Orb Plateau 1: Implementation Roadmap, Gap Analysis, and Completion Criteria

Executive Summary

Based on analysis of the architecture map (v0.13.6) and the Plateau 1 checklist, Orb has a solid foundation but requires focused work in **six areas** to reach a stable, predictable, recoverable state. The architecture map shows mature routing (5-type system), authentication, encryption, and multi-LLM orchestration—but the tool layer, audit logging, and several UI/memory edge cases need attention.

Key Finding: The architecture map documents a sophisticated high-stakes critique pipeline and routing system, but is notably silent on the **tool binding layer** mentioned in the checklist. This is the largest gap.

Part 1: Plateau 1 Implementation Roadmap

Phase Overview

Phase	Name	Duration	Dependencies
P1.1	Core LLM Loop Stabilisation	3-4 days	None

P1.2	Memory + UI Stabilisation	4-5 days	P1.1
P1.3	Tool Layer Restoration	3-4 days	P1.1
P1.4	Routing Refinements (Vision + Overrides)	2-3 days	P1.1, P1.3
P1.5	Audit Logging	2 days	P1.1
P1.6	Bug Sweep + Acceptance	2-3 days	All above

Total estimated time: 16-21 working days (3-4 weeks for solo dev)

Phase P1.1: Core LLM Loop Stabilisation

Goal: Make the core loop rock-solid: user prompt → router → chosen model → response → UI. No hangs, no double sends, no dropped messages.

Subsystems to Touch

Component	Location	Changes
Backend router	`app/llm/router.py`	Add request deduplication, timeout handling
Stream router	`app/llm/stream_router.py`	Add connection cleanup, retry logic
Provider registry	`app/providers/registry.py`	Add timeout config, simple retry with backoff
API client	`orb-desktop/src/services/api.ts`	Add request cancellation, retry with backoff

Chat interface	`orb-desktop/src/components/ChatInterface.tsx`	Add pending state, prevent double-submit

Order of Operations

1. Backend timeout handling (router.py)
   └── Add configurable timeout per provider (OpenAI: 60s, Anthropic: 120s, Gemini: 90s)
   └── Wrap all provider calls in asyncio.wait_for()
   └── Return structured error on timeout (not hang)

2. Request deduplication (main.py)
   └── Add request_id to all /chat and /stream/chat requests
   └── Track in-flight requests in memory dict (sufficient for single-process)
   └── Reject duplicate request_id within 5s window
   └── Note: This in-memory approach works for single-process; revisit if multi-worker needed later

3. Simple retry with exponential backoff (router.py)
   └── On transient errors (timeout, 503, 429), retry up to 3 times
   └── Backoff: 1s, 2s, 4s delays
   └── Do NOT implement full circuit breaker state machine—overkill for single-user app

4. Stream connection cleanup (stream_router.py)
   └── Add try/finally to ensure generator cleanup
   └── Handle client disconnect (asyncio.CancelledError)
   └── Log incomplete streams for debugging

5. Frontend double-submit prevention (ChatInterface.tsx)
   └── Disable send button while request pending
   └── Add isSubmitting state
   └── Clear on response OR timeout OR error

6. Frontend request cancellation (api.ts)
   └── Use AbortController for all fetch calls
   └── Cancel pending request on component unmount
   └── Cancel on new submit if previous still pending (optional)

Quick-Win Refactors

Centralise timeout and retry constants in `app/llm/config.py`:

```
PROVIDER_TIMEOUTS = {
    "openai": 60,
    "anthropic": 120,  # Opus can be slow
    "google": 90,
}

RETRY_CONFIG = {
    "max_retries": 3,
    "base_delay": 1.0,  # seconds
    "backoff_multiplier": 2.0,
}
```

1.

Add request tracking middleware in `main.py`:

```
# Simple in-memory tracking (single-process safe)
in_flight_requests: dict[str, float] = {}

@app.middleware("http")
async def track_requests(request: Request, call_next):
    request_id = request.headers.get("X-Request-ID", str(uuid4()))
    # ... dedup and tracking logic
```

2.

