSnippet* create_complex_snippet();
bool compare_snippets(const AgenkSnippet
* s1, const AgenkSnippet* s2);

// --- Test Suites ---
void test_suite_registry_lifecycle() {
    TEST_SUITE_START("SnippetRegistry(
FullRound-Trip)");
    object_store_t* store = NULL;
    ASSERT(object_store_create(&store)
== 0, "ObjectStoreCreated");
    ASSERT(object_store_init_db(store, "
./test_snippet_db", 0, 0) == FB_
SERIALIZER_OK, "Database_
initialized");
    ASSERT(snippet_registry_init(store),
"SnippetRegistryinitialized")
;
    TEST_CASE("Creating_a_complex_
snippet('snippet_out')");
    AgenkSnippet* snippet_out = create_
complex_snippet();
    ASSERT(snippet_out != NULL, "snippet
_out_created_successfully");
    TEST_CASE("Adding_snippet_out_to_the
registry");
    ASSERT(snippet_registry_add(snippet_
out), "snippet_registry_add_
succeeded");
    TEST_CASE("Retrieving_snippet_by_ID_
('snippet_in')");
    AgenkSnippet* snippet_in = snippet_
registry_get_by_id("tools.text.
formatter.v1");
    ASSERT(snippet_in != NULL, "snippet_
registry_get_by_id_succeeded");
    TEST_CASE("Performing_deep_
comparison_of_snippet_out_and_
snippet_in");
    ASSERT(compare_snippets(snippet_out,
snippet_in), "Snippets_are_
identical_after_round-trip");
    TEST_CASE("Deleting_the_snippet_from
the_registry");
    ASSERT(snippet_registry_delete("
tools.text.formatter.v1"), "
snippet_registry_delete_
succeeded");
    TEST_CASE("Verifying_snippet_is_no_
longer_in_the_registry");
    AgenkSnippet* deleted_snippet =
snippet_registry_get_by_id("
tools.text.formatter.v1");
    ASSERT(deleted_snippet == NULL, "
Verified_snippet_is_deleted");
    snippet_free(snippet_out);
    snippet_free(snippet_in);
    snippet_registry_shutdown();

    object_store_destroy(&store);
    system("rm -rf ./test_snippet_db");
}

int main() {
    printf("
=====
n");
    printf("Running AGENK Snippets_
SubstrateTests\n");
    printf("
=====
n");
    CLASS_INIT();
    test_suite_registry_lifecycle();
    CLASS_SHUTDOWN();
    printf("\n=====
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)
;
        return 1;
    }
}

// --- Helper Implementations ---

AgenkSnippet* create_complex_snippet() {
    AgenkSnippet* s = snippet_create("
Formats text to a specified_
width.");

    // THE DEFINITIVE FIX: Do not use
the non-standard strdup().
// It causes a compiler warning and
results in an invalid pointer,
// leading to a segfault.
// Allocate and copy the string
manually using standard C
functions.
    const char* id_str = "tools.text.
formatter.v1";
    s->snippet_id = (char*)malloc(strlen
(id_str) + 1);
    if (s->snippet_id) {
        strcpy(s->snippet_id, id_str);
    }

    AgenkCode* code = code_create(LANG_C
, "format_text");
    code_add_variant(code, OS_ANY, "//_C
source_code_would_be_here.");
    code_add_variant(code, OS_LINUX, "//
Linux-specific_C_source.");
    snippet_set_execution_code(s, code);

    AgenkDependency* dep1 = dependency_
create(DEP_SYSTEM, "libc");
    snippet_add_dependency(s, dep1);
}

```

```

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         }
143         if(!found_match) return false;
144     }
145     return true;
146 }
147
148 bool compare_snippets(const AgenkSnippet
149 * s1, const AgenkSnippet* s2) {
150     if (!s1 || !s2) return false;
151     if (strcmp(s1->snippet_id, s2->
152               snippet_id) != 0) return false;
153     if (strcmp(s1->description, s2->
154               description) != 0) return false;
155     if (!compare_code(s1->execution_code
156                       , s2->execution_code)) return
157         false;
158     if (s1->num_dependencies != s2->num_
159         dependencies) return false;
160     for(size_t i = 0; i < s1->num_
161         dependencies; i++) {
162         const AgenkDependency *d1 = s1->
163             dependencies[i];
164         const AgenkDependency *d2 = s2->
165             dependencies[i];
166         if (d1->type != d2->type) return
167             false;
168         if (strcmp(d1->uri, d2->uri) !=
169             0) return false;
170     }
171     return true;
172 }

```

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 lifecycle 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

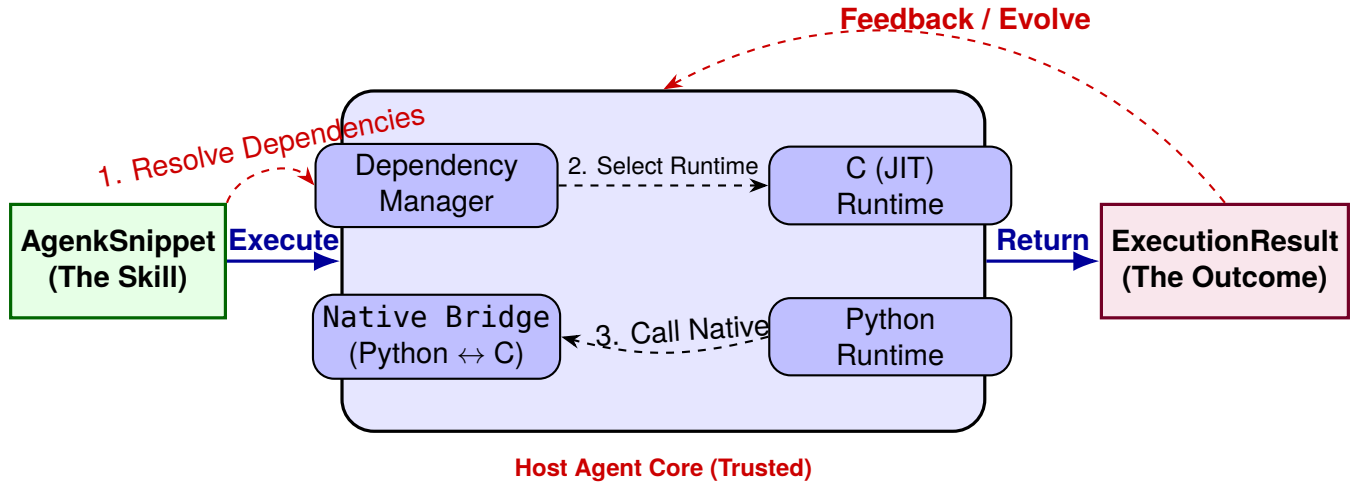


Figure 2: High-level architectural flow of the Execution Substrate. An immutable `AgenkSnippet` is passed to the core `engine` , which orchestrates dependency resolution and delegates execution to a sandboxed language runtime. The `Native Bridge` provides a secure interface for guest code to call back 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 ${
19   CMAKE_BINARY_DIR}/lib)
20 set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${
21   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: ${
27   CMAKE_BUILD_TYPE}")
28
29 option(AGENK_ENABLE_PYTHON "Enable_
30   Python_runtime_and_native_bridge" ON
31 )
32
33 option(AGENK_ENABLE_NODEJS "Enable_
34   NodeJS/NPM_runtime" OFF) #
35   Placeholder for the future
36
37 # =====
38 # SECTION 2: VENDORED &
39 # SYSTEM DEPENDENCY MANAGEMENT
40 # =====
41 # --- System Libraries ---
42 find_package(Threads REQUIRED)
43 find_library(MATH_LIBRARY m)
44
45 # --- Dependency Discovery ---
46 find_package(Threads REQUIRED)
47 if(AGENK_ENABLE_PYTHON)
48   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 # --- mbed TLS (vendored from deps/) ---
50 set(ENABLE_TESTING OFF CACHE BOOL "
51     Disable_mbedtls_tests")
52 set(ENABLE_PROGRAMS OFF CACHE BOOL "
53     Disable_mbedtls_programs")
54 add_subdirectory(deps/mbedtls)
55 set(MBEDTLS_LIBRARIES mbedtls mbedx509
56     mbedcrypto)
57 set(MBEDTLS_LIBRARY_DIR ${CMAKE_ARCHIVE_
58     OUTPUT_DIRECTORY})
59 set(MBEDTLS_INCLUDE_DIR ${CMAKE_CURRENT_
60     SOURCE_DIR}/deps/mbedtls/include)
61 # --- libcurl (vendored from deps/,
62     built with custom command) ---
63 ExternalProject_Add(
64     curl_local_build
65     SOURCE_DIR ${CMAKE_CURRENT_SOURCE_
66     DIR}/deps/curl
67     BINARY_DIR ${CMAKE_BINARY_DIR}/curl-
68     build
69     # Use bash -c to create a reliable
70     shell environment for the
71     configure script
72     CONFIGURE_COMMAND bash -c
73     "LDFLAGS='-L${MBEDTLS_LIBRARY_
74     DIR}'_LIBS='-lmbedtls_-
75     lmbedx509_-lmbedcrypto_-
76     lpthread'_CPPFLAGS='-I${
77     MBEDTLS_INCLUDE_DIR}'_<
78     SOURCE_DIR>/configure--
79     disable-shared--with-ssl--
80     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     DIR}/curl-build/lib/.libs/
90     libcurl.a"
91 )
92 add_dependencies(libcurl curl_local_
93     build)
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)

```

```

83     add_library(cjson STATIC deps/cJSON/
84     cJSON.c)
85     target_include_directories(cjson
86     PUBLIC ${CMAKE_CURRENT_SOURCE_
87     DIR}/deps/cJSON)
88     set(CJSON_LIB cjson)
89     set(CJSON_FOUND TRUE)
90 endif()
91 ExternalProject_Add(tcc_external_project
92     SOURCE_DIR ${CMAKE_CURRENT_SOURCE_
93     DIR}/deps/tcc CONFIGURE_COMMAND ""
94     BUILD_COMMAND "" INSTALL_COMMAND "")
95 add_custom_target(build_tcc_library ALL
96     COMMAND ${CMAKE_COMMAND} -E chdir ${
97     CMAKE_CURRENT_SOURCE_DIR}/deps/tcc
98     ./configure --libdir=${CMAKE_CURRENT_
99     SOURCE_DIR}/deps/tcc/lib --extra-
100     cflags="-fPIC" COMMAND ${CMAKE_
101     COMMAND} -P ${CMAKE_CURRENT_BINARY_
102     DIR}/cmake_script_patch_tcc_makefile.
103     cmake COMMAND ${CMAKE_COMMAND} -E
104     chdir ${CMAKE_CURRENT_SOURCE_DIR}/
105     deps/tcc make libtcc.a COMMAND ${
106     CMAKE_COMMAND} -E chdir ${CMAKE_
107     CURRENT_SOURCE_DIR}/deps/tcc/lib
108     make COMMAND ${CMAKE_COMMAND} -E
109     copy_if_different ${CMAKE_CURRENT_
110     SOURCE_DIR}/deps/tcc/libtcc.a ${
111     CMAKE_BINARY_DIR}/lib/ COMMAND ${
112     CMAKE_COMMAND} -E copy_if_different
113     ${CMAKE_CURRENT_SOURCE_DIR}/deps/tcc
114     /libtcc1.a ${CMAKE_BINARY_DIR}/lib/
115     DEPENDS tcc_external_project)
116 file(WRITE ${CMAKE_CURRENT_BINARY_DIR}/
117     cmake_script_patch_tcc_makefile.
118     cmake "set(TCC_LIB_MAKEFILE_PATH\"$
119     {CMAKE_CURRENT_SOURCE_DIR}/deps/tcc/
120     lib/Makefile\")\nfile(READ\"${TCC_
121     LIB_MAKEFILE_PATH}\"TCC_LIB_MAKEFILE_
122     CONTENT)\nstring(REPLACE\"bcheck.o
123     \"\"\"#bcheck.o\"\"TCC_LIB_MAKEFILE_
124     PATCHED\"\"${TCC_LIB_MAKEFILE_
125     CONTENT}\")\nfile(WRITE\"${TCC_LIB_
126     MAKEFILE_PATH}\"\"${TCC_LIB_MAKEFILE_
127     PATCHED}\")\nmessage(STATUS\"
128     Patched_TCC_lib/Makefile_to_disable_
129     bcheck.\")\n")
130 add_library(libtcc STATIC IMPORTED
131     GLOBAL)
132 set_target_properties(libtcc PROPERTIES
133     IMPORTED_LOCATION "${CMAKE_BINARY_
134     DIR}/lib/libtcc.a")
135 add_dependencies(libtcc build_tcc_
136     library)
137 # =====
138 # SECTION 3: AUTOMATED CODE
139 # GENERATION (from FlatBuffers)
140 # =====
141 set(FBS_SCHEMA ${CMAKE_CURRENT_SOURCE_
142     DIR}/src/object_store/core/object.
143     fbs)
144 set(GENERATED_INCLUDE_DIR ${CMAKE_
145     CURRENT_BINARY_DIR}/generated/object_
146     store)
147 file(MAKE_DIRECTORY ${GENERATED_INCLUDE_
148     DIR})
149 set(GENERATED_FILES
150     ${GENERATED_INCLUDE_DIR}/object_
151     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
126 # --- LIBRARY 1: object store (The
127     DataBase) ---
128 # --- NEW LIBRARY: object_store (The
129     Persistence Layer) ---
130 add_library(object_store STATIC
131     src/object_store/core/fb_serializer.
132     c
133     src/object_store/core/object_store.c
134 )
135 # This library depends on the auto-
136     generated FlatBuffers headers
137 add_dependencies(object_store generate_
138     object_headers)
139
140 # Tell the compiler where to find its
141     headers
142 target_include_directories(object_store
143     PUBLIC
144     ${CMAKE_CURRENT_SOURCE_DIR}/src/
145     object_store
146     ${GENERATED_INCLUDE_DIR}
147     ${CMAKE_CURRENT_SOURCE_DIR}/deps/
148     flatcc/include
149 )
150
151 # Tell the linker what other libraries
152     it needs
153 target_link_libraries(object_store
154     PRIVATE
155     flatccrt # The FlatBuffers
156         runtime library
157     lmbd
158     md5
159     Threads::Threads
160     ${MATH_LIBRARY} # ADD THIS LINE
161 )
162
163 # Handle the optional dependency on
164     cJSON
165 if(CJSON_FOUND)
166     target_link_libraries(object_store
167         PRIVATE ${CJSON_LIB})
168     target_compile_definitions(object_
169         store PRIVATE CJSON_ENABLED)
170 endif()
171
172 # --- LIBRARY 2: agenk_tensor (The Data
173     Canvas) ---
174 add_library(agenk_tensor STATIC
175     src/tensors/tensor_lifecycle.c
176     src/tensors/tensor_ops.c
177     src/tensors/tensor_views.c
178     src/tensors/tensor_materialize.c
179 )
180 target_include_directories(agenk_tensor
181     PUBLIC
182     ${CMAKE_CURRENT_SOURCE_DIR}/src/
183     tensors/include
184 )
185
186 # --- LIBRARY 3: agenk_snippet (The Data
187     Structures & Registry) ---
188 add_library(agenk_snippet STATIC
189     src/snippets/snippet.c
190     src/snippets/snippet_registry.c #
191     <-- ADD NEW SOURCE FILE
192 )
193 target_include_directories(agenk_snippet
194     PUBLIC
195     ${CMAKE_CURRENT_SOURCE_DIR}/src/
196     snippets/include
197 )
198 # Snippets now need the ObjectStore for
199     persistence and Tensors for data
200     types.
201 target_link_libraries(agenk_snippet
202     PRIVATE
203     agenk_tensor
204     object_store
205     md5 # Dependency of the registry for
206         key generation
207 )
208
209 # --- LIBRARY 4: class_framework (The
210     Live Object Runtime) ---
211 add_library(class_framework STATIC
212     src/objects/core/object_impl.c
213     src/objects/core/object_registry.c
214     src/objects/core/object_db.c
215 )
216 target_include_directories(class_
217     framework PUBLIC ${CMAKE_CURRENT_
218     SOURCE_DIR}/src/objects)
219 target_link_libraries(class_framework
220     PRIVATE lmbd Threads::Threads)
221
222 # --- LIBRARY 5: agenk_core (The "Kernel
223     " / Stable Native API) ---
224 set(CORE_SOURCES src/core/core.c)
225 if(AGENK_ENABLE_PYTHON)
226     list(APPEND CORE_SOURCES src/core/
227         bridge/native_bridge_python.c)
228 endif()
229 add_library(agenk_core STATIC ${CORE_
230     SOURCES})
231
232 target_include_directories(agenk_core
233     PUBLIC
234     ${CMAKE_CURRENT_SOURCE_DIR}/src/core
235     /include
236 )
237 if(AGENK_ENABLE_PYTHON)
238     target_include_directories(agenk_
239         core PUBLIC
240         ${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
230 target_link_libraries(agenk_sensation
231     PRIVATE
232     agenk_core
233     md5
234     libtcc
235     object_store # This must be here
236 )
237 # --- LIBRARY 6: agenk_execution (The "
238 # CPU" / Execution Engine) ---
239 set(EXECUTION_SOURCES
240     src/execution/core/engine.c
241     src/execution/builder/dependency_
242     manager.c
243     src/execution/runtimes/runtime_c.c
244     src/execution/runtimes/runtime_shell
245     .c
246 )
247 if(AGENK_ENABLE_PYTHON)
248     list(APPEND EXECUTION_SOURCES src/
249         execution/runtimes/runtime_
250         python.c)
251 endif()
252 add_library(agenk_execution STATIC ${
253     EXECUTION_SOURCES})
254
255 target_include_directories(agenk_
256     execution PUBLIC
257     ${CMAKE_CURRENT_SOURCE_DIR}/src/
258     execution/include
259     PRIVATE
260     ${CMAKE_CURRENT_SOURCE_DIR}/src/
261     execution
262 )
263 target_compile_definitions(agenk_
264     execution PRIVATE
265     AGENK_SOURCE_DIR="${CMAKE_SOURCE_DIR}"
266     AGENK_BINARY_DIR="${CMAKE_BINARY_DIR}"

