QUAD-AI CONSENSUS ENGINE

INTEGRATION GUIDE FOR ENTERPRISE PARTNERS

PUBLIC DOCUMENT — No NDA Required

Version: 1.0 | **Date:** March 20, 2026

Contact: Lisa Russell, CEO, Instant Tutor | lrussell@educationaltutorialservices.com

Complete API reference, code examples, and integration patterns for the Quad-AI Consensus Engine.

Replace one API call. Get four-provider consensus verification with full audit trails.

Table of Contents

1. What the Consensus Engine Does

The Quad-AI Consensus Engine is multi-model AI middleware. Your application sends a single API request. The engine:

1. Fires the query to **4 AI providers in true parallel** (not sequentially, not as fallbacks)
2. Applies **weighted consensus voting** based on query type and provider reliability
3. Monitors each provider through **independent circuit breakers**
4. Returns the **consensus-verified answer** with a complete audit trail

From your application's perspective, it looks like calling one AI provider. Under the hood, four independent models verify each other before the answer is returned.

What Your Application Receives

- The consensus-verified answer
- A confidence score (0–100%)
- Per-provider latency and success/failure status
- A unique trace ID for audit and replay
- Consensus method used (weighted-voting, cache, early-consensus)
- Circuit breaker status for each provider
- Cost saved through early consensus and caching

Current Providers in the Engine

Provider	Model	Strength
OpenAI	GPT-4o / GPT-5	Creative reasoning, nuanced content
Anthropic	Claude Sonnet / Opus	Computational reasoning, step-by-step logic
Google	Gemini 2.5 Pro / Flash	STEM, scientific accuracy
Perplexity	Sonar Pro	Real-time data, fact verification with citations

2. Integration Options

Option A — Consensus-as-a-Service (API)

Your application calls our hosted engine. Replace one API call. Deployed in days.

```
Your App → POST /api/ai/quad-consensus → Consensus-Verified Answer + Audit Trail
```

What changes in your code: One API endpoint. Your existing AI call becomes a call to the consensus engine. Everything else stays the same.

Option B — Owned Orchestration Layer

You acquire the engine and deploy it in your own infrastructure. Plug in your own AI providers (including proprietary or open-source models). The standardized provider adapter interface means adding a new model is a configuration change.

```
Your App → Your Infrastructure (Quad-AI Engine) → Your Choice of Providers
```

What changes: You own and operate the engine. Full control over providers, routing weights, and configuration. Custom provider adapters available under NDA.

Option C — Regional / Portfolio License

Exclusive deployment rights for a region or across a portfolio of companies. Per-query pricing available.

---

3. API Reference — Consensus Endpoint

POST /api/ai/quad-consensus

Sends a query to all 4 AI providers in parallel and returns the consensus-verified answer.

Request Headers

```
Content-Type: application/json
Authorization: Bearer <your_api_key>
X-Correlation-ID: <optional_your_tracking_id>
```

Request Body

```
{
  "prompt": "Explain photosynthesis to a 4th grader",
  "systemMessage": "You are a science tutor for elementary students. Use simple language.",
  "taskType": "explanation",
  "metadata": {
    "clientId": "your-app-id",
    "userId": "anonymous-user-hash",
    "domain": "science",
    "gradeLevel": "4"
  }
}
```

Request Parameters

Field	Type	Required	Description
`prompt`	string	Yes	The query to send to all providers. Max 10,000 characters.
`systemMessage`	string	No	Custom system prompt. If omitted, selected based on `taskType`. Max 5,000 characters.
`taskType`	string	No	Pre-configured task category (see Section 8). Determines default system message and routing weight adjustments.
`metadata`	object	No	Your application metadata. Passed through to the audit trail. Not sent to AI providers.