Deliverables

- No request hangs indefinitely (all timeout within 2min max)
- Double-click on send doesn't create duplicate messages
- Stream disconnects don't leave zombie processes
- Transient errors retry automatically with backoff
- Error responses include actionable error codes

Phase P1.2: Memory + UI Stabilisation

Goal: Messages and notes save/load correctly per project. Fix project switching, UI reloads, context never disappears or mixes.

Subsystems to Touch

Component	Location	Changes
Message persistence	`app/memory/`	Add transaction safety, conflict detection
Project switching	`orb-desktop/src/hooks/`	Add loading states, cache invalidation
Chat history	`orb-desktop/src/components/MessageList.tsx`	Fix scroll restoration, history loading
Drag-drop upload	`orb-desktop/src/components/FileUpload.tsx`	Fix drop zone, file validation
Markdown rendering	`orb-desktop/src/components/`	Fix code blocks, table rendering

Order of Operations

1. Backend message transaction safety
   └── Wrap message save in explicit transaction
   └── Add optimistic locking (version column) if concurrent edits possible
   └── Return saved message with server-generated ID

2. Project context isolation
   └── Add project_id validation to all /memory/* endpoints
   └── Reject requests where project_id doesn't match session

└── Clear frontend cache on project switch

3. Frontend project switching
   └── Add loading state during switch
   └── Cancel in-flight requests for old project (use AbortController)
   └── Clear message list before loading new project
   └── Scroll to bottom after load

4. Chat history persistence
   └── Verify messages table has all required columns
   └── Add index on (project_id, created_at) for fast history load
   └── Limit history load to last 100 messages (pagination later)

5. UI fixes
   └── Scroll: Add ref to message container, scrollIntoView on new message
   └── Smart scroll: Only auto-scroll if user is near bottom
   └── Drag-drop: Fix event.preventDefault() in all drag handlers
   └── Markdown: Verify remark-gfm config, add table CSS

Quick-Win Refactors

Add useProject hook that centralises project state:

```
const { currentProject, switchProject, isLoading, error } = useProject();
```

1.

Add message cache with invalidation:

```
const messageCache = new Map<string, Message[]>();
function invalidateProject(projectId: string) {
    messageCache.delete(projectId);
}
// Call invalidateProject() on every project switch
```

2.

Deliverables

- Project switch never shows messages from wrong project
- Messages persist across app restart
- Scroll position maintained during conversation (smart scroll)
- Drag-drop works for all supported file types
- arkdown tables and code blocks render correctly

Phase P1.3: Tool Layer Restoration

Goal: Restore and stabilise the tool layer (weather, HTTP fetch, search, etc.) so all models use Orb's tools instead of provider-side browsing.

Critical Note: The architecture map mentions `app/tools/` but doesn't document its contents. This phase requires discovery + implementation.

Subsystems to Touch

Component	Location	Changes
Tool registry	`app/tools/registry.py` (create if missing)	Central tool registration

Tool definitions	`app/tools/*.py`	Individual tool implementations
Tool binding	`app/llm/router.py`	Inject tools into LLM calls
Web search	`app/llm/web_search_router.py`	Integrate as tool, not separate endpoint
Provider clients	`app/llm/clients.py`	Add tool/function calling support

Order of Operations

1. Audit existing tool code
   └── List all files in app/tools/
   └── Identify working vs broken tools
   └── Document tool function signatures

2. Create tool registry (if missing)
   └── app/tools/registry.py
   └── Register tools by name with schema
   └── Support enable/disable per tool

3. Define core tools for Plateau 1
   └── web_search: Use existing web_search_router logic
   └── http_fetch: Simple GET/POST with timeout (10s)
   └── datetime: Current time/date (already in datetime context?)
   └── file_read: Read from project files (already in memory?)