```

```

251     }"
252     AGENK_TCC_PATH="${CMAKE_CURRENT_
253     SOURCE_DIR}/deps/tcc"
254     # THE FIX: Provide a reliable path
255     to the *built* TCC libraries
256     AGENK_TCC_LIB_PATH="${CMAKE_ARCHIVE_
257     OUTPUT_DIRECTORY}"
258     # Provide include directories for
259     runtime compilation
260     AGENK_CORE_INCLUDE_DIR="${CMAKE_
261     CURRENT_SOURCE_DIR}/src/core/
262     include"
263     AGENK_SNIPPET_INCLUDE_DIR="${CMAKE_
264     CURRENT_SOURCE_DIR}/src/snippets
265     /include"
266 )
267 if(AGENK_ENABLE_PYTHON)
268     # Add compile definitions needed for
269     the Python runtime.
270     # - AGENK_PYTHON_SUPPORT enables all
271     Python-related code blocks.
272     # - PYTHON_SITE_PACKAGES tells the C
273     code where to find system
274     packages.
275     target_compile_definitions(agenk_
276     execution PRIVATE
277     AGENK_PYTHON_SUPPORT
278     "PYTHON_SITE_PACKAGES=\"${
279     Python3_SITELIB}\""
280 )
281 endif()
282 target_link_libraries(agenk_execution
283     PRIVATE
284     agenk_core agenk_snippet libtcc
285     Threads::Threads md5
286 )
287 if(AGENK_ENABLE_PYTHON)
288     target_link_libraries(agenk_
289     execution PRIVATE Python3::
290     Python)
291 endif()
292
293 # =====
294 # SECTION 5: EXECUTABLE
295 # TARGETS (TESTS & EXAMPLES)
296 # =====
297
298 # --- EXECUTABLES ---
299 add_executable(test_tensors src/tensors/
300     test/test_tensors.c)
301 target_link_libraries(test_tensors
302     PRIVATE agenk_tensor)
303
304 add_executable(test_sensation src/senses
305     /test/test_sensation.c)
306 target_link_libraries(test_sensation
307     PRIVATE agenk_sensation)
308
309 add_executable(test_sensation_build src/
310     senses/test/test_sensation_build.c)
311 target_link_libraries(test_sensation_
312     build PRIVATE agenk_sensation)
313
314 add_executable(test_sense_registry src/
315     senses/test/test_sense_registry.c)
316 target_link_libraries(test_sense_
317     registry PRIVATE agenk_sensation ${

```

<pre> 294 MATH_LIBRARY}) 295 add_executable(test_concurrency src/ 296 objects/test/test_concurrency.c) 297 target_link_libraries(test_concurrency 298 PRIVATE class_framework Threads:: 299 Threads) 300 301 add_executable(test_errors src/objects/ 302 test/test_errors.c) 303 target_link_libraries(test_errors 304 PRIVATE class_framework) 305 306 add_executable(test_hierarchy src/ 307 objects/test/test_hierarchy.c) 308 target_link_libraries(test_hierarchy 309 PRIVATE class_framework) 310 311 add_executable(test_stress src/objects/ 312 test/test_stress.c) 313 target_link_libraries(test_stress 314 PRIVATE class_framework) 315 316 add_executable(example_basic_usage src/ 317 objects/examples/basic_usage.c) 318 target_link_libraries(example_basic_ 319 usage PRIVATE class_framework) 320 321 add_executable(example_lifecycle src/ 322 objects/examples/lifecycle_and_ 323 groups.c) 324 target_link_libraries(example_lifecycle 325 PRIVATE class_framework) 326 327 add_executable(test_object_store src/ 328 object_store/test/test_object_store. 329 c) 330 target_link_libraries(test_object_store 331 PRIVATE object_store Threads:: 332 Threads \${CJSON_LIB}) 333 334 # --- NEW EXECUTABLE for Snippet testing 335 --- 336 add_executable(test_snippets src/ 337 snippets/test/test_snippets.c) 338 target_link_libraries(test_snippets 339 PRIVATE 340 agenk_snippet 341 class_framework # For CLASS_INIT 342 object_store # For the registry's 343 backend 344) 345 346 add_executable(test_execution src/ 347 execution/test/test_execution.c) 348 349 if(AGENK_ENABLE_PYTHON) 350 target_compile_definitions(test_ 351 execution PRIVATE AGENK_PYTHON_ 352 SUPPORT) # <-- ADD THIS LINE 353 BACK 354 # THE DEFINITIVE FIX for systems 355 using GNU ld or clang ld. 356 # This consolidates all linking into 357 a single, ordered command. 358 if(CMAKE_SYSTEM_NAME STREQUAL "Linux" 359 " OR CMAKE_SYSTEM_NAME STREQUAL 360 "Darwin") 361 target_link_libraries(test_ 362 execution PRIVATE </pre>	<pre> 332 # --- Libraries used directly by 333 the test C code --- 334 agenk_execution 335 class_framework 336 337 # --- Force-load the python 338 bridge, THEN provide its 339 dependencies --- 340 "-Wl,--whole-archive" 341 agenk_core 342 "-Wl,--no-whole-archive" 343 agenk_snippet 344 agenk_tensor 345 346 # --- Provide all other system/ 347 vendored dependencies --- 348 lmbd 349 Threads::Threads 350 Python3::Python 351 Python3::NumPy 352) 353 else() 354 # Fallback for other systems (e.g 355 ., Windows). 356 target_link_libraries(test_ 357 execution PRIVATE 358 agenk_execution class_ 359 framework agenk_core agenk 360 snippet 361 agenk_tensor lmbd Threads:: 362 Threads Python3::Python 363) 364 endif() 365 else() 366 # Link normally if Python is 367 disabled 368 target_link_libraries(test_execution 369 PRIVATE 370 agenk_execution class_framework 371 agenk_core agenk_snippet 372 agenk_tensor lmbd Threads:: 373 Threads 374) 375 endif() 376 377 if(CMAKE_SYSTEM_NAME STREQUAL "Linux") 378 target_link_options(test_execution 379 PRIVATE "-Wl,--export-dynamic") 380 endif() 381 382 # ===== 383 # SECTION 6: CTEST 384 # INTEGRATION 385 # ===== 386 enable_testing() 387 388 # --- TEST --- 389 add_test(NAME Snippets. 390 LifecycleAndRegistry COMMAND test_ 391 snippets) 392 add_test(NAME Execution.Substrate 393 COMMAND test_execution) 394 add_test(NAME Tensor.Substrate 395 COMMAND test_tensors) 396 add_test(NAME Sensation.Subsystem.Core 397 COMMAND test_sensation) 398 add_test(NAME Sensation.Subsystem.Build 399 COMMAND test_sensation_build) 400 add_test(NAME Sensation.Registry COMMAND </pre>
---	---

```

382     test_sense_registry)
add_test(NAME ObjectStore.Core
COMMAND test_object_store)
383 add_test(NAME Framework.Concurrency
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 compilation 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 #ifndef AGENK_EXECUTION_H
17 #define AGENK_EXECUTION_H
18
19 // THIS IS THE FIX:
20 // Include the single source of truth
21 // for data structures.
22 // All the struct definitions that were
23 // previously here have been REMOVED.
24 #include "agenk_snippet.h"
25
26 // --- Substrate Lifecycle ---
27 bool engine_init(const char* cache_path);
28 void engine_shutdown();
29
30 // --- Core Execution Function ---
31 ExecutionResult* engine_execute(const
32     AgenkSnippet* snippet, const
33     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: %s\n", \
41         FILE_, \
42         LINE_, ##_VA_ARGS_)
43 #define EXEC_LOG_ERROR(format, ...) \
44     fprintf(stderr, "[EXEC_ERROR] %s: \
45         %s\n", \
46         FILE_, \
47         LINE_, ##_VA_ARGS_)
48 #define EXEC_ASSERT(condition) \
49     assert(condition)
50 #else

```



```

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(
    const char* error_message);
47
48 // From builder/dependency_manager.c
49 bool resolve_dependencies(const
    AgenkSnippet* snippet);
50 // NEW: Internal handlers for different
    dependency types
51 bool handle_source_dependency(const
    AgenkDependency* dep);
52 bool handle_pip_dependency(const
    AgenkDependency* dep);
53 bool handle_npm_dependency(const
    AgenkDependency* dep);
54 // THE FIX: Added prototype for run_
    command, making it visible to
    runtime_shell.c
55 int run_command(const char* command,
    const char* working_dir, char** out_
    stdout, char** out_stderr);
56
57 // From runtimes/
58 ExecutionResult* execute_c_snippet(const
    AgenkCode* code, const
    ExecutionContext* context);
59 bool python_runtime_init();
60 void python_runtime_shutdown();
61 ExecutionResult* execute_python_snippet(
    const AgenkCode* code, const
    ExecutionContext* context);
62 ExecutionResult* execute_shell_snippet(
    const char* command_string, const
    ExecutionContext* context);
63
64 // From bridge/native_bridge_python.c
65 // THE FIX: Added prototype for the
    native module init function.
66 PyObject* PyInit_agenk_core(void);
67 // THE FIX: Added prototype for the
    native module init function.
68 // PyObject* PyInit_agenk_core(void);
69
70 // --- Utility Functions ---
71 char* safe_strdup(const char* s);
72 bool ensure_dir_exists(const char* path)
    ;
73 const char* get_os_string();
74
75 #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.

```

1 // 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,
    cross-platform dependency
    resolution and
4  *      build system for the AGENK
    Execution Substrate.
5
6  * This module is a core component of
    the Execution Substrate,
    responsible for
7  * giving the agent the ability to
    manage its own software
    dependencies. It can
8  * check for required tools, fetch
    source code from Git, execute build
    commands,
9  * and manage a local cache of artifacts
    in a way that is portable across
10 * Windows and POSIX-compliant systems (
    Linux, macOS).
11
12 * The design prioritizes robustness,
    portability, and security:

```

```

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

```

```

112     all_from_fd(stderr_pipefd[0]);
113     close(stdout_pipefd[0]); // Close
        read end
114     close(stderr_pipefd[0]); // Close
        read end
115     int status;
116     waitpid(pid, &status, 0);
117     return WIFEXITED(status) ?
        WEXITSTATUS(status) : -1;
118 #endif
119 }
120 //=====
121 // SECTION 3: DEPENDENCY MANAGEMENT
122 // LOGIC (Now Cross-Platform)
123 //=====
124 /**
125  * @brief Generates a unique,
126  * deterministic 32-character hex
127  * string hash for a dependency.
128  */
129 static void get_dep_unique_hash(const
130 AgenkDependency* dep, char hash_out
131 [33]) {
132     md5_state_t pms;
133     md5_byte_t digest[16]; // FIX: Was a
        single byte, needs to be an
        array of 16
134     md5_init(&pms);
135     md5_append(&pms, (const md5_byte_t*)
        dep->uri, strlen(dep->uri));
136     md5_finish(&pms, digest);
137     for (int i = 0; i < 16; ++i) {
138         sprintf(&hash_out[i * 2], "%02x"
        , digest[i]);
139     }
140     hash_out[32] = '\0'; // FIX: Null-
        terminate the string
141 }
142 /**
143  * @brief Checks if all required
144  * artifacts for a dependency already
145  * exist in the cache.
146  */
147 static bool check_artifacts_exist(const
148 AgenkDependency* dep) {
149     char path_buffer[MAX_PATH_LEN]; //
        FIX: Was a single char, now a
        buffer
150     for (size_t i = 0; i < dep->num_
        required_artifacts; ++i) {
151         snprintf(path_buffer, sizeof(
        path_buffer), "%s/%s", g_
        engine_state.cache_path, dep
        ->required_artifacts[i]);
152         struct stat st;
153         if (stat(path_buffer, &st) != 0)
154             return false;
155     }
156     return dep->num_required_artifacts >
        0;
157 }
158 /**
159  * @brief Copies the required build
160  * artifacts from a temp build
161  * directory to the agent's cache.
162  */
163
164 static bool install_artifacts(const
165 AgenkDependency* dep, const char*
166 source_dir) {
167     char command[MAX_CMD_LEN]; // FIX:
        Was a single char, now a buffer
168     int rc;
169     for (size_t i = 0; i < dep->num_
        required_artifacts; ++i) {
170         #ifdef WIN32
171             snprintf(command, sizeof(command)
        , "xcopy %E\\I\\Y\\%s\\%s
        \\ %s\\%s\\", source_dir,
        dep->required_artifacts[i],
        g_engine_state.cache_path);
172         #else
173             snprintf(command, sizeof(command)
        , "cp -r %s/%s %s/%s",
        source_dir, dep->required_
        artifacts[i], g_engine_state
        .cache_path);
174         #endif
175         rc = run_command(command, ".",
        NULL, NULL);
176         if (rc != 0) { EXEC_LOG_ERROR("
        Failed to install artifact:
        %s", dep->required_artifacts
        [i]); return false; }
177     }
178     return true;
179 }
180 /**
181  * @brief Handles dependencies that must
182  * be built from source (e.g., from
183  * Git).
184  */
185 bool handle_source_dependency(const
186 AgenkDependency* dep) {
187     char dep_hash[33]; // FIX: Sized for
        32 hex chars + null terminator
188     get_dep_unique_hash(dep, dep_hash);
189     char build_dir[MAX_PATH_LEN]; // FIX
        : Was a single char, now a
        buffer
190     snprintf(build_dir, sizeof(build_dir)
        , "%s/builds/%s", g_engine_
        state.cache_path, dep_hash);
191     ensure_dir_exists(build_dir);
192
193     char command[MAX_CMD_LEN]; // FIX:
        Was a single char, now a buffer
194     snprintf(command, sizeof(command), "
        git clone --depth 1 %s.", dep->
        uri);
195     if (run_command(command, build_dir,
        NULL, NULL) != 0) { return false
        ; }
196
197     for (size_t i = 0; i < dep->num_
        build_commands; ++i) {
198         if (run_command(dep->build_
        commands[i], build_dir, NULL
        , NULL) != 0) { return false
        ; }
199     }
200     if (!install_artifacts(dep, build_
        dir)) return false;
201     return true;
202 }

```

```

192 /**
193  * @brief Handles Python package
      dependencies using PIP.
194  */
195 bool handle_pip_dependency(const
      AgenkDependency* dep) {
196     char python_cache_dir[MAX_PATH_LEN];
      // FIX: Was a single char, now
      a buffer
197     snprintf(python_cache_dir, sizeof(
      python_cache_dir), "%s/python_
      packages", g_engine_state.cache_
      path);
198     ensure_dir_exists(python_cache_dir);
199     char command[MAX_CMD_LEN]; // FIX:
      Was a single char, now a buffer
200     snprintf(command, sizeof(command), "
      python3 -m pip install %s --
      target \"%s\"", dep->uri, python
      _cache_dir);
201     if (run_command(command, ".", NULL,
      NULL) != 0) { return false; }
202     return true;
203 }
204
205 /**
206  * @brief Handles NodeJS package
      dependencies using NPM.
207  */
208 bool handle_npm_dependency(const
      AgenkDependency* dep) {
209     char node_cache_dir[MAX_PATH_LEN];
      // FIX: Was a single char, now a
      buffer
210     snprintf(node_cache_dir, sizeof(node
      _cache_dir), "%s/node_modules_
      cache", g_engine_state.cache_
      path);
211     ensure_dir_exists(node_cache_dir);
212     char command[MAX_CMD_LEN]; // FIX:
      Was a single char, now a buffer
213     snprintf(command, sizeof(command), "
      npm install %s --prefix \"%s\"",
      dep->uri, node_cache_dir);
214     if (run_command(command, ".", NULL,
      NULL) != 0) { return false; }
215     return true;
216 }
217
218 /**
219  * @brief Handles system-level
      dependencies (currently a no-op).
220  */
221 bool handle_system_dependency(const
      AgenkDependency* dep) {
222     EXEC_LOG("Assuming system dependency
      %s is met.", dep->uri);
223     return true;
224 }
225
226 //=====
227 // SECTION 4: MAIN PUBLIC
228 // API FUNCTION
229 //=====
230
231 /**
232  * @brief Resolves all dependencies
      listed in a Snippet.
233  */
234 bool resolve_dependencies(const

```