Successful Response (200)


```
{
  "success": true,
  "consensus": true,
  "result": "Photosynthesis is how plants make their own food using sunlight...",
  "confidence": 75,
  "providersUsed": 3,
  "totalProviders": 4,
  "traceId": "quad-1711000000000-abc1234",
  "providers": [
    {
      "name": "Claude Opus 4.5",
      "model": "claude-opus-4-5",
      "ok": true,
      "latencyMs": 2847,
      "response": "Photosynthesis is how plants make their own food..."
    },
    {
      "name": "GPT-5",
      "model": "gpt-5.1",
      "ok": true,
      "latencyMs": 1923,
      "response": "Think of photosynthesis like a recipe that plants..."
    },
    {
      "name": "Gemini 2.5 Pro",
      "model": "gemini-2.5-pro",
      "ok": true,
      "latencyMs": 1350,
      "response": "Plants use a process called photosynthesis to convert..."
    },
    {
      "name": "Perplexity",
      "model": "sonar-pro",
      "ok": false,
      "latencyMs": 0,
      "error": "Perplexity skipped for faster response."
    }
  ],
  "evidence": {
    "traceId": "quad-1711000000000-abc1234",
    "parallelExecution": true,
    "startTime": 1711000000000,
    "endTime": 1711000002847,
    "totalLatencyMs": 2847,
    "consensusMethod": "weighted-voting",
    "strictMode": false,
    "earlyConsensus": true,
    "abortedProviders": ["perplexity"],
    "cacheHit": false,
    "costSaved": 0.003
  }
}
```

4. API Reference — Streaming Endpoint

POST /api/ai/quad-consensus/stream

Same request format as the consensus endpoint, but returns results via **Server-Sent Events (SSE)** as each provider completes.

Response Headers

```
Content-Type: text/event-stream
Cache-Control: no-cache
Connection: keep-alive
```

SSE Event Stream

```
event: provider_complete
data: {"provider":"gemini","name":"Gemini 2.5 Pro","latencyMs":1350,"ok":true}

event: provider_complete
data: {"provider":"openai","name":"GPT-5","latencyMs":1923,"ok":true}

event: provider_complete
data: {"provider":"anthropic","name":"Claude Opus 4.5","latencyMs":2847,"ok":true}

event: early_consensus
data: {"confidence":75,"providersComplete":3,"threshold":75}

event: consensus_result
data: {"success":true,"consensus":true,"result":"...","confidence":75,"traceId":"quad-..."}

event: done
data: [DONE]
```

> **Use streaming when:** You want to show users a progress indicator as each provider completes, your UI can display intermediate results, or you need real-time visibility into provider performance.

---

5. API Reference — Health & Status

Endpoint	Method	Description
`GET /ready`	GET	Returns 200 if the engine is ready to accept requests. Use for load balancer health checks.
`GET /alive`	GET	Returns 200 if the engine process is running. Use for container orchestration liveness probes.
`GET /metrics/prometheus`	GET	Returns Prometheus-format metrics for monitoring dashboards.

6. Response Schema — Complete Decision Record

Every response from the consensus engine includes a complete decision record designed for regulatory compliance, audit, and debugging.

Top-Level Fields

Field	Type	Description
`success`	boolean	Whether at least one provider returned a valid response
`consensus`	boolean	Whether consensus was achieved (at least 2 providers agree)
`result`	string	The consensus-verified answer
`confidence`	number	Consensus confidence score (0–100). 75+ = high confidence (3/4). 100 = cache hit or 4/4 agreement.
`providersUsed`	number	Count of providers that returned successfully
`totalProviders`	number	Always 4 — the engine is designed for 4 concurrent providers
`traceId`	string	Unique identifier. Format: `quad-{timestamp}-{random}`. Use for audit trail lookup.

Provider Array

Field	Type	Description
`name`	string	Human-readable provider name (e.g., "Claude Opus 4.5")
`model`	string	Specific model identifier (e.g., "claude-opus-4-5")
`ok`	boolean	Whether this provider returned successfully
`latencyMs`	number	Response time in milliseconds
`response`	string	First 100 characters (truncated). Full responses available under NDA.
`error`	string	Error description if failed. Sanitized — no internal details exposed.

Evidence Object (Audit Trail)

Field	Type	Description
`traceId`	string	Same as top-level. Included for audit record completeness.
`parallelExecution`	boolean	`true` for parallel execution. `false` only for cache hits.
`startTime`	number	Unix timestamp (ms) when query started

`endTime`	number	Unix timestamp (ms) when consensus reached
`totalLatencyMs`	number	End-to-end latency in milliseconds
`consensusMethod`	string	`"weighted-voting"`, `"cache"`, or `"none"`
`strictMode`	boolean	Whether strict mode was enabled (requires 2+ providers)
`earlyConsensus`	boolean	`true` if returned before all 4 providers completed
`abortedProviders`	string[]	Providers cancelled after early consensus
`cacheHit`	boolean	`true` if served from cache
`costSaved`	number	Estimated cost saved (USD) through early consensus or caching