4. Bind tools to LLM calls
   └── OpenAI: Use function_call / tools parameter
   └── Anthropic: Use tool_use blocks
   └── Gemini: Use function_declarations
   └── Add tool execution loop in router.py

5. Disable provider-side browsing
   └── OpenAI: Don't enable web browsing in API calls
   └── Gemini: Use only Orb's web_search tool
   └── Anthropic: No native browsing (already compliant)

6. Tool execution safety
   └── Timeout per tool (10s default)
   └── Sanitise tool outputs before injecting into context

└── Log tool calls (name only) in audit log (Phase P1.5)

Tool Schema Example

```
# app/tools/registry.py
from dataclasses import dataclass
from typing import Callable, Any

@dataclass
class Tool:
    name: str
    description: str
    parameters: dict  # JSON Schema
    handler: Callable[..., Any]
    timeout: int = 10

TOOL_REGISTRY: dict[str, Tool] = {}

def register_tool(tool: Tool):
    TOOL_REGISTRY[tool.name] = tool

def get_tools_for_provider(provider: str) -> list[dict]:
    """Format tools for specific provider's API."""
    if provider == "openai":
        return [_format_openai_tool(t) for t in TOOL_REGISTRY.values()]
    elif provider == "anthropic":
        return [_format_anthropic_tool(t) for t in TOOL_REGISTRY.values()]
    elif provider == "google":
        return [_format_gemini_tool(t) for t in TOOL_REGISTRY.values()]
```

Deliverables

- `web_search` tool works through Orb (not provider browsing)
- `http_fetch` tool can GET public URLs with 10s timeout
- Tools bound to all three providers
- Tool calls logged (name only, prep for P1.5)
- No provider-side browsing enabled in any API calls

Phase P1.4: Routing Refinements (Vision + Overrides)

Goal: Complete vision routing for multi-image and video; add explicit text overrides.

Subsystems to Touch

Component	Location	Changes
Job classifier	`app/llm/job_classifier.py`	Add multi-image detection, override parsing
Vision routing	`app/llm/gemini_vision.py`	Handle multiple images
Router	`app/llm/router.py`	Add override stripping
Main endpoint	`main.py`	Pass attachment count to classifier

Order of Operations

1. Multi-image detection (job_classifier.py)
   └── Count images in attachments
   └── 1 image → SIMPLE_VISION (gemini-2.0-flash)
   └── 2+ images → HEAVY_MULTIMODAL_CRITIQUE (gemini-2.5-pro)

2. Video routing refinement
   └── Already have: >10MB → gemini-2.5-pro
   └── Add: Multiple videos → gemini-3.0-pro-preview
   └── Add: Video + deep analysis keywords → gemini-3.0-pro-preview

3. Override text parsing (job_classifier.py)
   └── Detect patterns: "OVERRIDE SEND TO <MODEL>"

└── Supported models: GEMINI_3_PRO, OPUS, SONNET, GPT
└── Strip override text from message before LLM call
└── Return override in RoutingDecision

4. Override application (router.py)
└── Check for override in RoutingDecision
└── If present, use override model regardless of classification
└── Log override usage in debug output