```

235 AgenkSnippet* snippet) {
236     if (!snippet) return true;
      for (size_t i = 0; i < snippet->num_
      dependencies; ++i) {
237         AgenkDependency* dep = snippet->
      dependencies[i];
238         if (check_artifacts_exist(dep))
      continue;
239         bool success = false;
240         switch (dep->type) {
241             case DEP_SOURCE: success =
      handle_source_dependency(
      dep); break;
242             case DEP_PIP: success =
      handle_pip_dependency(
      dep); break;
243             case DEP_NPM: success =
      handle_npm_dependency(
      dep); break;
244             case DEP_SYSTEM: success =
      handle_system_dependency(
      dep); break;
245             default: EXEC_LOG_ERROR("
      Unsupported dependency
      type: %d", dep->type);
246         }
247         if (!success) { EXEC_LOG_ERROR("
      Failed to resolve dependency
      for '%s'", dep->uri);
      return false; }
248     }
249     return true;
250 }

```

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  * The Snippet is responsible for
27  * allocating and returning a valid
28  * ExecutionResult struct.
29  * This runtime safely calls that
30  * function and takes ownership of the
31  * returned struct.
32  *
33  * @version 1.0.0
34  * @author Ankush Yadav, Ankit Yadav,
35  *         AuctaSapience
36  */
37
38 #include <stdio.h>
39 #include <stdlib.h>
40 #include <libtcc.h>
41 #include "../private/execution_private.h"
42
43 #include "agenk_core.h"
44 #include "agenk_snippet.h"
45
46 // Define AGENK_TCC_PATH via CMake
47 #ifndef AGENK_TCC_PATH
48 #error "AGENK_TCC_PATH must be defined
49       by CMake"
50 #endif
51
52 // Function pointer type for the snippet
53 entry point
54 typedef ExecutionResult* (*process_func)
55   (const ExecutionContext*);
56
57 ExecutionResult* execute_c_snippet(const
58   AgenkCode* code, const
59   ExecutionContext* context) {
60   ExecutionResult* result = (
61     ExecutionResult*)calloc(1,
62     sizeof(ExecutionResult));
63   TCCState *s = NULL;

```

```

40 int ret = -1;
41 process_func process = NULL;
42 const char* source = code->variants
43   ->source_code;
44
45 s = tcc_new();
46 if (!s) {
47   EXEC_LOG_ERROR("Could not create
48     TCC state.");
49   result->exit_code = -1;
50   return result;
51 }
52
53 // THE FIX: Point TCC to the build
54 output directory for its
55 libraries
56 tcc_set_lib_path(s, AGENK_TCC_LIB_
57   PATH);
58
59 // Add include paths for project
60 headers
61 tcc_add_sysinclude_path(s, AGENK_TCC
62   _PATH "/include");
63 tcc_add_include_path(s, AGENK_CORE_
64   INCLUDE_DIR);
65 tcc_add_include_path(s, AGENK_
66   SNIPPET_INCLUDE_DIR);
67
68 tcc_set_output_type(s, TCC_OUTPUT_
69   MEMORY);
70
71 // This isn't strictly necessary
72 with the linker flag, but is
73 good practice
74 tcc_add_symbol(s, "core_create_group
75   ", (const void*)core_create_
76   group);
77
78 // Compile the source code
79 ret = tcc_compile_string(s, source);
80 if (ret < 0) {
81   EXEC_LOG_ERROR("TCC compilation
82     failed for C snippet.");
83   result->exit_code = -1;
84   tcc_delete(s);
85   return result;
86 }
87
88 // Relocate symbols and retrieve
89 entry point
90 ret = tcc_relocate(s, TCC_RELOCATE_
91   AUTO);
92 if (ret < 0) {
93   EXEC_LOG_ERROR("TCC relocation
94     failed.");
95   result->exit_code = -1;
96   tcc_delete(s);
97   return result;
98 }
99
100 process = (process_func)tcc_get_
101   symbol(s, code->entry_point);
102
103 if (!process) {
104   EXEC_LOG_ERROR("TCC entry point
105     '%s' not found.", code->
106     entry_point);
107   result->exit_code = -1;
108 } else {
109   // Execute the function

```



```

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

```

Listing 40: The final, robust C JIT runtime using TCC ('runtime_c.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  * Aucta Sapience
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
33         'agenk_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 (path_str) {
47         if (PyList_Append(sys_path,
48         path_str) < 0) {
49             PyErr_Print();
50             EXEC_LOG_ERROR("Failed
51             to append site-
52             packages path to sys
53             .path.");
54         } else {
55             EXEC_LOG("Appended '%s'
56             to Python sys.path",
57             PYTHON_SITE_
58             PACKAGES);
59         }
60     }
61     Py_DECREF(path_str);
62 }
63
64 #endif
65
66 return true;
67 }
68
69 void python_runtime_shutdown() {
70     if (Py_IsInitialized()) Py_
71     FinalizeEx();
72 }
73
74 ExecutionResult* execute_python_snippet(
75     const AgenkCode* code, const
76     ExecutionContext* context) {
77     ExecutionResult* result = calloc(1,
78     sizeof(ExecutionResult));
79     if (!result) return NULL;
80
81     PyObject *old_stdout = NULL, *old
82     stderr = NULL, *io_module = NULL
83     ;
84     PyObject *py_stdout = NULL, *py_
85     stderr = NULL, *local_dict =
86     NULL;
87     PyObject *exec_result = NULL, *func
88     = NULL, *py_context = NULL;
89     PyObject *pArgs = NULL, *call_result
90     = NULL;
91
92     old_stdout = PySys_GetObject("stdout
93     ");
94     old_stderr = PySys_GetObject("stderr
95     ");
96
97     io_module = PyImport_ImportModule("
98     io");
99     if (!io_module) { result->exit_code
100     = -1; goto cleanup; }
101     py_stdout = PyObject_CallMethod(io_
102     module, "StringIO", NULL);
103     py_stderr = PyObject_CallMethod(io_
104     module, "StringIO", NULL);

```

```

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

```

Listing 41: The sandboxed Python runtime with I/O and exception capture ('runtime_python.c').

```

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

```

```

6  * @author Ankush Yadav, Ankit Yadav,
   *   AuctaSapience
7  */
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 }

```

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  * @version 1.0.0
28  * @author Ankush Yadav, Ankit Yadav,
29  *   AuctaSapience
30  */
31
32 // --- FIX #1: Standard library and
33 //     external headers FIRST ---
34 #include <stdio.h>
35 #include <stdlib.h>
36 #include <string.h>
37
38 #ifdef AGENK_PYTHON_SUPPORT
39 #define PY_SSIZE_T_CLEAN
40 #include <Python.h>
41 #endif
42
43 // --- FIX #2: Your project's headers
44 //     LAST ---
45 #include "agenk_execution.h"
46 #include "agenk_core.h"
47 #include "object.h" // Needed for CLASS_
48 //     INIT()
49
50 // *** BEGIN WORKAROUND for incomplete
51 //     object.h ***
52 // These declarations are missing from
53 // the class_framework's public header.
54 // Adding them here resolves the "
55 // implicit declaration" warnings and
56 // allows
57 // the linker to correctly find the
58 // functions in the library.
59 void CLASS_INIT(void);
60 void CLASS_SHUTDOWN(void);
61 // *** END WORKAROUND ***
62
63 // --- Testing framework macros ---
64 static int g_tests_run = 0;
65 static int g_tests_passed = 0;
66 #define COLOR_GREEN "\x1B[32m"
67 #define COLOR_RED "\x1B[31m"
68 #define COLOR_RESET "\x1B[0m"
69 #define TEST_SUITE_START(name) printf("\
70     n---_Running_Test_Suite:_%s_---\n",
71     name)
72 #define TEST_CASE(name) printf("\
73     _[TEST]_
74     _%s\n", name)
75 #define ASSERT(condition)
76
77     \
78     do {
79
80         \
81         g_tests_run++;

```

```

56         \
        if (condition) {
57             \
            g_tests_passed++;
58             \
            printf(COLOR_GREEN "[PASS]
            %s:%d\n" COLOR_RESET, _
            _FILE_, _LINE_); \
59         } else {
60             \
            fprintf(stderr, COLOR_RED "[
            FAIL] %s:%d: Assertion
            failed: %s\n" \
61                 COLOR_RESET, _FILE_
            , _LINE_, #
            condition); \
62         }
63     } while (0)
64
65 // --- Test Suites ---
66
67 // ... (No changes to any of the test_
68 suite functions) ...
69 void test_suite_snippet_lifecycle() {
70     TEST_SUITE_START("SnippetLifecycle
71     &MemoryManagement");
72     AgenkSnippet* s = snippet_create("A
73     test_snippet_for_file_operations
74     .");
75     ASSERT(s != NULL);
76     AgenkCode* exec_code = code_create(
77     LANG_SHELL, NULL);
78     code_add_variant(exec_code, OS_LINUX
79     , "ls-l");
80     snippet_set_execution_code(s, exec_
81     code);
82     AgenkDependency* dep = dependency_
83     create(DEP_SOURCE, "http://
84     example.com/dep.git");
85     dependency_add_build_command(dep, "
86     make_all");
87     snippet_add_dependency(s, dep);
88     ASSERT(s->dependencies[0]->num_build
89     commands == 1);
90     TEST_CASE("DeepFreeofaComplex
91     Snippet");
92     snippet_free(s);
93     ASSERT(1);
94 }
95
96 void test_suite_shell_execution() {
97     TEST_SUITE_START("ShellExecution
98     Runtime");
99     AgenkSnippet* s = snippet_create("
100     Echo_test");
101     AgenkCode* code = code_create(LANG_
102     SHELL, NULL);
103     code_add_variant(code, OS_ANY, "echo
104     'Hello,Shell!'");
105     snippet_set_execution_code(s, code);
106     ExecutionContext ctx = { .input_uri
107     = NULL, .api_request = NULL };
108     ExecutionResult* result = engine_
109     execute(s, &ctx);
110     ASSERT(result != NULL);
111     if (result) {
112         ASSERT(result->exit_code == 0);
113         ASSERT(result->return_value !=
114         NULL);
115         if (result->return_value) {
116             printf("[INFO] C_Snippet
117             returned: %s\n", (
118             char*)result->return_
119             value);
120             free(result->return_value);
121         }
122         execution_result_free(result);
123     }
124     snippet_free(s);
125 }
126
127 #ifdef AGENK_PYTHON_SUPPORT

```

```

92     execute(s, &ctx);
93     ASSERT(result != NULL);
94     if (result) {
95         ASSERT(result->exit_code == 0);
96         ASSERT(result->stdout_buffer !=
97         NULL);
98         ASSERT(strncmp(result->stdout_
99         buffer, "Hello,Shell!", 13)
100         == 0);
101         execution_result_free(result);
102     }
103     snippet_free(s);
104 }
105
106 void test_suite_c_execution() {
107     TEST_SUITE_START("C_JIT_Execution_
108     Runtime");
109     const char* c_source =
110         "#include <agenk_core.h>\n"
111         "#include <stdio.h>\n"
112         "#include <stdlib.h>\n"
113         "#include <string.h>\n"
114         "ExecutionResult* process(const_
115         ExecutionContext* context) {
116         \n"
117         "    ExecutionResult* result = (
118         ExecutionResult*) calloc(1,
119         sizeof(ExecutionResult)); \n"
120         "    unsigned long id = core_
121         create_group(\"Test_from_C_
122         Snippet\"); \n"
123         "    char* result_str = (char*)
124         malloc(128); \n"
125         "    sprintf(result_str, \"C_
126         Snippet_created_group_with_
127         ID: %lu\", id); \n"
128         "    result->exit_code = 0; \n"
129         "    result->return_value = (
130         void*) result_str; \n"
131         "    return result; \n"
132         "    } \n";
133     AgenkSnippet* s = snippet_create("C_
134     JIT_core_API_test");
135     AgenkCode* code = code_create(LANG_C
136     , "process");
137     code_add_variant(code, OS_ANY, c_
138     source);
139     snippet_set_execution_code(s, code);
140     ExecutionContext ctx = { .input_uri
141     = NULL, .api_request = NULL };
142     ExecutionResult* result = engine_
143     execute(s, &ctx);
144     ASSERT(result != NULL);
145     if (result) {
146         ASSERT(result->exit_code == 0);
147         ASSERT(result->return_value !=
148         NULL);
149         if (result->return_value) {
150             printf("[INFO] C_Snippet
151             returned: %s\n", (
152             char*)result->return_
153             value);
154             free(result->return_value);
155         }
156         execution_result_free(result);
157     }
158     snippet_free(s);
159 }
160
161 #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_test");
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             agenk_core.create_group()')\n"
150         "    \n"
151         "    group_id=agenk_core.
152             create_group('Test_from_
153             Python')\n"
154         "    print(f'Python:Received_
155             group_ID:{group_id}')\n"
156         "    return_group_id\n";
157     code_add_variant(code, OS_ANY,
158         python_source);
159     snippet_set_execution_code(s, code);
160     ExecutionContext ctx = { .input_uri
161         = NULL, .api_request = NULL };
162     ExecutionResult* result = engine_
163         execute(s, &ctx);
164     ASSERT(result != NULL);
165     if(result) {
166         ASSERT(result->exit_code == 0);
167         ASSERT(result->return_value !=
168             NULL);
169         if (result->return_value) Py_
170             XDECREF((PyObject*)result->
171             return_value);
172         execution_result_free(result);
173     }
174     snippet_free(s);
175 }
176 #endif
177
178 void test_suite_failure_modes() {
179     TEST_SUITE_START("Failure_Mode_
180         Validation");
181     ExecutionContext ctx = { .input_uri
182         = NULL, .api_request = NULL };
183     TEST_CASE("Executing_Snippet_with_
184         invalid_dependency_URI");
185     AgenkSnippet* s_bad_git = snippet_
186         create("Bad_Git_URL");
187     AgenkDependency* dep_bad_git =
188         dependency_create(DEP_SOURCE, "
189             https://github.com/nonexistent/
190             repo12345.git");
191     snippet_add_dependency(s_bad_git,
192         dep_bad_git);
193     AgenkCode* code_dummy = code_create(
194         LANG_SHELL, NULL);
195     code_add_variant(code_dummy, OS_ANY,
196         "echo_'should_not_run'");
197     snippet_set_execution_code(s_bad_git
198         , code_dummy);
199     ExecutionResult* result_bad_git =
200         engine_execute(s_bad_git, &ctx);
201     ASSERT(result_bad_git != NULL &&
202         result_bad_git->exit_code != 0);
203     execution_result_free(result_bad_git
204         );
205     snippet_free(s_bad_git);
206     #ifdef AGENK_PYTHON_SUPPORT
207     TEST_CASE("Executing_Python_Snippet_
208         with_Syntax_Error");
209     AgenkSnippet* s_py_syntax = snippet_
210         create("Python_Syntax_Error");
211     AgenkCode* code_py_syntax = code_
212         create(LANG_PYTHON, "run_test");
213     code_add_variant(code_py_syntax, OS_
214         ANY, "def_run_test(context):\n
215             print_'hello'\n");
216     snippet_set_execution_code(s_py_
217         syntax, code_py_syntax);
218     ExecutionResult* result_py_syntax =
219         engine_execute(s_py_syntax, &ctx
220         );
221     ASSERT(result_py_syntax != NULL &&
222         result_py_syntax->exit_code !=
223         0);
224     ASSERT(result_py_syntax->stderr_
225         buffer != NULL && strstr(result_
226         py_syntax->stderr_buffer, "
227         SyntaxError"));
228     execution_result_free(result_py_
229         syntax);
230     snippet_free(s_py_syntax);
231     #endif
232 }
233
234 // --- Test Runner ---
235 int main() {
236     printf("=====\n");
237     printf("Running_AGENK_Execution_
238         Substrate_Tests\n");
239     printf("=====+\n");
240
241     // *** FIX: Revert to the correct
242     // function names from the class_
243     // framework library ***
244     CLASS_INIT();
245     ASSERT(engine_init("./agenk_cache"))
246         ;
247
248     test_suite_snippet_lifecycle();
249     test_suite_shell_execution();
250     test_suite_c_execution();
251     #ifdef AGENK_PYTHON_SUPPORT
252     test_suite_python_execution();
253     #else
254     printf("\n---Skipping_Python_
255         Test_Suite:_Python_support_
256         disabled---\n");
257     #endif
258     test_suite_failure_modes();
259
260     engine_shutdown();
261     // *** FIX: Revert to the correct
262     // function names from the class_
263     // framework library ***
264     CLASS_SHUTDOWN();
265
266     printf("\n=====\n");
267     printf("TEST_SUMMARY\n");
268     printf("=====\n");
269     if (g_tests_passed == g_tests_run) {
270         printf(COLOR_GREEN "SUCCESS:_All_
271             _%d_tests_passed.\n" COLOR_
272             RESET, g_tests_run);
273         return 0;
274     } else {
275         fprintf(stderr, COLOR_RED "
276             FAILURE:_%d_out_of_%d_tests_

```



```

225         failed.\n" COLOR_RESET,
           g_tests_run - g_tests_
             passed, g_tests_run)
226         ;
227     return 1;
228 }