7. Code Examples

Python

```
import requests

QUAD_AI_URL = "https://your-endpoint.com/api/ai/quad-consensus"
API_KEY = "your_api_key"

def get_consensus_answer(prompt, task_type="question", system_message=None):
    payload = {"prompt": prompt, "taskType": task_type}
    if system_message:
        payload["systemMessage"] = system_message

    response = requests.post(
        QUAD_AI_URL,
        json=payload,
        headers={
            "Content-Type": "application/json",
            "Authorization": f"Bearer {API_KEY}"
        },
        timeout=60
    )
    response.raise_for_status()
    data = response.json()

    if data["consensus"]:
        print(f"Consensus: {data['confidence']}% | Providers: {data['providersUsed']}/{data['totalProvi
        print(f"Latency: {data['evidence']['totalLatencyMs']}ms | Trace: {data['traceId']}")
        return data["result"]
    else:
        print(f"No consensus -- escalating. Trace: {data['traceId']}")
        return None

# Education
answer = get_consensus_answer("What is the capital of France?", "question")

# Healthcare
diagnosis = get_consensus_answer(
    "Patient presents with symptoms X, Y, Z. Differential diagnoses?",
    system_message="You are a clinical decision support system. Evidence-based.",
    task_type="analysis"
)

# Finance
risk = get_consensus_answer(
    "Analyze risk: 60% equities, 30% bonds, 10% alternatives",
    system_message="You are a financial risk analyst. Quantitative assessment.",
    task_type="analysis"
)
```

JavaScript / Node.js

```
const QUAD_AI_URL = "https://your-endpoint.com/api/ai/quad-consensus";
const API_KEY = "your_api_key";

async function getConsensusAnswer(prompt, taskType = "question", systemMessage) {
  const payload = { prompt, taskType };
  if (systemMessage) payload.systemMessage = systemMessage;

  const response = await fetch(QUAD_AI_URL, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "Authorization": `Bearer ${API_KEY}`
    },
    body: JSON.stringify(payload)
  });

  if (!response.ok) throw new Error(`Engine error: ${response.status}`);
  const data = await response.json();

  if (data.consensus) {
    console.log(`Consensus: ${data.confidence}% | ${data.providersUsed}/${data.totalProviders}`);
    console.log(`Latency: ${data.evidence.totalLatencyMs}ms | Trace: ${data.traceId}`);
    return data.result;
  } else {
    console.log(`No consensus. Trace: ${data.traceId}`);
    return null;
  }
}

const answer = await getConsensusAnswer("Explain photosynthesis", "explanation");
```

JavaScript / Node.js — Streaming

```
async function streamConsensus(prompt, taskType = "question") {
  const response = await fetch(QUAD_AI_URL + "/stream", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "Authorization": `Bearer ${API_KEY}`
    },
    body: JSON.stringify({ prompt, taskType })
  });

  const reader = response.body.getReader();
  const decoder = new TextDecoder();

  while (true) {
    const { done, value } = await reader.read();
    if (done) break;

    const chunk = decoder.decode(value);
    const lines = chunk.split("\n").filter(l => l.startsWith("data:"));

    for (const line of lines) {
      const data = JSON.parse(line.slice(5).trim());
      if (data.provider)
        console.log(`${data.name}: ${data.ok ? "OK" : "FAILED"} (${data.latencyMs}ms)`);
      if (data.consensus !== undefined) return data.result;
    }
  }
}
```

Java

```
import java.net.http.*;
import java.net.URI;
import com.google.gson.*;

public class QuadAIClient {
    private static final String URL = "https://your-endpoint.com/api/ai/quad-consensus";
    private static final String KEY = "your_api_key";
    private final HttpClient client = HttpClient.newHttpClient();

    public String getConsensusAnswer(String prompt, String taskType) throws Exception {
        JsonObject payload = new JsonObject();
        payload.addProperty("prompt", prompt);
        payload.addProperty("taskType", taskType);

        HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(URL))
            .header("Content-Type", "application/json")
            .header("Authorization", "Bearer " + KEY)
            .POST(HttpRequest.BodyPublishers.ofString(payload.toString()))
            .build();

        HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());
        JsonObject data = JsonParser.parseString(response.body()).getAsJsonObject();

        if (data.get("consensus").getAsBoolean()) {
            System.out.printf("Consensus: %.0f%% | Trace: %s%n",
                data.get("confidence").getAsDouble(),
                data.get("traceId").getAsString());
            return data.get("result").getAsString();
        }
        return null;
    }
}
```