5. Vision endpoint updates (main.py)
└── Pass image_count to classifier
└── Handle multi-image upload to Gemini

Override Pattern Specification

```
import re
from typing import Optional

# Supported override patterns (case-insensitive)
OVERRIDE_PATTERNS = {
    r"OVERRIDE\s+SEND\s+TO\s+GEMINI\s*3\s*PRO": ("google",
"gemini-3.0-pro-preview"),
    r"OVERRIDE\s+SEND\s+TO\s+OPUS": ("anthropic",
"claude-opus-4-5-20251101"),
    r"OVERRIDE\s+SEND\s+TO\s+SONNET": ("anthropic",
"claude-sonnet-4-5-20250929"),
    r"OVERRIDE\s+SEND\s+TO\s+GPT": ("openai", "gpt-4.1-mini"),
    r"FORCE\s+OPUS": ("anthropic", "claude-opus-4-5-20251101"),  #
Existing
    r"USE\s+GPT": ("openai", "gpt-4.1-mini"),  # Existing
}

def detect_and_strip_override(message: str) -> tuple[str,
Optional[tuple[str, str]]]:
    """Returns (cleaned_message, optional (provider, model) override)."""
    for pattern, target in OVERRIDE_PATTERNS.items():
        match = re.search(pattern, message, re.IGNORECASE)
        if match:
            cleaned = re.sub(pattern, "", message,
flags=re.IGNORECASE).strip()
            # Clean up any resulting double spaces or leading colons
            cleaned = re.sub(r"^\s*:\s*", "", cleaned)
```

```
            cleaned = re.sub(r"\s+", " ", cleaned).strip()
            return (cleaned, target)
    return (message, None)
```

Deliverables

- 2+ images → routes to Gemini 2.5 Pro
- Multiple videos → routes to Gemini 3 Pro
- `OVERRIDE SEND TO GEMINI 3 PRO` forces routing
- Override text stripped from prompt sent to model
- Override logged in debug output

Phase P1.5: Audit Logging

Goal: Add audit_logs table. Log model, job type, success/failure, rate-limit info. No raw prompts or secrets.

Subsystems to Touch

Component	Location	Changes
Database schema	`app/db.py`	Add audit_logs table
Audit service	`app/audit/service.py` (new)	Logging functions
Router integration	`app/llm/router.py`	Log after each LLM call
Tool integration	`app/tools/registry.py`	Log tool executions

API endpoint	`app/audit/router.py` (new)	Query audit logs

Schema Design

```
# app/audit/models.py
from sqlalchemy import Column, Integer, String, Boolean, DateTime, JSON,
ForeignKey
from sqlalchemy.ext.declarative import declarative_base
from datetime import datetime

Base = declarative_base()

class AuditLog(Base):
    __tablename__ = "audit_logs"

    id = Column(Integer, primary_key=True)
    timestamp = Column(DateTime, default=datetime.utcnow, index=True)

    # Request identification
    request_id = Column(String(36), index=True)  # UUID
    project_id = Column(Integer, ForeignKey("projects.id"), nullable=True)

    # Routing info
    job_type = Column(String(50))  # e.g., "SMALL_CODE",
"BIG_ARCHITECTURE"
    provider = Column(String(20))  # e.g., "openai", "anthropic", "google"
    model = Column(String(50))     # e.g., "claude-sonnet-4-5-20250929"

    # Outcome
    success = Column(Boolean)
    error_code = Column(String(50), nullable=True)
    error_message = Column(String(500), nullable=True)  # Truncated, no
sensitive data

    # Performance
    latency_ms = Column(Integer)
    input_tokens = Column(Integer, nullable=True)
    output_tokens = Column(Integer, nullable=True)

    # Rate limiting (when available from provider headers)
    rate_limit_remaining = Column(Integer, nullable=True)
    rate_limit_reset = Column(DateTime, nullable=True)
```

```
    # Tool usage (names only, not inputs/outputs)
    tools_called = Column(JSON, nullable=True)  # e.g., ["web_search",
"http_fetch"]

    # Critique pipeline
    critique_triggered = Column(Boolean, default=False)
    critique_success = Column(Boolean, nullable=True)

    # Override tracking
    override_used = Column(Boolean, default=False)
    override_target = Column(String(50), nullable=True)  # e.g., "opus"
```

Order of Operations

1. Create audit module structure
   └── app/audit/__init__.py
   └── app/audit/models.py (ORM model)
   └── app/audit/service.py (log_llm_call, log_tool_call)
   └── app/audit/router.py (GET /audit/logs endpoint)