```

Listing 43: The comprehensive test suite, including failure mode validation (`testeexecution.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 `AgentkTensor`.
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 `agentk_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
  " / Stable Native API) ---
2 set(CORE_SOURCES src/core/core.c)
3 if(AGENK_ENABLE_PYTHON)
4     list(APPEND CORE_SOURCES src/core/
5           bridge/native_bridge_python.c)
6 endif()
7 add_library(agenk_core STATIC ${CORE_
8             SOURCES})
9
10 target_include_directories(agenk_core
11                             PUBLIC
12                             ${CMAKE_CURRENT_SOURCE_DIR}/src/core
13                             /include
14 )
15 # The PUBLIC keyword is critical here.
16 # It ensures that any module
17 # that links against agenk_core (like
18 # agenk_execution) also gets the
19 # include paths for the foundational
20 # data structures.
21 target_link_libraries(agenk_core PUBLIC
22                       agenk_snippet agenk_tensor class_
23                       framework
24 )
25 if(AGENK_ENABLE_PYTHON)
26     target_link_libraries(agenk_core
27                           PRIVATE Python3::Python Python3
28                           ::NumPy)
29 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
   * lifecycle management.
10 *
11 * This API is designed to be immutable.
   * Functions added here become a
   * permanent
12 * part of the agent's core
   * functionality.
13 *
14 * @version 1.0.0
15 * @author Ankush Yadav, Ankit Yadav,
   * AuctaSapience
16 */
17
18 #ifndef AGENK_CORE_H
19 #define AGENK_CORE_H
20
21 #include "agenk_snippet.h" // For
   * AgenkTensor forward declaration
22 #include <stddef.h>
23 #include <stdint.h>
24 #include <stdbool.h>
25
26 // --- API Symbol Exposure for JIT
   * Runtimes ---
27 /**
28  * @brief Maps an API function's string
   * name to its address in memory.
29  * This is used by JIT runtimes (like
   * the C runtime) to link against
30  * the host application's live functions
   * .
31  */
32 typedef struct {
33     const char* name;
34     void* func;
35 } AgenkApiSymbol;
36
37 // --- API Self-Documentation Structures
   * ---
38 /** @brief Describes a single parameter
   * for a core API function. */
39 typedef struct {
40     const char* name;
41     const char* type;
42     const char* description;
43 } AgenkApiParameterDoc;
44
45 /** @brief Describes a single, stable
   * function exposed by the AGENK Core.
   */
46 typedef struct {
47     const char* function_name;
48     const char* description;
49     AgenkApiParameterDoc* parameters;
50     size_t num_parameters;
51     const char* return_type;
52     const char* return_description;
53 } AgenkApiFunctionDoc;
54
55 // --- Core API Function Declarations
   * ---
56
57 // Object Framework Bindings
58 unsigned long core_create_group(const
   * char* description);
59 void core_destroy_by_group(unsigned long
   * group_id);

```

```

62 // Tensor Substrate Bindings
63 AgenkTensor* core_tensor_create(const
   * size_t* shape, size_t ndim, int
   * dtype, int ctype);
64 void core_tensor_free(AgenkTensor*
   * tensor);
65 size_t core_tensor_get_ndim(const
   * AgenkTensor* tensor);
66 // ... other tensor accessors would be
   * added here (get_shape, get_data_ptr,
   * etc.)
67
68 // --- API Management Functions ---
69
70 /**
71  * @brief Retrieves the complete list of
   * core API symbols for JIT linking.
72  * @param count Pointer to an integer
   * which will be filled with the
   * number of symbols.
73  * @return A constant pointer to the
   * internal array of API symbols. The
   * caller
74  * does NOT own this memory and
   * must NOT free or modify it.
75  */
76 const AgenkApiSymbol* core_get_api_
   * symbols(int* count);
77
78 /**
79  * @brief Retrieves the complete,
   * machine-readable documentation for
   * the Core API.
80  * @param count Pointer to a size_t
   * which will be filled with the
   * number of functions.
81  * @return A constant pointer to the
   * internal array of function
   * documentation. The caller
82  * does NOT own this memory and
   * must NOT free it.
83  */
84 const AgenkApiFunctionDoc* core_get_api_
   * documentation(size_t* count);
85
86 #endif // AGENK_CORE_H

```

Listing 45: The public API header for the Core Substrate ('agenk_core.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 #include "agenk_core.h"
21 #include "object.h" // *** FIX #1:
22 // Include the header for the class
23 // framework API ***
24
25 // --- Core API Function Implementations
26 ---
27
28 unsigned long core_create_group(const
29     char* description) {
30     // This function simply acts as a
31     // stable wrapper around the
32     // underlying object framework's
33     // implementation.
34     // *** FIX #2: Use the correct
35     // function names (without CLASS_
36     // prefix) ***
37     return CREATE_GROUP(description);
38 }
39
40 void core_destroy_by_group(unsigned long
41     group_id) {
42     // Wrapper for the object framework's
43     // implementation.
44     // *** FIX #2: Use the correct
45     // function names (without CLASS_
```

```
46     prefix) ***
47     DESTROY_BY_GROUP(group_id);
48 }
49
50 // NOTE: Implementations for tensor
51 // functions would go here.
52 // For now, they are just stubs as
53 // defined in the header.
54
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
91 static const AgenkApiSymbol G_CORE_API_
92     SYMBOLS[] = {
93     {"core_create_group", (void*)core_
94         create_group},
95     {"core_destroy_by_group", (void*)
96         core_destroy_by_group},
97     {"core_tensor_create", (void*)core_
98         tensor_create},
99     {"core_tensor_free", (void*)core_
100         tensor_free},
101     {"core_tensor_get_ndim", (void*)core_
102         _tensor_get_ndim}
103 };
104
105 /**
106  * @brief Retrieves the complete list of
107  *       core API symbols for JIT linking.
108  */
109
110 const AgenkApiSymbol* core_get_api_
111     symbols(int* count) {
112     *count = sizeof(G_CORE_API_SYMBOLS)
113         / sizeof(G_CORE_API_SYMBOLS[0]);
114     return G_CORE_API_SYMBOLS;
115 }
116
117 /**
118  * @brief Retrieves the complete,
119  *       machine-readable documentation for
```

```

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

```

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

```



```

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; //
70             // Corresponds to AGENK_DATA_
71             // TYPE_FLOAT32
72         case 2: return NPY_UINT8; //
73             // Corresponds to AGENK_DATA_
74             // TYPE_UINT8
75         case 3: return NPY_INT16; //
76             // Corresponds to AGENK_DATA_
77             // TYPE_INT16
78         case 4: return NPY_UINT16; //
79             // Corresponds to AGENK_DATA_
80             // TYPE_UINT16
81         default: return -1;
82     }
83 }
84 // 3. Tensor Creation Binding
85 static PyObject* core_native_tensor_
86     create(PyObject *self, PyObject *
87     args) {
88     (void)self;
89     PyObject *shape_list;
90     unsigned int dtype_val, ctype_val;
91
92     if (!PyArg_ParseTuple(args, "O!II",
93         &PyList_Type, &shape_list, &
94         dtype_val, &ctype_val)) {
95         PyErr_SetString(PyExc_TypeError,
96             "Expected_(shape:_list,_"
97             "dtype:_int,ctype:_int)");
98         return NULL;
99     }
100
101     size_t ndim = (size_t)PyList_Size(
102         shape_list);
103     size_t *shape_c = (size_t*)malloc(
104         ndim * sizeof(size_t));
105     if (!shape_c) return PyErr_NoMemory
106         ();
107
108     for (size_t i = 0; i < ndim; ++i) {
109         PyObject *item = PyList_GetItem(
110             shape_list, i);
111         if (!PyLong_Check(item)) {
112             free(shape_c);
113             PyErr_SetString(PyExc_
114                 TypeError, "Shape_list_
115                 must_contain_only_
116                 integers.");
117             return NULL;
118         }
119         shape_c[i] = PyLong_AsSize_t(
120             item);
121     }
122
123     // Calls the stable core API
124     function
125     AgenkTensor *t = core_tensor_create(
126         shape_c, ndim, (int)dtype_val, (
127         int)ctype_val);
128     free(shape_c);
129
130     if (!t) {
131         PyErr_SetString(PyExc_
132             RuntimeError, "Core_C-level_
133             tensor_creation_failed.");
134         return NULL;
135     }
136
137     // Wrap the C pointer in a PyCapsule
138     // , attaching our custom
139     // destructor.
140     return PyCapsule_New((void *)t,
141         AGENK_TENSOR_CAPSULE_NAME,
142         tensor_capsule_destructor);
143 }
144
145 // 4. Tensor to NumPy Bridge
146 static PyObject* core_native_tensor_to_
147     numpy(PyObject *self, PyObject *args
148     ) {
149     (void)self;
150     PyObject* capsule_obj;
151     if (!PyArg_ParseTuple(args, "O", &
152         capsule_obj) || !PyCapsule_
153         CheckExact(capsule_obj)) {
154         PyErr_SetString(PyExc_TypeError,
155             "Argument_must_be_a_valid_
156             AgenkTensor_handle.");
157         return NULL;
158     }
159
160     AgenkTensor* t = (AgenkTensor*)
161         PyCapsule_GetPointer(capsule_obj
162         , AGENK_TENSOR_CAPSULE_NAME);
163     if (!t) { PyErr_SetString(PyExc_
164         ValueError, "Invalid_AgenkTensor_
165         _handle."); return NULL; }
166
167     npy_intp* dims = (npy_intp*)malloc(
168         core_tensor_get_ndim(t) * sizeof
169         (npy_intp));
170     if (!dims) return PyErr_NoMemory();
171     for (size_t i = 0; i < core_tensor_
172         get_ndim(t); ++i) {
173         // This is a placeholder; a full
174         // implementation would need a
175         // core_tensor_get_shape()
176         // For now, we assume direct (
177         // but unsafe) access for the
178         // prototype.
179         // In the final version, the
180         // core API would provide a
181         // shape accessor.
182         dims[i] = 0; // Placeholder
183     }
184
185     // This is also a placeholder until
186     // the full API is exposed
187     int numpy_type = NPY_UINT8; //
188     Placeholder
189
190     PyObject* np_array = PyArray_
191         SimpleNew(core_tensor_get_ndim(t
192         ), dims, numpy_type);
193     free(dims);
194
195     // Placeholder for memcpy
196     // void* np_data = PyArray_DATA((

```

```

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
154     new object lifecycle group."},
155     {"tensor_create", core_native_tensor_
156     create, METH_VARARGS, "Creates a
157     new AgenkTensor and returns an
158     opaque handle."},
159     {"tensor_to_numpy", core_native_
160     tensor_to_numpy, METH_VARARGS, "
161     Converts an AgenkTensor handle
162     to a NumPy array (placeholder)."},
163     {NULL, NULL, 0, NULL} // Sentinel
164 };
165
166 static struct PyModuleDef agenk_core_
167 module = {
168     PyModuleDef_HEAD_INIT,
169     "agenk_core", // The name to use in
170     Python: `import agenk_core`
171     "Native C bridge for the AGENK Core
172     Substrate.",
173     -1,
174     AgenkCoreMethods
175 };
176
177 // --- Module Initializer ---
178 PyMODINIT_FUNC PyInit_agenk_core(void) {
179     PyObject* m = PyModule_Create(&agenk_
180     core_module);
181     if (m == NULL) {
182         return NULL;
183     }
184
185     // This macro expands to code that
186     includes its own return
187     statement
188     // on failure. It must be called on
189     its own line.
190     import_array();
191
192     return m;
193 }

```

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 ${
21     CMAKE_BINARY_DIR}/lib)
22 set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${
23     CMAKE_BINARY_DIR}/bin)
24
25 if(NOT CMAKE_BUILD_TYPE)
26     set(CMAKE_BUILD_TYPE Debug)
27 endif()
28 message(STATUS "Build type set to: ${
29     CMAKE_BUILD_TYPE}")
30
31 #
32
33 # =====
34 # SECTION 2: VENDORED DEPENDENCY
35 # MANAGEMENT
36 # =====
37
38 find_package(Threads REQUIRED)
39 find_library(MATH_LIBRARY m)
40
41 # --- FlatCC, LMDB, MD5, cJSON... (These

```

```

are correct and unchanged) ---
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 add_library(lmdb STATIC deps/lmdb/mdb.c
deps/lmdb/midl.c)
39 target_include_directories(lmdb PUBLIC $
{CMAKE_CURRENT_SOURCE_DIR}/deps/lmdb
)
40 add_library(md5 STATIC deps/md5/md5.c)
41 target_include_directories(md5 PUBLIC ${
CMAKE_CURRENT_SOURCE_DIR}/deps/md5)
42 set(CJSON_LIB "")
43 if(EXISTS ${CMAKE_CURRENT_SOURCE_DIR}/
deps/cJSON/cJSON.c)
44 message(STATUS "cJSON_source_found,
building_library.")
45 add_library(cjson STATIC deps/cJSON/
cJSON.c)
46 target_include_directories(cjson
PUBLIC ${CMAKE_CURRENT_SOURCE_
DIR}/deps/cJSON)
47 set(CJSON_LIB cjson)
48 set(CJSON_FOUND TRUE)
49 endif()
50
51 # --- TinyCC (JIT Compiler, built via
ExternalProject) ---
52 ExternalProject_Add(
53 tcc_external_project
54 SOURCE_DIR ${CMAKE_CURRENT_SOURCE_
DIR}/deps/tcc
55 CONFIGURE_COMMAND ""
56 BUILD_COMMAND ""
57 INSTALL_COMMAND ""
58 )
59
60 add_custom_target(build_tcc_library ALL
# Step 1: Run configure.
61 COMMAND ${CMAKE_COMMAND} -E chdir ${
CMAKE_CURRENT_SOURCE_DIR}/deps/
tcc
62 ./configure --libdir=${CMAKE
CURRENT_SOURCE_DIR}/
deps/tcc/lib
63 --extra-cflags="
-fPIC"
64
65 # Step 2: Manually patch the correct
sub-directory Makefile to
disable bcheck.
66 COMMAND ${CMAKE_COMMAND} -P ${CMAKE_
CURRENT_BINARY_DIR}/cmake_script
_patch_tcc_makefile.cmake
67
68 # Step 3: THE FIX - Run surgical `
make` commands to build ONLY the
libraries we need.
69 COMMAND ${CMAKE_COMMAND} -E chdir ${
CMAKE_CURRENT_SOURCE_DIR}/deps/
tcc make libtcc.a
70
71 # Invoke make's default target,
which builds ../libtcc1.a from
the lib/ directory.
72 COMMAND ${CMAKE_COMMAND} -E chdir ${
CMAKE_CURRENT_SOURCE_DIR}/deps/
tcc/lib make
73
74 # Step 4: Copy the final compiled
libraries from their correct
locations.
75 COMMAND ${CMAKE_COMMAND} -E copy_if_
different
76 ${CMAKE_CURRENT_SOURCE_DIR}/
deps/tcc/libtcc.a
77 ${CMAKE_BINARY_DIR}/lib/
78 # The 'lib/Makefile' places its
output in the parent directory
('deps/tcc').
79 COMMAND ${CMAKE_COMMAND} -E copy_if_
different
80 ${CMAKE_CURRENT_SOURCE_DIR}/
deps/tcc/libtcc1.a
81 ${CMAKE_BINARY_DIR}/lib/
82
83 DEPENDS tcc_external_project
84 )
85
86 # This CMake script patches the correct
Makefile in the lib/ subdirectory.
87 file(WRITE ${CMAKE_CURRENT_BINARY_DIR}/
cmake_script_patch_tcc_makefile.
cmake
88 "
89 set(TCC_LIB_MAKEFILE_PATH "${CMAKE
CURRENT_SOURCE_DIR}/deps/tcc/lib/
Makefile")
90 file(READ ${TCC_LIB_MAKEFILE_PATH} TCC_
LIB_MAKEFILE_CONTENT)
91 string(REPLACE "\bcheck.o" "\b#bcheck.o
" TCC_LIB_MAKEFILE_PATCHED "${TCC_
LIB_MAKEFILE_CONTENT}")
92 file(WRITE ${TCC_LIB_MAKEFILE_PATH} "\
${TCC_LIB_MAKEFILE_PATCHED}")
93 message(STATUS "Patched TCC lib/
Makefile to disable bcheck.")
94 "
95 )
96
97 add_library(libtcc STATIC IMPORTED
GLOBAL)
98 set_target_properties(libtcc PROPERTIES
IMPORTED_LOCATION "${CMAKE_BINARY_
DIR}/lib/libtcc.a"
99 )
100 add_dependencies(libtcc build_tcc_
library)
101
102 # =====
103 # SECTION 3: AUTOMATED
104 # CODE GENERATION
105 # =====
106
107 set(FBS_SCHEMA ${CMAKE_CURRENT_SOURCE_
DIR}/src/object_store/core/object.
fbs)
108
109 set(GENERATED_INCLUDE_DIR ${CMAKE_
CURRENT_BINARY_DIR}/generated/object_
store)
110
111 file(MAKE_DIRECTORY ${GENERATED_INCLUDE_
DIR})
112
113 set(GENERATED_FILES
${GENERATED_INCLUDE_DIR}/object_
builder.h
114 ${GENERATED_INCLUDE_DIR}/object_
reader.h
115 ${GENERATED_INCLUDE_DIR}/object_