C# / .NET

```
using System.Net.Http;
using System.Text.Json;

public class QuadAIClient {
    private const string Url = "https://your-endpoint.com/api/ai/quad-consensus";
    private readonly HttpClient _client;

    public QuadAIClient(string apiKey) {
        _client = new HttpClient();
        _client.DefaultRequestHeaders.Add("Authorization", $"Bearer {apiKey}");
    }

    public async Task<string?> GetConsensusAnswer(string prompt, string taskType = "question") {
        var payload = new { prompt, taskType };
        var content = new StringContent(
            JsonSerializer.Serialize(payload),
            System.Text.Encoding.UTF8, "application/json");

        var response = await _client.PostAsync(Url, content);
        response.EnsureSuccessStatusCode();

        var json = await response.Content.ReadAsStringAsync();
        var data = JsonDocument.Parse(json).RootElement;

        if (data.GetProperty("consensus").GetBoolean()) {
            Console.WriteLine($"Consensus: {data.GetProperty("confidence").GetDouble()}%");
            return data.GetProperty("result").GetString();
        }
        return null;
    }
}
```

cURL

```
curl -X POST https://your-endpoint.com/api/ai/quad-consensus \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer your_api_key" \
  -d '{
    "prompt": "What is the capital of France?",
    "taskType": "question"
  }'
```

---

8. Task Types & System Message Routing

When you provide a `taskType` instead of a custom `systemMessage`, the engine selects an optimized system prompt and may adjust routing weights based on the task category.

Task Type	Description	Optimized For
`question`	Direct factual questions	Accuracy, conciseness

`explanation`	Detailed concept explanations	Clarity, step-by-step breakdown
`analysis`	Content analysis, pattern recognition	Depth, evidence-based reasoning
`homework`	Guided problem-solving	Pedagogical quality
`assessment`	Evaluate work, provide feedback	Specificity, actionable suggestions
`creative`	Generate creative content	Engagement, originality

Custom system messages override task type defaults entirely. Use custom messages when your domain (healthcare, legal, finance) requires specific instructions.

Healthcare Example

```
{
  "prompt": "Patient: 45M, chest pain, shortness of breath, elevated troponin",
  "systemMessage": "You are a clinical decision support system. Provide evidence-based
    differential diagnoses ranked by probability. Cite clinical guidelines.",
  "metadata": { "domain": "cardiology", "urgency": "high" }
}
```

Legal Example

```
{
  "prompt": "Review this contract clause for potential liability exposure: [clause text]",
  "systemMessage": "You are a legal analysis system. Identify liability exposure, ambiguous
    language, and missing protections. Flag issues for attorney review.",
  "metadata": { "domain": "contract-review", "jurisdiction": "UAE" }
}
```

Financial Example

```
{
  "prompt": "Analyze Q4 earnings: revenue $4.2B (+12% YoY), margin 18.3% (-2.1pp)",
  "systemMessage": "You are a financial analyst. Quantitative analysis with sector
    benchmarks. Identify risks and opportunities.",
  "metadata": { "domain": "equity-research", "sector": "technology" }
}
```

---

9. Circuit Breaker Behavior

Each AI provider has an independent circuit breaker that protects your queries from provider failures. This is automatic — your application does not need to handle provider outages.

State Machine

```
CLOSED (normal)  →  3 consecutive failures  →  OPEN (blocking)
                                                    ↓
                                              30-second cooldown
                                                    ↓
HALF-OPEN (testing)  →  2 successful probes  →  CLOSED (restored)
```

Circuit Breaker States

State	What Happens	Your Application Sees
CLOSED	Normal operation. All requests sent.	Provider: `ok: true`
OPEN	Provider blocked. No requests sent.	Provider: `ok: false`, "temporarily unavailable"
HALF-OPEN	Testing recovery. 2 probe requests sent.	Provider may show `ok: true` or `ok: false`

Configuration Per Provider

Parameter	Value	Description
Failure Threshold	3 consecutive failures	Failures before circuit opens
Reset Timeout	30 seconds	Cooldown before half-open testing
Half-Open Requests	2 probes	Successful probes needed to restore
Request Timeout	25–40 seconds	Individual provider timeout (varies)

> **What this means for your application:** You don't need retry logic, provider health monitoring, or failover handling. The circuit breaker manages all of this automatically. Degradation is automatic. Recovery is automatic.