2. Add migration for audit_logs table
   └── scripts/add_audit_logs.py
   └── Run migration on local SQLite DB

3. Integrate into router.py
   └── Wrap LLM calls with timing (start_time = time.time())
   └── Extract rate limit headers from responses where available
   └── Call audit_service.log_llm_call() after each call

4. Integrate into tool registry
   └── Log tool name, success, latency
   └── Do NOT log tool inputs/outputs (may contain secrets or large data)

5. Add query endpoint
   └── GET /audit/logs?project_id=&start_date=&end_date=&provider=&success=
   └── Paginated response (default 50, max 200)
   └── Filter by success/failure, provider, job_type

What NOT to Log (Privacy & Size)

- ❌ Raw prompts (privacy, size)
- ❌ Raw responses (privacy, size)
- ❌ API keys (obviously)
- ❌ File contents
- ❌ User passwords or tokens
- ❌ Tool input parameters (may contain sensitive data)
- ❌ Tool output data (may be large)

Deliverables

- audit_logs table exists with correct schema
- Every LLM call creates audit log entry
- Tool calls logged (name only, not inputs/outputs)
- Rate limit info captured when available from provider headers
- `/audit/logs` endpoint returns paginated results
- No sensitive data in audit logs (verified by inspection)

Phase P1.6: Bug Sweep + Acceptance

Goal: One full sweep over everything. Plateau One ends when Orb is stable, predictable, recoverable.

Subsystems to Touch

All of them—this is integration testing.

Bug Sweep Checklist

□ LLM Loop
- □ Send 10 rapid messages → no duplicates, no hangs
- □ Close app mid-stream → restart works, no zombie processes
- □ Timeout test: mock slow provider → graceful error within timeout + buffer
- □ Retry test: simulate 503 → automatic retry with backoff

□ Memory
- □ Create project A, send messages, switch to project B
- □ Switch back to A → messages still there
- □ Restart app → all messages persist
- □ Delete project → messages deleted (cascade)

□ Tools
- □ Ask "what's the weather in London" → uses Orb's tool
- □ Ask "search for latest news on X" → uses Orb's web_search
- □ Tool timeout → graceful error, not hang

□ Routing
- □ "Write a Python function" → routes to Sonnet
- □ "Design the architecture for X" → routes to Opus
- □ Upload 1 image + question → routes to Gemini Flash
- □ Upload 3 images + question → routes to Gemini 2.5 Pro
- □ Upload video > 10MB → routes to Gemini 2.5 Pro
- □ "OVERRIDE SEND TO OPUS: simple question" → routes to Opus
- □ Override text stripped from actual prompt

□ UI
- □ Scroll to bottom on new message (when near bottom)
- □ Stay at scroll position when scrolled up (smart scroll)
- □ Drag-drop file → upload succeeds
- □ Markdown code block renders with highlighting
- □ Markdown table renders correctly
- □ Model badge shows correct model

□ Audit
- □ Send message → audit log entry created
- □ Use tool → tool name logged (not inputs)
- □ API error → error logged with code
- □ Query /audit/logs → returns entries
- □ Verify no prompts/responses in audit data

Recovery Testing

□ Kill backend mid-request → frontend shows error, can retry
□ Kill frontend mid-stream → backend cleans up, no zombie
□ Corrupt one message row in DB → app still loads (skip corrupt or show error)
□ Invalid API key → clear error: "Authentication failed", not crash
□ Rate limited → error message with retry guidance, retry later works
□ Backend restart → frontend detects disconnect, prompts reconnect/re-auth

Deliverables

- All bug sweep items pass
- All recovery tests pass
- No known P0/P1 bugs remaining
- Document any deferred issues as "Future Hardening"