```

```

116     verifier.h
117     ${GENERATED_INCLUDE_DIR}/flatbuffers
118     _common.h
119 )
120 add_custom_command(
121     OUTPUT ${GENERATED_FILES}
122     COMMAND $<TARGET_FILE:flatcc_cli> -a
123         -o ${GENERATED_INCLUDE_DIR} ${FBS_SCHEMA}
124     DEPENDS ${FBS_SCHEMA} flatcc_cli
125     COMMENT "Generating C headers from FlatBuffers schema"
126 )
127 add_custom_target(generate_object_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 )
139 target_include_directories(agenk_tensor
140     PUBLIC
141     ${CMAKE_CURRENT_SOURCE_DIR}/src/
142     tensors/include
143 )
144 add_library(agenk_sensation STATIC
145     src/senses/builder/tool_check.c
146     src/senses/builder/build_controller.c
147     src/senses/core/sense_manager.c
148     src/senses/core/sense_registry.c
149 )
150 target_include_directories(agenk_sensation
151     PUBLIC
152     ${CMAKE_CURRENT_SOURCE_DIR}/src/
153     senses/include
154 )
155 target_compile_definitions(agenk_sensation
156     PRIVATE
157     -DAGENK_SOURCE_DIR="${CMAKE_SOURCE_DIR}"
158 )
159 target_link_libraries(agenk_sensation
160     PRIVATE
161     agenk_tensor
162     libtcc
163     object_store
164     md5
165     Threads::Threads
166 )
167 add_library(class_framework STATIC
168     src/objects/core/object_impl.c
169     src/objects/core/object_registry.c
170     src/objects/core/object_db.c
171 )
172 target_include_directories(class_framework
173     PUBLIC
174     ${CMAKE_CURRENT_SOURCE_DIR}/src/objects
175 )
176 target_link_libraries(class_framework
177     PRIVATE
178     libmdb
179     Threads::Threads
180 )
181
182 # =====
183 # SECTION 5: EXECUTABLE
184 # TARGETS (TESTS & EXAMPLES)
185 # =====
186 add_executable(test_tensors src/tensors/
187     test/test_tensors.c)
188 target_link_libraries(test_tensors
189     PRIVATE
190     agenk_tensor
191 )
192 add_executable(test_sensation src/senses/
193     test/test_sensation.c)
194 target_link_libraries(test_sensation
195     PRIVATE
196     agenk_sensation
197 )
198 add_executable(test_sensation_build src/
199     senses/test/test_sensation_build.c)
200 target_link_libraries(test_sensation_build
201     PRIVATE
202     agenk_sensation
203 )
204 # Find the math library
205 find_library(MATH_LIBRARY m)
206 # Add the link to the test_sense_registry executable
207 add_executable(test_sense_registry src/
208     senses/test/test_sense_registry.c)
209 target_link_libraries(test_sense_registry
210     PRIVATE
211     agenk_sensation
212     MATH_LIBRARY
213 )
214 add_executable(test_concurrency src/
215     objects/test/test_concurrency.c)
216 target_link_libraries(test_concurrency
217     PRIVATE
218     class_framework
219     Threads::Threads
220 )
221 add_executable(test_errors src/objects/

```



```

    test/test_errors.c)
218 target_link_libraries(test_errors
    PRIVATE class_framework)
219
220 add_executable(test_hierarchy src/
    objects/test/test_hierarchy.c)
221 target_link_libraries(test_hierarchy
    PRIVATE class_framework)
222
223 add_executable(test_stress src/objects/
    test/test_stress.c)
224 target_link_libraries(test_stress
    PRIVATE class_framework)
225
226 add_executable(example_basic_usage src/
    objects/examples/basic_usage.c)
227 target_link_libraries(example_basic_
    usage PRIVATE class_framework)
228
229 add_executable(example_lifecycle src/
    objects/examples/lifecycle_and_
    groups.c)
230 target_link_libraries(example_lifecycle
    PRIVATE class_framework)
231
232 add_executable(test_object_store src/
    object_store/test/test_object_store.
    c)
233 target_link_libraries(test_object_store
    PRIVATE object_store Threads::
    Threads ${CJSON_LIB})
234
235
236 # =====
237 # SECTION 6: CTEST INTEGRATION
238 # =====
239 enable_testing()
240 add_test(NAME Tensor.Substrate
    COMMAND test_tensors)
241 add_test(NAME Sensation.Subsystem.Core
    COMMAND test_sensation)
242 add_test(NAME Sensation.Subsystem.Build
    COMMAND test_sensation_build)
243 add_test(NAME Sensation.Registry COMMAND
    test_sense_registry)
244 add_test(NAME ObjectStore.Core
    COMMAND test_object_store)
245 add_test(NAME Framework.Concurrency
    COMMAND test_concurrency)
246 add_test(NAME Framework.ErrorHandling
    COMMAND test_errors)
247 add_test(NAME Framework.Hierarchy
    COMMAND test_hierarchy)
248 add_test(NAME Framework.Stress
    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
    char* sense_id);
28 void sense_object_free(SenseObject*
    sense);
29
30 // --- Blueprint "Setter" API ---
31 void sense_object_set_source_uri(
    SenseObject* sense, const char* uri)
    ;
32 void sense_object_set_library_template(
    SenseObject* sense, const char*
    template_path);
33 void sense_object_set_installed_name(
    SenseObject* sense, const char* name
    );
34 void sense_object_add_supported_
    extension(SenseObject* sense, const
    char* ext);
35 void sense_object_add_build_tool(
    SenseObject* sense, const char* tool
    );
36 void sense_object_add_build_command(
    SenseObject* sense, const char*
    command);
37 void sense_object_set_jit_driver(
    SenseObject* sense, const char*
    source_code);
38
39 // --- Blueprint "Getter" API ---
40 const char* sense_object_get_id(const
    SenseObject* sense);
41 const char* sense_object_get_source_uri(
    const SenseObject* sense);
42 const char* sense_object_get_installed_
    name(const SenseObject* sense);
43 size_t sense_object_get_num_extensions(
    const SenseObject* sense);
44 const char* sense_object_get_extension(
    const SenseObject* sense, size_t
    index);
45 size_t sense_object_get_num_build_tools(
    const SenseObject* sense);
46 const char* sense_object_get_build_tool(
    const SenseObject* sense, size_t
    index);
47 size_t sense_object_get_num_build_
    commands(const SenseObject* sense);
48 const char* sense_object_get_build_
    command(const SenseObject* sense,
    size_t index);
49
50 // --- Database Interaction API ---
51 bool sense_registry_add_blueprint(const
    SenseObject* sense);
52 SenseObject* sense_registry_get_by_id(
    const char* sense_id);
53 char* sense_registry_find_by_extension(
    const char* extension);
54
55 #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 -
    CORRECTED
2
3 #ifndef SENSE_PRIVATE_H
4 #define SENSE_PRIVATE_H
5
6 #include "../include/agenk_sensation.h"
7 #include <stddef.h>
8 #include <stdbool.h> // For bool type
9
10 // --- THE FIX: Define the struct with a
    name that matches the public
    typedef ---
11 struct SenseObject {
12     char* sense_id;
13     char** supported_extensions;
14     size_t num_extensions;
15     size_t extensions_capacity;
16
17     // The Build Manifest
18     char** required_build_tools;
19     size_t num_tools;
20     size_t tools_capacity;
21     char* dependency_source_uri;
22     char** build_commands;
23     size_t num_commands;
24     size_t commands_capacity;
25     char* library_path_template;
26     char* installed_library_name;
27
28     // The JIT Driver Script
29     char* source_code;
30 };
31
32 // Forward declare the internal utility
    function
33 bool check_tools(const SenseObject*
    sense);
34
35 #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 #if defined(__linux__) || defined(__APPLE__) || defined(__FreeBSD__)
20 #define _POSIX_C_SOURCE 200809L
21 #endif
22
23 #include "../private/sense_private.h"
24 #include "../object_store/object_store.h"
25 #include "md5.h"
26 #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
32 static bool add_string_to_array(char*** array, size_t* count, size_t* capacity, const char* str){
33     if (*count >= *capacity) {
34         size_t new_cap = (*capacity == 0) ? 8 : *capacity * 2;
35         char** new_arr = (char**)realloc(*array, new_cap * sizeof(char*));
36         if (!new_arr) return false;
37         *array = new_arr; *capacity = new_cap;
38     }
39     (*array)[*count] = safe_strdup(str);
40     if (!(*array)[*count]) return false;
41     (*count)++;
42     return true;
43 }
44
45 // --- Forward Declarations ---
46 static c_object_t* sense_object_to_c_object(const SenseObject* sense);
47 static SenseObject* sense_object_from_c_object(const c_object_t* obj);
48 static void get_key_for_sense_id(const char* sense_id, unsigned char key_out[OS_MAX_KEY_SIZE]);
49
50 // --- Public API ---
51 bool sense_registry_init(const char* db_path) {
52     if (g_store) return true;
53     if (object_store_create(&g_store) != 0) return false;
54     if (object_store_init_db(g_store, db_path, 0, 0) != FB_SERIALIZER_OK) {
55         object_store_destroy(&g_store);
56         return false;
57     }
58     return (object_store_create_table(g_store, SENSE_TABLE_NAME) == FB_SERIALIZER_OK);
59 }
60
61 void sense_registry_shutdown() { object_store_destroy(&g_store); }
62
63 SenseObject* sense_object_create(const char* sense_id) {
64     SenseObject* s = (SenseObject*)calloc(1, sizeof(SenseObject));
65     if(s) s->sense_id = safe_strdup(sense_id);
66     return s;
67 }
68
69 void sense_object_free(SenseObject* sense) {
70     if (!sense) return;
71     free(sense->sense_id); free(sense->

```

```

68     dependency_source_uri);
69     free(sense->library_path_template);
70     free(sense->installed_library_name);
71     free(sense->source_code);
72     if (sense->supported_extensions) {
73         for (size_t i = 0; i < sense->
74             num_extensions; ++i) free(
75                 sense->supported_extensions[
76                     i]);
77         free(sense->supported_extensions
78             );
79     }
80     if (sense->required_build_tools) {
81         for (size_t i = 0; i < sense->
82             num_tools; ++i) free(sense->
83                 required_build_tools[i]);
84         free(sense->required_build_tools
85             );
86     }
87     if (sense->build_commands) {
88         for (size_t i = 0; i < sense->
89             num_commands; ++i) free(
90                 sense->build_commands[i]);
91         free(sense->build_commands);
92     }
93     free(sense);
94 }

95 void sense_object_set_source_uri(
96     SenseObject* s, const char* v) { if(
97     s){free(s->dependency_source_uri); s
98     ->dependency_source_uri=safe_strdup(
99     v);}}
100 void sense_object_set_library_template(
101     SenseObject* s, const char* v) { if(
102     s){free(s->library_path_template); s
103     ->library_path_template=safe_strdup(
104     v);}}
105 void sense_object_set_installed_name(
106     SenseObject* s, const char* v) { if(
107     s){free(s->installed_library_name);
108     s->installed_library_name=safe_
109     strdup(v);}}
110 void sense_object_add_supported_
111 extension(SenseObject* s, const char
112 * v) { if(s) add_string_to_array(&s-
113     >supported_extensions, &s->num_
114     extensions, &s->extensions_capacity,
115     v);}
116 void sense_object_add_build_tool(
117     SenseObject* s, const char* v) { if(
118     s) add_string_to_array(&s->required_
119     build_tools, &s->num_tools, &s->
120     tools_capacity, v);}
121 void sense_object_add_build_command(
122     SenseObject* s, const char* v) { if(
123     s) add_string_to_array(&s->build_
124     commands, &s->num_commands, &s->
125     commands_capacity, v);}
126 void sense_object_set_jit_driver(
127     SenseObject* s, const char* v) { if(
128     s){free(s->source_code); s->source_
129     code=safe_strdup(v);}}

130 const char* sense_object_get_id(const
131     SenseObject* s) { return s ? s->
132     sense_id : NULL; }
133 const char* sense_object_get_source_uri(
134     const SenseObject* s) { return s ? s
135     ->dependency_source_uri : NULL; }
136 const char* sense_object_get_installed_
137 name(const SenseObject* s) { return
138     s ? s->installed_library_name : NULL
139     ; }
140 size_t sense_object_get_num_extensions(
141     const SenseObject* s) { return s ? s
142     ->num_extensions : 0; }
143 const char* sense_object_get_extension(
144     const SenseObject* s, size_t i) {
145     return (s && i < s->num_extensions)
146     ? s->supported_extensions[i] : NULL;
147     }
148 size_t sense_object_get_num_build_tools(
149     const SenseObject* s) { return s ? s
150     ->num_tools : 0; }
151 const char* sense_object_get_build_tool(
152     const SenseObject* s, size_t i) {
153     return (s && i < s->num_tools) ? s->
154     required_build_tools[i] : NULL; }
155 size_t sense_object_get_num_build_
156 commands(const SenseObject* s) {
157     return s ? s->num_commands : 0; }
158 const char* sense_object_get_build_
159 command(const SenseObject* s, size_t
160     i) { return (s && i < s->num_
161     commands) ? s->build_commands[i] :
162     NULL; }

163 bool sense_registry_add_blueprint(const
164     SenseObject* sense) {
165     if (!g_store || !sense || !sense->
166         sense_id) return false;
167     c_object_t* obj = sense_object_to_c_
168     object(sense);
169     if (!obj) return false;
170     unsigned char key[OS_MAX_KEY_SIZE];
171     get_key_for_sense_id(sense->sense_id
172         , key);
173     fb_serializer_status_t status =
174         object_store_put_object(g_store,
175         SENSE_TABLE_NAME, key, obj);
176     free_c_object(obj);
177     return (status == FB_SERIALIZER_OK);
178 }

179 SenseObject* sense_registry_get_by_id(
180     const char* sense_id) {
181     if (!g_store || !sense_id) return
182     NULL;
183     unsigned char key[OS_MAX_KEY_SIZE];
184     get_key_for_sense_id(sense_id, key);
185     c_object_t* obj = NULL;
186     if (object_store_get_object(g_store,
187         SENSE_TABLE_NAME, key, &obj) !=
188         FB_SERIALIZER_OK) return NULL;
189     SenseObject* sense = sense_object_
190     from_c_object(obj);
191     free_c_object(obj);
192     return sense;
193 }

194 char* sense_registry_find_by_extension(
195     const char* extension) {
196     (void)extension; // Proper
197     implementation requires DB
198     cursors, a future task.
199     return safe_strdup("sense_jpeg_v1");
200     // Stub for testing.
201 }

```

```

129 static c_field_t* create_string_array_
130 field(const char* name, char** data,
      size_t count) {
131     if (count == 0) return NULL;
132     c_field_t* field = (c_field_t*)
      calloc(1, sizeof(c_field_t));
133     field->name = safe_strdup(name);
134     field->value = (c_field_value_t*)
      calloc(1, sizeof(c_field_value_t
      ));
135     field->value->type = C_FIELD_VALUE_
      TYPE_ARRAY_VALUE;
136
137     c_field_t* array_tail = NULL;
138     for (size_t i = 0; i < count; ++i) {
139         c_field_t* item = (c_field_t*)
      calloc(1, sizeof(c_field_t))
      ;
140
141         // Create a synthetic name for
      the array item (e.g., "0",
      "1", etc.)
142         char item_name[32];
143         snprintf(item_name, sizeof(item_
      name), "%zu", i);
144         item->name = safe_strdup(item_
      name);
145
146         item->value = create_c_string_
      value(data[i]);
147         if (!field->value->data.array_
      values) field->value->data.
      array_values = item;
148         else array_tail->next = item;
149         array_tail = item;
150     }
151     return field;
152 }
153
154 static c_object_t* sense_object_to_c_
      object(const SenseObject* sense) {
155     c_object_t* obj = (c_object_t*)
      calloc(1, sizeof(c_object_t));
156     obj->object_id = safe_strdup(sense->
      sense_id);
157     obj->table_name = safe_strdup(SENSE_
      TABLE_NAME);
158     c_field_t* tail = NULL;
159
160     // This local helper is safer than a
      complex macro
161     void add_field_local(c_field_t**
      head, c_field_t** tail, c_field_
      t* f) {
162         if (!f) return;
163         if (*head == NULL) {
164             *head = f;
165             *tail = f;
166         } else {
167             (*tail)->next = f;
168             *tail = f;
169         }
170     }
171
172     // ... string fields added correctly
      with a helper ...
173     c_field_t* temp_field;
174     #define ADD_STRING_FIELD(fname, fvalue)
      do { \
175         if (fvalue) { \
176             temp_field = (c_field_t*)calloc
      (1, sizeof(c_field_t)); \
177             temp_field->name = safe_strdup((
      fname)); \
178             temp_field->value = create_c_
      string_value((fvalue)); \
179             add_field_local(&obj->fields, &
      tail, temp_field); \
180         } \
181     } while(0)
182
183     ADD_STRING_FIELD("dependency_source_
      uri", sense->dependency_source_
      uri);
184     ADD_STRING_FIELD("library_path_
      template", sense->library_path_
      template);
185     ADD_STRING_FIELD("installed_library_
      name", sense->installed_library_
      name);
186     ADD_STRING_FIELD("source_code",
      sense->source_code);
187     add_field_local(&obj->fields, &tail,
      create_string_array_field("
      supported_extensions", sense->
      supported_extensions, sense->num
      extensions));
188     add_field_local(&obj->fields, &tail,
      create_string_array_field("
      required_build_tools", sense->
      required_build_tools, sense->num
      tools));
189     add_field_local(&obj->fields, &tail,
      create_string_array_field("
      build_commands", sense->build_
      commands, sense->num_commands));
190
191     return obj;
192 }
193
194 static SenseObject* sense_object_from_c_
      object(const c_object_t* obj) {
195     if (!obj) return NULL;
196     SenseObject* sense = sense_object_
      create(obj->object_id);
197     if (!sense) return NULL;
198
199     for (c_field_t* field = obj->fields;
      field; field = field->next) {
200         if (!field->value || field->
      value->type == C_FIELD_VALUE_
      TYPE_NONE) continue;
201
202         if (strcmp(field->name, "
      dependency_source_uri") == 0
      && field->value->type == C_
      FIELD_VALUE_TYPE_STRING_
      VALUE)
203             sense_object_set_source_uri(
      sense, field->value->
      data.string_val);
204         else if (strcmp(field->name, "
      library_path_template") == 0
      && field->value->type == C_
      FIELD_VALUE_TYPE_STRING_
      VALUE)
205             sense_object_set_library_
      template(sense, field->
      value->data.string_val);