Part 2: Gap Analysis

Comparing Architecture Map vs Plateau 1 Checklist

Area	Architecture Map Status	Plateau 1 Requirement	Gap
Tool Layer	Mentioned (`app/tools/`) but **not documented**	"Restore and stabilise"	**MAJOR GAP**
LLM Job Flow	Detailed (router, streaming, critique)	"Rock-solid, no hangs"	Timeout/retry handling missing
Memory System	Detailed (projects, notes, messages)	"Save/load correctly"	Transaction safety unclear
UI Basics	Components listed	"Scrolling, drag-drop, markdown"	Implementation details sparse
Routing	Very detailed (5-type, overrides)	"Vision + overrides"	Multi-image routing missing
Audit Logging	**Not mentioned**	"Add audit_logs table"	**MAJOR GAP**

Detailed Gap Analysis

Gap 1: Tool Layer (MAJOR - Required for P1)

What is missing:

- No documentation of `app/tools/` contents
- No tool registry architecture
- No tool binding to LLM calls documented
- Web search exists but as separate endpoint, not as tool

Why it matters:

- Models may use provider-side browsing (uncontrolled, potentially inconsistent)
- No consistent tool interface across providers
- Can't audit tool usage
- "All internet must go through Orb" requirement unmet

Classification: Required for Plateau 1

Gap 2: Audit Logging (MAJOR - Required for P1)

What is missing:

- No `audit_logs` table in schema
- No audit service documented
- No logging of model calls, job types, or errors

Why it matters:

- Can't debug routing issues
- Can't track rate limits across providers

- Can't identify failure patterns
- "Stable, predictable, recoverable" requires observability

Classification: Required for Plateau 1

Gap 3: Request Timeout / Retry Handling (Required for P1)

What is missing:

- No timeout configuration per provider
- No retry strategy documented
- No handling of transient errors (503, 429)

Why it matters:

- Slow providers can hang indefinitely
- Transient errors cause unnecessary failures
- No graceful degradation on provider issues

Implementation note: Use simple retry with exponential backoff (1s, 2s, 4s). Do NOT implement full circuit breaker state machine—overkill for single-user desktop app.

Classification: Required for Plateau 1

Gap 4: Request Deduplication (Required for P1)

What is missing:

- No request ID tracking documented
- No duplicate detection

Why it matters:

- Double-sends create duplicate messages
- UI bugs can spam requests

Implementation note: Simple in-memory dict tracking is sufficient for single-process backend. If you ever move to multi-worker, revisit with Redis or similar.

Classification: Required for Plateau 1

Gap 5: Multi-Image Routing (Required for P1)

What is missing:

- Architecture map doesn't specify multi-image handling
- Only single image → SIMPLE_VISION documented

Why it matters:

- Checklist explicitly requires: "Multi-image uploads must route to Gemini 2.5 Pro Vision"

Classification: Required for Plateau 1

Gap 6: Override Text Stripping (Required for P1)

What is missing:

- Existing overrides ("force Opus", "use GPT") documented
- New `OVERRIDE SEND TO <MODEL>` pattern not documented
- No documentation of text stripping before sending to model

Why it matters:

- Checklist explicitly requires override patterns
- Override text in prompt confuses models and wastes tokens

Classification: Required for Plateau 1

Gap 7: Frontend State Management Details (Nice to Have)

What is missing:

- No documentation of React hooks implementation
- No state management patterns documented
- Cache invalidation strategy unclear

Why it matters:

- Project switching bugs likely
- Message list state bugs likely
- But can be fixed incrementally during P1.2

Classification: Nice to have in Plateau 1 if time allows

Gap 8: Error Taxonomy Completeness (Nice to Have)

What is missing:

- Error taxonomy section exists but may be incomplete
- No mapping of errors to user-facing messages
- No retry guidance per error type

Why it matters:

- Users see cryptic errors
- No guidance on when to retry vs. report bug

Classification: Nice to have in Plateau 1 if time allows

Gap 9: Database Migration Strategy (Future Hardening)

What is missing:

- No migration framework documented (Alembic or similar)
- Ad-hoc scripts in `scripts/`
- No rollback capability

Why it matters:

- Schema changes risky without proper migrations
- But manageable for solo dev on single machine with backups

Classification: Future Hardening (beyond Plateau 1)

Gap 10: Supervisor/CI Integration Details (Future Hardening)

What is missing:

- Architecture map doesn't detail Supervisor integration
- No CI pipeline documentation
- SandboxOrb interaction not fully documented

Why it matters:

- Self-improvement system needs this
- But not required for basic stability

Classification: Future Hardening (beyond Plateau 1)

"Are there any important things we've missed from the Plateau 1 list?"