```



```

206     else if (strcmp(field->name, "
207         installed_library_name") ==
208         0 && field->value->type == C_
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, "
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
222     // --- THE DEFINITIVE FIX ---
223     // The issue was a subtle use-
224     // after-free. The functions
225     // sense_object_add_*
226     // were not making copies, they
227     // were taking ownership of the
228     // pointers from
229     // the temporary c_object t.
230     // When that temporary object
231     // was freed, the pointers
232     // inside the new SenseObject
233     // became invalid. The fix is
234     // to ensure the add
235     // functions make their own
236     // copies.
237     else if (strcmp(field->name, "
238         supported_extensions") == 0
239         && field->value->type == C_
240         FIELD_VALUE_TYPE_ARRAY_VALUE
241         ) {
242         for (c_field_t* item = field
243             ->value->data.array_
244             values; item; item =
245             item->next) {
246             if(item->value && item->
247                 value->type == C_
248                 FIELD_VALUE_TYPE_
249                 STRING_VALUE)
250                 sense_object_add_
251                 supported_
252                 extension(sense,
253                 item->value->
254                 data.string_val)
255                 ;
256         }
257     }
258     else if (strcmp(field->name, "
259         required_build_tools") == 0
260         && field->value->type == C_
261         FIELD_VALUE_TYPE_ARRAY_VALUE
262         ) {
263         for (c_field_t* item = field
264             ->value->data.array_
265             values; item; item =
266             item->next) {
267             if(item->value && item->
268                 value->type == C_
269                 FIELD_VALUE_TYPE_
270                 STRING_VALUE)
271                 sense_object_add_
272                 build_tool(sense
273                 , item->value->
274                 data.string_val)
275                 ;
276         }
277     }
278     }
279     else if (strcmp(field->name, "
280         build_commands") == 0 &&
281         field->value->type == C_
282         FIELD_VALUE_TYPE_ARRAY_VALUE
283         ) {
284         for (c_field_t* item =
285             field->value->data.
286             array_values; item;
287             item = item->next) {
288             if(item->value && item->
289                 value->type == C_
290                 FIELD_VALUE_TYPE_
291                 STRING_VALUE)
292                 sense_object_add_
293                 build_command(
294                 sense, item->
295                 value->data.
296                 string_val);
297         }
298     }
299     }
300     return sense;
301 }
302
303 static void get_key_for_sense_id(const
304 char* sense_id, unsigned char key_
305 out[OS_MAX_KEY_SIZE]) {
306     md5_state_t pms;
307     md5_init(&pms);
308     md5_append(&pms, (const md5_byte_t*)
309         sense_id, strlen(sense_id));
310     md5_finish(&pms, key_out);
311     memset(key_out + 16, 0, OS_MAX_KEY_
312         SIZE - 16);
313 }

```

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 #include "../private/sense_private.h"
23 #include <string.h>
24 #include <stdlib.h>
25 #include <stdio.h>
26
27 #if defined(_WIN32) || defined(_WIN64)
28 #include <windows.h>
29 #else
30 #include <unistd.h>
31 #include <sys/wait.h>
32 #include <fcntl.h>
33 #endif
34
35 /**
36 * @brief Checks if a command-line tool
37 * is available and executable in the
38 * system's PATH.
39 *
40 * This function attempts to execute the
41 * tool with a harmless flag (like --
42 * 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     snprintf(command, sizeof(command), "
62         where %s > nul 2>&1", tool_name)
63     ;
64     return (system(command) == 0);
65 #else
66     pid_t pid = fork();
67     if (pid == -1) {
68         perror("[TOOL_CHECK] fork failed
69             ");
70         return false;
71     } else if (pid == 0) { // Child
72         process
73         int dev_null_fd = open("/dev/
74             null", O_WRONLY);
75         if (dev_null_fd != -1) {
76             dup2(dev_null_fd, STDOUT_
77                 FILENO);
78             dup2(dev_null_fd, STDERR_
79                 FILENO);
80             close(dev_null_fd);
81         }
82         char* const argv[] = {(char*)
83             tool_name, "--version", NULL
84             };
85         execvp(tool_name, argv);
86         _exit(127); // Command not found
87     } else { // Parent process
88         int status;
89         waitpid(pid, &status, 0);
90         return (WIFEXITED(status) &&
91             WEXITSTATUS(status) != 127);
92     }
93 }
94 #endif
95 }
96
97 /**
98 * @brief (THE MISSING FUNCTION)
99 * Iterates through a SenseObject's
100 * required tools.
101 *
102 * @param sense The SenseObject
103 * containing the build manifest.
104 * @return True if all required tools
105 * are found, false otherwise.
106 */
107 bool check_tools(const SenseObject*
108 sense) {
109     for (size_t i = 0; i < sense->num_
110         tools; i++) {
111         const char* tool = sense->
112             required_build_tools[i];
113         if (!tool_check_exists(tool)) {
114             fprintf(stderr, "[BUILD_
115                 ERROR] Required tool not
116                 found: %s\n", tool);
117             return false;
118         }
119     }
120     return true;
121 }

```

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
   * execute the build manifest defined
   * in
6  * a SenseObject. It handles checking
   * for dependencies, downloading
   * source code,
7  * running build commands, and
   * installing the final library
   * artifact.
8  *
9  * It is designed to be robust and cross
   * -platform, providing detailed
   * logging
10 * for debugging build failures.
11 *
12 * @version 1.0.0
13 * @author Ankush Yadav, Ankit Yadav,
   *   AuctaSapience
14 */
15
16 // --- Feature Test Macros ---
17 // Define these macros at the very top
   // of the file, before any includes.
18 // This is the standard way to request
   // functions and definitions from POSIX
19 // and XOPEN standards that are not part
   // of the strict ANSI C standard.
20 // _DEFAULT_SOURCE is the modern way on
   // glibc to get everything.
21 // _XOPEN_SOURCE=700 is a more portable
   // way to request modern POSIX features
22
23 #if defined(__linux__) || defined(__APPLE__) || defined(__FreeBSD__)
   #define _XOPEN_SOURCE 700
   #define _DEFAULT_SOURCE
24 #endif
25
26 #include "../private/sense_private.h"
27 #include <stdio.h>
28 #include <stdlib.h>
29 #include <string.h>
30
31 #if defined(_WIN32) || defined(_WIN64)
32     #include <windows.h>
33     #include <direct.h>
34     #define MKDIR(path) _mkdir(path)
35 #else
36     #include <sys/stat.h>
37     #include <sys/types.h>
38     #include <unistd.h>
39     #include <ftw.h>
40     #define MKDIR(path) mkdir(path,
41         0755)
42 #endif
43
44 //=====
45 // SECTION 1: INTERNAL
46 // HELPER FUNCTIONS
47 //=====
48
49 /**
50  * @brief A simple, cross-platform
   *   wrapper for running a system
   *   command.
51  * A production-grade version would have
   *   more robust output/error capturing
52  *
   * @param command The full command
   *   string to execute.

```

```

53  * @param working_dir The directory to
   *   execute the command in.
54  * @return 0 on success, non-zero on
   *   failure.
55  */
56 static int run_command(const char*
   command, const char* working_dir) {
57     char full_command[4096];
58     #if defined(_WIN32) || defined(_WIN64)
59         snprintf(full_command, sizeof(full_
   command), "cmd_/c_ \"cd_/d_ \"%s_\"
   _&&_s_\"", working_dir, command)
   ;
60     #else
61         snprintf(full_command, sizeof(full_
   command), "cd_ \"%s_\" _&&_s_",
   working_dir, command);
62     #endif
63     printf("[BUILD]_Executing:_%s\n",
   full_command);
64     return system(full_command);
65 }
66 /**
67  * @brief Recursively creates a
   *   directory path.
68  */
69 static void ensure_dir_exists(const char
   * path) {
70     char* p = strdup(path); // This will
   now be correctly declared
71     if(!p) return;
72     for (char* ptr = p + 1; *ptr; ++ptr)
73     {
74         if (*ptr == '/' || *ptr == '\\')
75         {
76             *ptr = '\\0';
77             MKDIR(p);
78             *ptr = (path == p + (ptr - p
   )) ? '/' : '\\';
79         }
80     }
81     MKDIR(p);
82     free(p);
83 }
84
85 #ifndef _WIN32
86 static int unlink_cb(const char *fpath,
   const struct stat *sb, int typeflag,
   struct FTW *ftwbuf) {
87     (void)sb; (void)typeflag; (void)
   ftwbuf;
88     return remove(fpath);
89 }
90
91 static void remove_directory_recursive(
   const char* path) {
92     nftw(path, unlink_cb, 64, FTW_DEPTH
   | FTW_PHYS);
93 }
94 #endif
95
96 //=====
97 // SECTION 2: PUBLIC API
98 // FUNCTION IMPLEMENTATION
99 //=====
100
101 /**
102  * @brief Executes the build manifest of
   *   a SenseObject.
103  * This is the main orchestration

```

```

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
107 * sense, const char* install_dir) {
108     printf("[BUILD_CONTROLLER] Starting_
109     build_for_sense: %s\n", sense->
110     sense_id);
111
112     if (!check_tools(sense)) {
113         return false;
114     }
115
116     char temp_dir_template[1024];
117     snprintf(temp_dir_template, sizeof(
118     temp_dir_template), "/tmp/agenk_
119     build_%s_XXXXXX", sense->sense_
120     id);
121
122     char* temp_dir = mkdtemp(temp_dir_
123     template);
124     if (!temp_dir) {
125         perror("[BUILD_ERROR] Could not_
126         create temporary directory")
127         ;
128         return false;
129     }
130     printf("[BUILD_CONTROLLER] Using_
131     temporary build directory: %s\n"
132     , temp_dir);
133
134     char command[2048];
135
136     snprintf(command, sizeof(command), "
137     git clone --depth 1 %s.", sense
138     ->dependency_source_uri);
139     if (run_command(command, temp_dir)
140     != 0) { // <-- FIX: Pass temp_
141     dir
142     fprintf(stderr, "[BUILD_ERROR]_
143     Failed to clone dependency.\n"
144     );
145     return false;
146 }
147
148 for (int i = 0; i < sense->num_
149 commands; ++i) {
150     if (run_command(sense->build_
151     commands[i], temp_dir) != 0)
152     { // <-- FIX: Pass temp_dir
153     fprintf(stderr, "[BUILD_
154     ERROR] Build command_
155     failed: %s\n", sense->
156     build_commands[i]);
157     return false;
158     }
159 }
160
161 char source_lib_path[1024];
162 char dest_lib_path[1024];
163
164 snprintf(source_lib_path, sizeof(
165 source_lib_path), "%s/%s", temp_
166 dir, sense->library_path_

```

```

167 template);
168 snprintf(dest_lib_path, sizeof(dest_
169 lib_path), "%s/%s", install_dir,
170 sense->installed_library_name);
171
172 MKDIR(install_dir);
173
174 snprintf(command, sizeof(command), "
175 cp %s %s", source_lib_
176 path, dest_lib_path);
177 if (run_command(command, ".") != 0)
178 { // <-- FIX: Pass current
179 directory
180     fprintf(stderr, "[BUILD_ERROR]_
181     Failed to copy artifact.\n")
182     ;
183     return false;
184 }
185 printf("[BUILD_CONTROLLER] Artifact_
186 installed to: %s\n", dest_lib_
187 path);
188
189 printf("[BUILD_CONTROLLER] Cleaning_
190 up temporary build directory...\n"
191 );
192 #ifndef WIN32
193     remove_directory_recursive(temp_dir)
194     ;
195 #else
196     snprintf(command, sizeof(command), "
197     rd /s /q %s", temp_dir);
198     run_command(command, "."); // <--
199     FIX: Pass current directory
200 #endif
201
202 printf("[BUILD_CONTROLLER] Build for_
203 %s completed_
204 successfully.\n", sense->sense_
205 id);
206 return true;
207 }

```

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  *         Aucta Sapience
30  */
31
32 #if defined(__linux__) || defined(__APPLE__) || defined(__FreeBSD__)
33 #define _POSIX_C_SOURCE 200809L
34 #endif
35
36 #include "../include/agenk_sensation.h"
37 #include "../private/sense_private.h"
38 #include "../tensors/include/agenk_tensor.h"
39
40 #include <stdio.h>
41 #include <stdlib.h>
42 #include <string.h>
43 #include <libtcc.h>
44
45 #if defined(_WIN32) || defined(_WIN64)
46 #include <windows.h>
47 typedef HMODULE DLibHandle;
48 #define DL_OPEN(path) LoadLibrary(
49     path)
50 #define DL_SYM(handle, sym)
51     GetProcAddress(handle, sym)
52 #define DL_CLOSE(handle) FreeLibrary(
53     handle)
54 #else
55 #include <dlfcn.h>
56 typedef void* DLibHandle;
57 #define DL_OPEN(path) dlopen(path,
58     RTLD_LAZY)
59 #define DL_SYM(handle, sym) dlsym(
60     handle, sym)
61 #define DL_CLOSE(handle) dlclose(
62     handle)
63 #endif
64
65 bool build_sense_organ(const SenseObject
66     * sense, const char* install_dir);
67 //=====
68
69 // SECTION 1: INTERNAL
70 // HELPER FUNCTIONS
71 //=====
72
73 static char** duplicate_string_array(
74     const char** src, int count) {
75     if (!src || count == 0) return NULL;
76     char** dst = (char**)malloc(count *
77         sizeof(char*));
78     if (!dst) return NULL;
79     for (int i = 0; i < count; ++i) {
80         dst[i] = strdup(src[i]);
81         if (!dst[i]) {
82             for (int j = 0; j < i; ++j)
83                 free(dst[j]);
84             free(dst);
85             return NULL;
86         }
87     }
88     return dst;
89 }
90
91 static const char* get_file_extension(
92     const char* uri) {
93     const char *dot = strrchr(uri, '.');
94     if (!dot || dot == uri) return "";
95     return dot + 1;
96 }
97
98 static void free_sense_object(
99     SenseObject* sense) {
100     if (!sense) return;
101     free(sense->sense_id);
102     free(sense->source_code);
103     free(sense->dependency_source_uri);
104     free(sense->library_path_template);
105     free(sense->installed_library_name);
106     // Free the new field
107
108     if (sense->supported_extensions) {
109         for (int i = 0; i < sense->num_
110             extensions; ++i) free(sense
111                 ->supported_extensions[i]);
112         free(sense->supported_extensions
113             );
114     }
115     if (sense->required_build_tools) {
116         for (int i = 0; i < sense->num_
117             tools; ++i) free(sense->
118                 required_build_tools[i]);
119         free(sense->required_build_tools
120             );
121     }
122     if (sense->build_commands) {
123         for (int i = 0; i < sense->num_
124             commands; ++i) free(sense->
125                 build_commands[i]);
126         free(sense->build_commands);
127     }
128     free(sense);
129 }
130
131 static SenseObject* get_sense_object_for
132     _extension(const char* extension) {
133     if (strcmp(extension, "txt") == 0) {
134         // Text loader definition
135         remains the same
136         SenseObject* sense = (
137             SenseObject*)calloc(1,
138                 sizeof(SenseObject));
139     }
140 }

```



```

99     if (!sense) return NULL;
100     sense->sense_id = strdup("sense_
        text_loader_v1");
101     const char* script =
102         "#include<stddef.h>\n#
            include<stdio.h>\n#
            include<stdlib.h>\n"
103         "#include<string.h>\n#
            include<agenk_tensor.h
            >\n"
104         "struct AgenkTensor_Internal
            {int dt; size_t es
            ; void* data;};\n"
105         "AgenkTensor* process(const
            char* path)\n"
106         "{
            FILE* f = fopen(path,
            \"rb\"); if (!f) return
            0;\n"
107         "fseek(f, 0, SEEK_END);
            long size = ftell(f);
            fseek(f, 0, 0);\n"
108         "if (size < 0) fclose(
            f); return 0;\n"
109         "size_t shape[] = {size
            _t};\n"
110         "AgenkTensor* t = tensor
            _create(shape, 1, 2, 4)
            ;\n"
111         "if (!t) fclose(f);
            return 0;\n"
112         "if (size > 0 && fread
            (((struct AgenkTensor_
            Internal*)t)->data, 1,
            size, f) != (size_t)size
            ) fclose(f); tensor_
            free(t); return 0;\n"
113         "fclose(f);\n"
114         "return t;\n"
115         "}";
116     sense->source_code = strdup(
        script);
117     return sense;
118 }
119 if (strcmp(extension, "jpg") == 0 ||
        strcmp(extension, "jpeg") == 0)
120 {
121     SenseObject* sense = (
        SenseObject*)calloc(1,
        sizeof(SenseObject));
122     if (!sense) return NULL;
123     sense->sense_id = strdup("sense_
        jpeg_turbo_v1");
124     sense->num_extensions = 2;
125     sense->supported_extensions =
        duplicate_string_array((
        const char*[]){ "jpg", "jpeg"
        }, sense->num_extensions);
126
127     sense->num_tools = 3;
128     sense->required_build_tools =
        duplicate_string_array((
        const char*[]){ "git", "cmake
        ", "make"}, sense->num_tools
        );
129     sense->dependency_source_uri =
        strdup("https://github.com/
        libjpeg-turbo/libjpeg-turbo.
        git");
130
131     sense->num_commands = 2;
132     sense->build_commands =
        duplicate_string_array((
        const char*[]){
133         "cmake -B build -DENABLE_
            SHARED=ON -DENABLE_
            STATIC=OFF",
134         "cmake --build build --
            config Release"
135     }, sense->num_commands);
136
137     // --- THE FIX: Define the
        contract ---
138     #if defined(_WIN32) || defined(_
        WIN64)
139         sense->library_path_template
            = strdup("build/jpeg62.
            dll");
140         sense->installed_library_
            name = strdup("libjpeg.
            dll");
141     #elif defined(__APPLE__)
142         sense->library_path_template
            = strdup("build/libjpeg
            .62.dylib");
143         sense->installed_library_
            name = strdup("libjpeg.
            dylib");
144     #else
145         sense->library_path_template
            = strdup("build/libjpeg
            .so.62");
146         sense->installed_library_
            name = strdup("libjpeg.
            so");
147     #endif
148
149     sense->source_code = strdup(
        "#include<agenk_tensor.h>\n
        #include<stdio.h>\n#
        include<stdlib.h>\n"
        "struct jpeg_decompress_
        struct; struct jpeg_
        error_mgr;\n"
        "//... (extern function
        declarations remain the
        same)...\n"
        "AgenkTensor* process(const
        char* path)\n"
        "{
        return
        NULL; /* STUBBED */\n"
        });
150
151     if (!sense->supported_extensions
        || !sense->required_build_
        tools || !sense->build_
        commands) {
152         free_sense_object(sense);
153         return NULL;
154     }
155     return sense;
156 }
157 return NULL;
158 }
159
160 //=====
161 // SECTION 2: PUBLIC API
162 // IMPLEMENTATION
163 //=====
164 AgenkTensor* perceive_input(const char*
        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'. Attempting to build...\n", library_full_path);
194         if (!build_sense_organsense, install_dir)) {
195             fprintf(stderr, "[SENSE_MANAGER] ERROR: Failed to build required library for sense '%s'.\n", sense->sense_id);
196             free_sense_object(sense);
197             return NULL;
198         }
199         printf("[SENSE_MANAGER] Build successful. Proceeding with perception.\n");
200     }
201 }
202
203 AgenkTensor* result_tensor = NULL;
204 TCCState* tcc_state = tcc_new();
205 if (!tcc_state) {
206     free_sense_object(sense);
207     return NULL;
208 }
209
210 char tcc_lib_path[1024];
211 snprintf(tcc_lib_path, sizeof(tcc_lib_path), "%s/deps/tcc", AGENK_SOURCE_DIR);
212 tcc_set_lib_path(tcc_state, tcc_lib_path);
213
214 tcc_set_output_type(tcc_state, TCC_OUTPUT_MEMORY);
215
216 char include_path[1024];
217 snprintf(include_path, sizeof(include_path), "%s/src/tensors/include", AGENK_SOURCE_DIR);
218 tcc_add_include_path(tcc_state, include_path);
219
220 tcc_add_library_path(tcc_state, "lib");
221 tcc_add_library(tcc_state, "agenk_tensor");
222
223 tcc_add_symbol(tcc_state, "tensor_create", tensor_create);
224 tcc_add_symbol(tcc_state, "tensor_free", tensor_free);
225
226 DLlibHandle lib_handle = NULL;
227 if (library_full_path[0] != '\0') {
228     lib_handle = DL_OPEN(library_full_path);
229     if (!lib_handle) {
230         fprintf(stderr, "[SENSE_MANAGER] ERROR: Failed to dynamically load freshly built library: %s\n", library_full_path);
231         #ifndef WIN32
232         fprintf(stderr, "dlopen error: %s\n", dlerror());
233         #endif
234         tcc_delete(tcc_state);
235         free_sense_object(sense);
236         return NULL;
237     }
238 }
239
240 if (tcc_compile_string(tcc_state, sense->source_code) != 0) {
241     fprintf(stderr, "[SENSE_MANAGER] ERROR: Failed to compile JIT driver for %s.\n", sense->sense_id);
242 } else {
243     if (tcc_relocate(tcc_state, TCC_RELOCATE_AUTO) != 0) {
244         fprintf(stderr, "[SENSE_MANAGER] ERROR: TCC relocation failed.\n");
245     } else {
246         AgenkTensor* (*process_func)(const char*);
247         process_func = tcc_get_symbol(tcc_state, "process");
248     }
249 }

```

```

249         if (process_func) {
250             result_tensor = process_
                func(uri);
251         } else {
252             fprintf(stderr, "[SENSE_
                MANAGER_ERROR]_Could
                _not_find_'process'_
                function.\n");
253         }
254     }
255 }
256
257 if (lib_handle) {
258     DL_CLOSE(lib_handle);
259 }
260 tcc_delete(tcc_state);
261 free_sense_object(sense);
262
263 if(result_tensor) {
264     printf("[SENSE_MANAGER]_Perception
        _successful.\n");
265 }
266
267 return result_tensor;
268 }

```

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 `
16 * process(const char* path)` which
17 * must
18 * return a pointer to a newly allocated
19 * AgenkTensor on success, or NULL on
20 * failure.
21 *
22 * It relies on the Sense Manager to
23 * link it against the `agenk_tensor`
24 * library
25 * so it can call functions like tensor_
26 * create().
27 *
28 * @version 1.0.0
29 * @author Ankit Yadav, Ankush Yadav,
30 * AuctaSapience
31 */
32
33 // We expect the Sense Manager to
34 // provide this include path at JIT
35 // compile time.
36 #include <agenk_tensor.h>
37
38 #include <stdio.h>
39 #include <stdlib.h>
40
41 // This is the required entry point for
42 // the JIT driver.
43 AgenkTensor* process(const char* path) {
44     if (!path) {
45         // A real implementation might
46         // have a way to log errors
47         // back to the host.
48         // For now, returning NULL is
49         // the signal of failure.
50         return NULL;
51     }
52
53     FILE* f = fopen(path, "rb");
54     if (!f) {
55         return NULL;
56     }
57
58     // Determine the size of the file
59     fseek(f, 0, SEEK_END);
60     long size = ftell(f);
61     fseek(f, 0, SEEK_SET);
62
63     if (size <= 0) {
64         fclose(f);
65         // It's not an error to process
66         // an empty file, return an
67         // empty tensor.
68         size_t shape[] = {0};
69         return tensor_create(shape, 1,

```

```

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

```

Listing 55: An example JIT-compileable 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
    Sensation Subsystem.
4
5   * This file validates the end-to-end
    functionality of the Sense Manager,
6   * including its ability to find
    blueprints, execute JIT drivers,
    and
7   * correctly return materialized tensors
8   *
9   * @version 1.0.0
10  * @author Ankush Yadav, Ankit Yadav,
    AuctaSapience
11  */
12
13  #include "../include/agenk_sensation.h"
14  #include "../../tensors/include/agenk_
    tensor.h" // Now we only need the
    public header
15  #include <stdio.h>
16  #include <stdlib.h>
17  #include <string.h>
18
19  // --- Simple Testing Framework ---
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("
    --- Running Test Suite: %s ---\n",
    name)
26  #define TEST_CASE(name) printf("\n[TEST]
    %s\n", name)
27  #define ASSERT(condition)
    \

```

```

28     do {
29         \
30         g_tests_run++;
31
32         \
33         if (condition) {
34             \
35             g_tests_passed++;
36
37         } else {
38             \
39             fprintf(stderr, COLOR_RED "[
40             FAIL]_s:%d:_Assertion_
41             failed:_s\n" \
42             COLOR_RESET, _FILE_
43             , _LINE_, #
44             condition); \
45
46         }
47     } while (0)
48
49 // --- Test Suite ---
50
51 void test_suite_sense_manager() {
52     TEST_SUITE_START("Sense_Manager");
53     TEST_CASE("Perceiving_a_simple_text_
54     file");
55
56     const char* test_filename = "test_
57     input.txt";
58     const char* test_content = "Hello,_
59     world_of_senses!";
60     FILE* f = fopen(test_filename, "w");
61     if (!f) { ASSERT(0 && "Failed_to_
62     create_test_file"); return; }
63     fputs(test_content, f);
64     fclose(f);
65
66     AgenkTensor* result = perceive_input
67     (test_filename, "test_stream");
68
69     ASSERT(result != NULL);
70     if (result) {
71         ASSERT(tensor_get_ndim(result)
72         == 1);
73         ASSERT(tensor_get_data_type(
74         result) == AGENK_DATA_TYPE_
75         UINT8);
76         ASSERT(tensor_get_content_type(
77         result) == AGENK_CONTENT_
78         TYPE_UTF8_TEXT);
79
80         const size_t* shape = tensor_get
81         _shape(result);
82         ASSERT(shape[0] == strlen(test_
83         content));
84
85         // THE FIX: Use the new, safe
86         public accessor function to
87         get the data.
88         const void* data = tensor_get_
89         data_ptr(result);
90         ASSERT(data != NULL);
91         if (data) {

```

```

66             ASSERT(memcmp(data, test_
67             content, strlen(test_
68             content)) == 0);
69
70         }
71
72         tensor_free(result);
73     }
74
75     remove(test_filename);
76 }
77
78 // --- Test Runner ---
79
80 int main() {
81     printf("=====\n");
82     printf("Running_AGENK_Sensation_
83     Subsystem_Tests\n");
84     printf("=====\n");
85
86     test_suite_sense_manager();
87
88     printf("\n=====\n");
89     printf("_TEST_SUMMARY_\n");
90     printf("=====\n");
91     if (g_tests_passed == g_tests_run) {
92         printf(COLOR_GREEN "SUCCESS:_All
93         _d_tests_passed.\n" COLOR_
94         RESET, g_tests_run);
95         return 0;
96     } else {
97         fprintf(stderr, COLOR_RED "
98         FAILURE:_d_out_of_d_tests_
99         failed.\n" COLOR_RESET,
100         g_tests_run - g_tests_
101         passed, g_tests_run);
102
103         return 1;
104     }
105 }

```

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
19 #define ASSERT(condition) do { if (!(
    condition)) { \
20     fprintf(stderr, "[FAIL]_s:%d:_
        Assertion_failed:_s\n", _FILE_
        _, _LINE_, #condition); \
21     exit(1); } } while (0)
22
23 #define INSTALL_DIR "runtime_libs_test"
24
25 int main() {
26     printf("=====\n")
27     ;
28     printf("Running_AGENK_Autonomous_
        Build_Test\n");
29     printf("=====\n")
30     ;
31     system("rm -rf _INSTALL_DIR");
32     printf("[TEST]_Cleaned_test_
        installation_directory.\n");
33
34     const char* test_filename = "dummy.
        jpg";
35     FILE* f = fopen(test_filename, "w");
36     if (f) {
37         fputs("dummy", f);
38         fclose(f);
39     }
40     printf("[TEST]_Created_dummy_file:_
        s\n", test_filename);
41
42     printf("\n[TEST]_Calling_perceive_
        input,_expecting_build_to_
        trigger...\n");
43     AgenkTensor* result = perceive_input
        (test_filename, "build_test_
        stream");
44
45     printf("\n[TEST]_Validating_build_
        results...\n");
46
47     // --- THE FIX: Check for the
        correct shared library file ---
48     const char* expected_lib_path;
49     #if defined(_WIN32) || defined(_
        WIN64)
50         expected_lib_path = INSTALL_DIR
51         "/libjpeg.dll";
52     #elif defined(_APPLE_)
53         expected_lib_path = INSTALL_DIR
54         "/libjpeg.dylib";
55     #else
56         expected_lib_path = INSTALL_DIR
57         "/libjpeg.so";
58     #endif
59
60     FILE* lib_file = fopen(expected_lib_
        path, "rb");
61     ASSERT(lib_file != NULL);
62     if (lib_file) {
63         printf("[TEST]_SUCCESS:_Library_
        artifact_found_at_s\n",
        expected_lib_path);
64         fclose(lib_file);
65
66     }
67
68     ASSERT(result == NULL); // JIT
        script is still stubbed
69     tensor_free(result);

```

```

65
66     printf("[TEST]_Cleaning_up_artifacts
        ...\n");
67     remove(test_filename);
68     system("rm -rf _INSTALL_DIR");
69
70     printf("\n[SUCCESS]_Autonomous_Build
        _Test_Passed.\n");
71     return 0;
72 }

```

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

```

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
6  * sense_registry module. It validates
        the entire lifecycle of a
        SenseObject
7  * blueprint, ensuring the system's long
        -term memory for skills is robust,
8  * reliable, and correct.
9
10 * The test performs the following
        critical sequence of operations:
11 * 1. Initialization: It starts and
        cleans the database environment.
12 * 2. Creation: It programmatically
        constructs a complex, multi-valued
        `SenseObject` blueprint in memory
        using only the public API
        functions.
13 * This validates the blueprint
        builder API.
14 * 3. Serialization & Persistence:
        It calls `sense_registry_add_
        blueprint()`
15 * to serialize the in-memory C
        struct into the FlatBuffers format
        and
16 * persist it to the LMDB database.
17 * 4. Retrieval & Deserialization:
        It queries the database using the
        blueprint's unique ID to retrieve
        the raw FlatBuffers data and
        deserialize
18 * it back into a new in-memory `
        SenseObject` C struct.
19 * 5. Symmetrical Verification: It
        performs a deep, field-by-field
        comparison
20 * 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
21 * symmetrical and lossless.
22 * 6. Memory Management: It
        meticulously cleans up all
        allocated memory for
        both blueprint objects and shuts
        down the database, ensuring there
23
24
25
26
27

```

```

    are
    * no memory leaks.
    *
    * Passing this test provides high
    * confidence in the foundational
    * layer of the
    * agent's ability to learn and remember
    * new perceptual skills.
    *
    * @version 1.0.0
    * @author Ankush Yadav, Ankit Yadav,
    *   AuctaSapience
    */

#include "../include/agenk_sensation.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define ASSERT(c) do { if(!(c)) {
    fprintf(stderr, "[FAIL]_s:%d:_s\n",
    ,__FILE__, __LINE__, #c); exit(1); } }
    while(0)
#define DB_PATH_REGISTRY_TEST "../test_
    registry_db"

int main() {
    printf("---_Running_Sense_Registry_
    Tests_(Full_Lifecycle)_---\n");

    system("rm -rf" DB_PATH_REGISTRY_
    TEST);
    ASSERT(sense_registry_init(DB_PATH_
    REGISTRY_TEST));

    // 1. Create a complete blueprint
    // programmatically
    printf("[TEST]_Creating_'sense_jpeg_
    v1'_blueprint_in_memory...\n");
    SenseObject* sense_out = sense_
    object_create("sense_jpeg_v1");
    ASSERT(sense_out != NULL);
    sense_object_set_source_uri(sense_
    out, "https://github.com/libjpeg
    -turbo/libjpeg-turbo.git");
    sense_object_set_library_template(
    sense_out, "build/libjpeg.so.62"
    );
    sense_object_set_installed_name(
    sense_out, "libjpeg.so");
    sense_object_set_jit_driver(sense_
    out, "return_process_jpeg(path);
    ");
    sense_object_add_build_command(sense_
    out, "cmake -B build");
    sense_object_add_build_command(sense_
    out, "make install");
    sense_object_add_supported_extension
    (sense_out, "jpg");
    sense_object_add_supported_extension
    (sense_out, "jpeg");
    sense_object_add_build_tool(sense_
    out, "cmake");
    sense_object_add_build_tool(sense_
    out, "make");

    // 2. Add it to the database
    printf("[TEST]_Adding_blueprint_to_
    the_database...\n");
    ASSERT(sense_registry_add_blueprint(

```

```

    sense_out));

    // 3. Retrieve the full blueprint by
    // its ID
    printf("[TEST]_Retrieving_blueprint_
    by_ID_from_the_database...\n");
    SenseObject* sense_in = sense_
    registry_get_by_id("sense_jpeg_
    v1");
    ASSERT(sense_in != NULL);

    // 4. Deep, symmetrical comparison
    // using ONLY the public getter API
    printf("[TEST]_Performing_deep_
    comparison...\n");
    ASSERT(strcmp(sense_object_get_id(
    sense_out), sense_object_get_id(
    sense_in)) == 0);
    ASSERT(strcmp(sense_object_get_
    source_uri(sense_out), sense_
    object_get_source_uri(sense_in))
    == 0);
    ASSERT(strcmp(sense_object_get_
    installed_name(sense_out), sense_
    object_get_installed_name(sense_
    in)) == 0);

    ASSERT(sense_object_get_num_
    extensions(sense_out) == sense_
    object_get_num_extensions(sense_
    in));
    ASSERT(strcmp(sense_object_get_
    extension(sense_in, 0), "jpg")
    == 0);

    ASSERT(sense_object_get_num_build_
    tools(sense_out) == sense_object_
    get_num_build_tools(sense_in));
    ASSERT(strcmp(sense_object_get_build_
    tool(sense_in, 0), "cmake") ==
    0);

    ASSERT(sense_object_get_num_build_
    commands(sense_out) == sense_
    object_get_num_build_commands(
    sense_in));
    ASSERT(strcmp(sense_object_get_build_
    command(sense_in, 1), "make_
    install") == 0);

    // 5. Cleanup
    printf("[TEST]_Cleaning_up...\n");
    sense_object_free(sense_out);
    sense_object_free(sense_in);

    sense_registry_shutdown();
    system("rm -rf" DB_PATH_REGISTRY_
    TEST);

    printf("\n[SUCCESS]_Sense_Registry_
    Test_Passed.\n");
    return 0;
}

```

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 different 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 Per As Our Current Understanding, We Have Two Problems -

1. Representation Of Rectangles In Hierarchical 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`, `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 `set_name` 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.