Yes. Additions needed:

Request cancellation on component unmount — Frontend can leak requests if user navigates away mid-stream. Add AbortController cleanup in `api.ts` and all components that make API calls.

-

Graceful degradation on provider outage — If Anthropic is down, should Orb fall back to GPT for code tasks? **Recommendation for P1:** Log error, show user message, do NOT auto-fallback (user can manually override). Auto-fallback adds complexity and may produce unexpected results.

- [redacted]

Session recovery after backend restart — If backend restarts, frontend's session token may be invalid. **Recommendation:** Frontend detects 401/connection error and prompts user to refresh or re-authenticate.

- [redacted]

File upload size limits — No documented limits. Large files could OOM backend or take forever to upload. **Recommendation:**

- [redacted]
    - 50MB limit for regular files
    - 500MB limit for video (with progress indicator)
    - Show clear error on oversized files

- [redacted]

Concurrent request limits — No rate limiting on Orb's own endpoints. A bug could spam requests. **Recommendation:** 10 concurrent requests per session max, queue or reject additional requests.

- [redacted]

Health check endpoint expansion — `/ping` exists but doesn't check DB or provider connectivity.
Recommendation: Add `/health` endpoint that verifies:

-
  - DB connection alive
  - At least one provider API key valid (optional, can cache result)
  - Returns structured status
-

Part 3: Plateau 1 Completion Criteria / Acceptance Tests

Behavioural Acceptance Tests

A. LLM Job Flow (Core Loop)

Test ID	Test	Expected Result	Pass/Fail
LLM-01	Send simple message, wait for response	Response received within 60s	
LLM-02	Send message, immediately send another	Second message queued or blocked, no duplicate	
LLM-03	Send message, close app before response	App restarts cleanly, no zombie processes	

LLM-04	Send 20 messages in 1 minute	All processed, no hangs
LLM-05	Mock provider timeout (>120s)	Error shown within 130s, not hang
LLM-06	Invalid API key	Clear error: "Authentication failed"
LLM-07	Rate limited response (429)	Error shown, automatic retry after delay
LLM-08	Transient error (503)	Automatic retry with backoff, succeeds
LLM-09	Navigate away mid-stream	Request cancelled, no leaked connections

B. Routing

Test ID	Test	Expected Result	Pass/Fail
RTG-01	"What's 2+2?"	Routes to GPT (TEXT_ADMIN)	
RTG-02	"Write a Python function to sort a list"	Routes to Sonnet (SMALL_CODE)	
RTG-03	"Design the database schema for a social network"	Routes to Opus (BIG_ARCHITECTURE)	
RTG-04	Upload 1 screenshot + "What is this?"	Routes to Gemini Flash	
RTG-05	Upload 3 images + "Compare these"	Routes to Gemini 2.5 Pro	
RTG-06	Upload video <10MB + simple question	Routes to Gemini Flash	
RTG-07	Upload video >10MB	Routes to Gemini 2.5 Pro	

RTG-08	Upload 2+ videos	Routes to Gemini 3 Pro
RTG-09	"OVERRIDE SEND TO OPUS: What's 2+2?"	Routes to Opus, override stripped
RTG-10	"OVERRIDE SEND TO GEMINI 3 PRO: Simple question"	Routes to Gemini 3 Pro
RTG-11	Check debug log for RTG-09	Shows override detected and stripped
RTG-12	Verify prompt sent to model for RTG-09	Contains "What's 2+2?" only, no override text