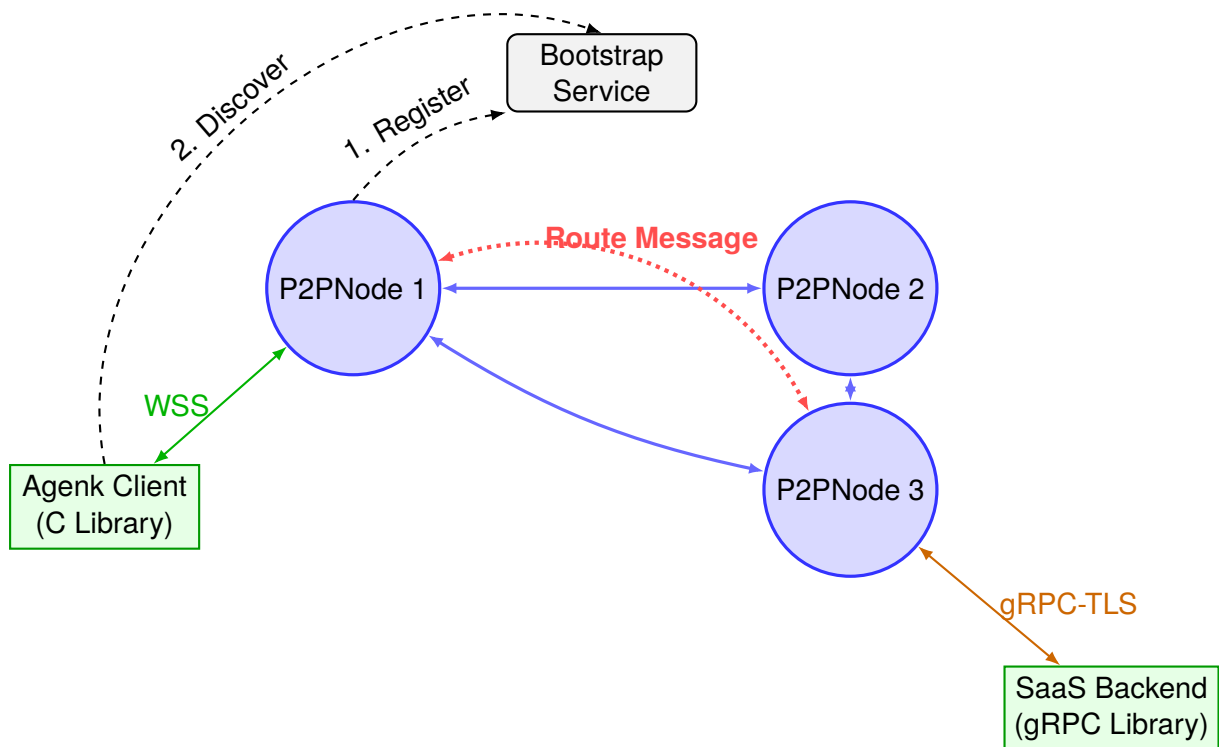


Figure 3: High-level architecture of the Network Fabric. P2P Nodes form a resilient mesh for internal communication. Clients discover and connect to a gateway node. A message from the SaaS Backend to an Agenk Client is routed through the P2P mesh from the entry gateway (Node 3) to the destination gateway (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 httpLib
8   header at configure time
9 include(FetchContent)
10 FetchContent_Declare(
11   httpLib
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   httpLib # 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 "httpLib.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;

```

```

16 };
17
18 // --- Global State (Thread-Safe) ---
19 std::vector<NodeInfo> active_nodes;
20 std::mutex nodes_mutex;
21 const int STALE_NODE_TIMEOUT_SECONDS =
22   90;
23
24 // Background thread to remove stale
25   nodes
26 void garbage_collector() {
27   while (true) {
28     std::this_thread::sleep_for(std::
29       chrono::seconds(30));
30     std::lock_guard<std::mutex> lock
31       (nodes_mutex);
32     auto now = std::chrono::steady_
33       clock::now();
34
35     active_nodes.erase(
36       std::remove_if(active_nodes.
37         begin(), active_nodes.
38         end(),
39         [&](const NodeInfo& node
40           ) {
41             auto elapsed = std::
42               chrono::duration
43                 _cast<std::
44                   chrono::seconds
45                 >(now - node.
46                   last_seen).count
47                   ());
48             if (elapsed > STALE_
49               NODE_TIMEOUT_
50                 SECONDS) {
51               std::cout << "[
52                 GC] Removing
53                 stale node
54                 " << node.
55                 address <<
56                 std::endl;
57               return true;
58             }
59             return false;
60           }
61         ),
62         active_nodes.end()
63       );
64   }
65 }
66
67 int main() {
68   httpLib::Server svr;
69   std::thread gc_thread(garbage_
70     collector);
71   gc_thread.detach();
72
73   svr.Get("/nodes", [](const httpLib::
74     Request&, httpLib::Response& res
75     ) {
76     std::vector<std::string> nodes_
77       to_send;
78     {
79       std::lock_guard<std::mutex>
80         lock(nodes_mutex);
81       for(const auto& node :
82         active_nodes) {
83         nodes_to_send.push_back(
84           node.address);
85       }
86     }
87   }
88 }

```

```

57 // Randomize the list to
58 // distribute client load
59 std::random_device rd;
60 std::mt19937 g(rd());
61 std::shuffle(nodes_to_send.begin(), nodes_to_send.end(), g);
62
63 std::string body = "[";
64 for (size_t i = 0; i < nodes_to_send.size(); ++i) {
65     body += "\"" + nodes_to_send[i] + "\"";
66     if (i < nodes_to_send.size() - 1) body += ",";
67 }
68 body += "]";
69
70 res.set_content(body, "application/json");
71 });
72
73 svr.Post("/register", [](const http::Request& req, http::Response& res) {
74     std::string address_key = "\"address\": \"";
75     auto pos = req.body.find(address_key);
76     if (pos == std::string::npos) {
77         res.status = 400;
78         res.set_content("Invalid JSON", "text/plain");
79         return;
80     }
81     auto start = pos + address_key.length();
82     auto end = req.body.find("\"", start);
83     if (end == std::string::npos) {
84         res.status = 400;
85         return;
86     }
87     std::string address = req.body.substr(start, end - start);
88
89     {
90         std::lock_guard<std::mutex> lock(nodes_mutex);
91         auto it = std::find_if(active_nodes.begin(), active_nodes.end(),
92             [&](const NodeInfo& node) { return node.address == address; });
93
94         if (it != active_nodes.end()) {
95             it->last_seen = std::chrono::steady_clock::now();
96         } else {
97             active_nodes.push_back({address, std::chrono::steady_clock::now()});
98         }
99     }
100
101     res.set_content("{\"status\": \"success\"}", "application/json");
102 });
103
104 int port = 9090;
105 std::cout << "Bootstrap server listening on port " << port << "\n";
106 svr.listen("0.0.0.0", port);
107 return 0;
108 }
109 #define CPPHTTPLIB_OPENSSL_SUPPORT // Enable this for future HTTPS support
110 #include "http.h"
111 #include <iostream>
112 #include <vector>
113 #include <string>
114 #include <mutex>
115 #include <chrono>
116 #include <thread>
117
118 // A simple struct to hold node info and its last check-in time
119 struct NodeInfo {
120     std::string address;
121     std::chrono::steady_clock::time_point last_seen;
122 };
123
124 // --- Global State (Thread-Safe) ---
125 std::vector<NodeInfo> active_nodes;
126 std::mutex nodes_mutex;
127 const int STALE_NODE_TIMEOUT_SECONDS = 90; // If a node hasn't checked in for 90s, remove it
128
129 // Background thread to remove stale nodes
130 void garbage_collector() {
131     while (true) {
132         std::this_thread::sleep_for(std::chrono::seconds(30));
133
134         std::lock_guard<std::mutex> lock(nodes_mutex);
135         auto now = std::chrono::steady_clock::now();
136
137         active_nodes.erase(
138             std::remove_if(active_nodes.begin(), active_nodes.end(),
139                 [&](const NodeInfo& node) {
140                     auto elapsed = std::chrono::duration_cast<std::chrono::seconds>(now - node.last_seen).count();
141                     if (elapsed > STALE_NODE_TIMEOUT_SECONDS) {
142                         std::cout << "Garbage Collector: Removing stale node " << node

```

```

142         address <<
143             std::endl;
144         return true;
145     },
146     return false;
147 },
148 active_nodes.end()
149 );
150 }
151 int main() {
152     httplib::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 httplib::
165         Request&, httplib::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 += ",";
181             }
182         }
183         body += "]";
184
185         res.set_content(body, "

```

```

186         res.set_content("Invalid_
187             JSON_body_Expected_{\"
188             address\": \"...\"}\", \"
189             text/plain");
190         return;
191     }
192
193     auto start = pos + address_key.
194         length();
195     auto end = req.body.find("\"",
196         start);
197     if (end == std::string::npos) {
198         res.status = 400;
199         return;
200     }
201
202     std::string address = req.body.
203         substr(start, end - start);
204
205     {
206         std::lock_guard<std::mutex>
207             lock(nodes_mutex);
208         // Check if node already
209         // exists and just update
210         // its timestamp
211         auto it = std::find_if(
212             active_nodes.begin(),
213             active_nodes.end(),
214             [&](const NodeInfo& node
215                 ) { return node.
216                     address == address;
217             });
218
219         if (it != active_nodes.end()
220             ) {
221             it->last_seen = std:::
222                 chrono::steady_clock
223                 ::now();
224         } else {
225             active_nodes.push_back({
226                 address, std::chrono
227                 ::steady_clock::now
228                 ()});
229         }
230     }
231
232     res.set_content("{\"status\": \"
233         success\"}\", \"application/
234         json\"");
235     std::cout << "POST_/register:_
236         Registered/updated_node_ <<
237         address << std::endl;
238     });
239
240     std::cout << "Bootstrap_server_
241         starting_on_port_9090..." << std
242         ::endl;
243     svr.listen("0.0.0.0", 9090);
244
245     return 0;
246 }

```

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         bootstrap_server..." << std
9         ::endl;
10        httpLib::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    }
28 }

```

```

19 void Node::registrationLoop() {
20     httpLib::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, "
27         application/json");
28         // Wait for 60 seconds before re
29         // -registering
30         for (int i = 0; i < 60 && !m_
31         stop_heartbeat; ++i) {
32             std::this_thread::sleep_for(
33             std::chrono::seconds(1))
34             ;
35         }
36     }
37 }

```

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

12.4 The Agenk 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 ‘p2p_client_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 {
20

```

```

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
23 if (client->wsi) {
24     lws_callback_on_writable(client
25         ->wsi);
26 }
27 return 0;
28 }

```

Listing 62: The thread-safe send queue from the C client library ('p2p_client.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 ??, implements 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 void P2PGrpcClientImpl::
4     listenerThreadMain() {
5     while (m_is_running) {
6         // Establish the persistent
7         // server stream
8         m_stream_context = std::make_
9         unique_ptr<grpc::ClientContext>
10         >();
11         p2p::ClientLogin request;
12         request.set_client_id(m_client_
13         id);
14
15         std::unique_ptr<grpc::
16         ClientReader<p2p::
17         ServerEnvelope>> stream =
18         m_stub->CreateSession(m_
19         stream_context.get(),
20         request);
21     }
22 }

```

```

13 if (m_on_status) m_on_status("
14     CONNECTED");
15
16 // Block here, waiting for the
17 // server to push messages
18 p2p::ServerEnvelope envelope;
19 while (m_is_running && stream->
20     Read(&envelope)) {
21     if (envelope.has_forwarded_
22     message()) {
23         const auto& msg =
24         envelope.forwarded_
25         message();
26         if (m_on_message) {
27             m_on_message(msg.
28                 sender_id(), msg
29                 .content());
30         }
31     }
32 }
33
34 // If the loop breaks, the
35 // connection was lost.
36 if (m_is_running) {
37     if (m_on_status) m_on_status
38     ("CONNECTION LOST");
39     std::this_thread::sleep_for(
40     std::chrono::seconds(5))
41     ;
42     if (m_on_status) m_on_status
43     ("RECONNECTING");
44 }
45 }

```

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 their 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, pdfL^AT_EX is strongly recommended (over original L^AT_EX plus dvips+ps2pdf or dvi2pdf). The style file `acl.sty` can also be used with luaL^AT_EX and XeL^AT_EX, 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^AT_EX or XeL^AT_EX.

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
<code>\a</code>	ä	<code>\c c</code>	ç
<code>\^e</code>	ê	<code>\u g</code>	ğ
<code>\`i</code>	ì	<code>\l</code>	ł
<code>\.I</code>	İ	<code>\~n</code>	ñ
<code>\o</code>	ø	<code>\H o</code>	ő
<code>\'u</code>	ú	<code>\v r</code>	ř
<code>\aa</code>	å	<code>\ss</code>	ß

Table 1: Example commands for accented characters, to be used in, *e.g.*, BibT_EX entries.

(Alternatives like txfonts or newtx are also acceptable.)

Please see the L^AT_EX 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 L^AT_EX 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.¹

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 L^AT_EX preamble (after the `\documentclass` declaration and before `\begin{document}`) using `\usepackage{graphicx}`.

¹This is a footnote.



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 \LaTeX may encounter the following error during compilation:

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

This happens when pdf\LaTeX is used and a citation splits across a page boundary. The best way to fix this is to upgrade \LaTeX 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 `\citeposs`. This is not a standard natbib command, so it is generally not compatible with other style files.

17.5 References

The \LaTeX and Bib \TeX 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 \LaTeX file will generate the references section for you:

```
\bibliography{custom}
```

You can obtain the complete ACL Anthology as a Bib \TeX 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 Bib \TeX files.

17.6 Equations

An example equation is shown below:

$$A = \pi r^2 \tag{4}$$

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 Bib \TeX Files

Unicode cannot be used in Bib \TeX entries, and some ways of typing special characters can disrupt Bib \TeX ’s alphabetization. The recommended way of typing special characters is shown in Table 1.

Please ensure that Bib \TeX 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 Bib \TeX 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` \LaTeX 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

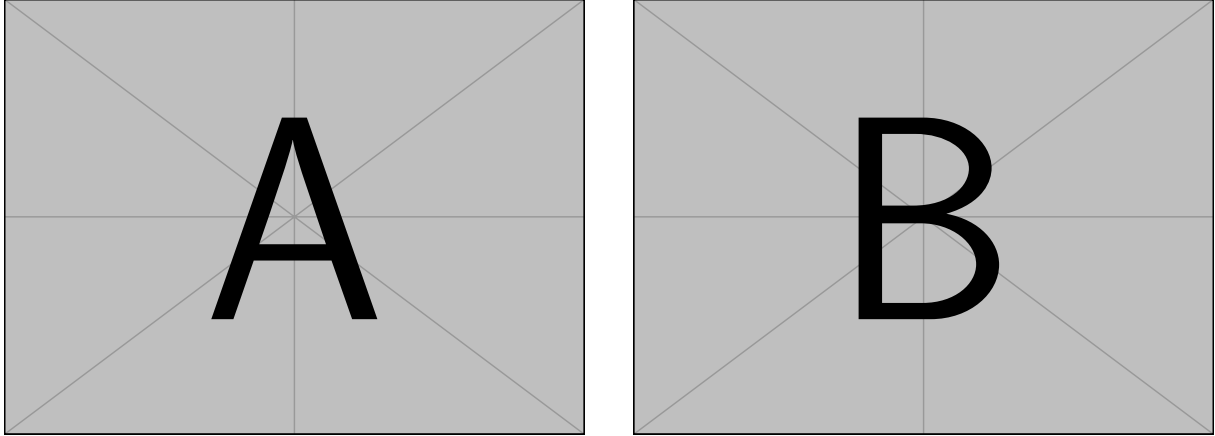


Figure 5: A minimal working example to demonstrate how to place two images side-by-side.

Output	natbib command	ACL only command
(?)	<code>\citep</code>	
?	<code>\citealp</code>	
?	<code>\citet</code>	
(?)	<code>\citeyearpar</code>	
?’s (?)		<code>\citeposs</code>

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, BibTeX 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.