Health Score Calculation (0–100)

- **40% weight:** Historical success rate (total successes / total requests)
- **30% weight:** Current circuit state (CLOSED=100, HALF-OPEN=50, OPEN=0)
- **30% weight:** Recent failure streak (0 failures=100, reduces by 20 per consecutive failure)

10. Cache Behavior

Scenario	Behavior	Latency	Cost
Cache miss	Full 4-provider parallel execution	2–4 seconds	Full API cost
Cache hit	Cached response returned immediately	<50ms	Zero
Early consensus	3/4 providers agree, 4th aborted	1.5–3 seconds	25–40% savings

Cache Properties

- **Key generation:** SHA-256 hash of prompt + options
- **Default TTL:** 24 hours
- **Time-sensitive TTL:** 1 hour (current events, real-time data)
- **Storage:** Hybrid — Redis (primary) with in-memory LRU fallback
- **Identification:** Cache hits flagged via `evidence.cacheHit: true`

11. Error Handling & Graceful Degradation

HTTP Status Codes

Status	Meaning	Your Action
200	Success — check `consensus` field	Parse response normally
400	Bad request — missing/invalid `prompt`	Fix request payload
401	Invalid or missing API key	Check Authorization header
429	Rate limit exceeded	Back off and retry
500	Internal engine error	Retry once. Contact support with traceId.

Graceful Degradation Levels

Providers Available	Behavior	Confidence
4/4	Full consensus voting	Up to 100%
3/4	Consensus still possible (3 agree at 75%+)	Up to 75%
2/4	Reduced consensus (2 providers cross-verify)	Up to 50%
1/4	Single provider — no consensus verification	25% — flagged
0/4	Engine returns error	0% — immediate escalation

Handling No-Consensus Responses

```
data = response.json()

if data["consensus"]:
    display_answer(data["result"])           # Use the verified answer
else:
    log_for_human_review(
        trace_id=data["traceId"],
        confidence=data["confidence"],
        providers=data["providers"]
    )
    display_fallback_message("This query requires expert review.")
```

---

12. Rate Limits & Quotas

Plan	Queries/Minute	Queries/Day	Concurrent Requests
Proof of Concept	10	500	2
Standard	60	10,000	10
Enterprise	Custom	Custom	Custom

Rate limit headers included in every response:

```
X-RateLimit-Limit: 60
X-RateLimit-Remaining: 57
X-RateLimit-Reset: 1711000060
```

---

13. Security & Compliance

Data Handling

- Queries processed in-memory. Not stored permanently unless caching is enabled.
- Cache entries encrypted at rest and expire automatically.
- No query content used for model training.
- All provider API calls use HTTPS/TLS 1.3.

Compliance

Standard	Status
FERPA	Compliant — no student data stored or shared beyond query
COPPA	Compliant — privacy-first architecture, ephemeral session data
SOC 2	Ready — structured JSON logging, correlation IDs, complete audit trails

GDPR	Considered — no personal data retained beyond cache TTL. DPA available.

Audit Trail

- Unique trace ID for replay and investigation
- Per-provider inputs, outputs, latency, and status
- Consensus method and confidence calculation
- Circuit breaker state at time of query
- Correlation ID for linking to your application's request chain

Input Validation

- Prompt: 1–10,000 characters, trimmed
- System message: 1–5,000 characters, trimmed
- Options validated against strict schema (Zod validation)
- Sanitized error messages — no internal details exposed

14. What's Included Under NDA

This public guide shows the API surface and integration patterns. Under NDA, you receive:

Document	Contents
Custom Integration Package	Company-specific API configuration, dedicated endpoints, custom routing weight profiles for your domain
Provider Adapter Specification	How to add your own AI models (proprietary, open-source, regional) to the engine
Routing Weight Methodology	How domain-specific weights are calibrated, testing methodology, documented failure patterns per provider
Circuit Breaker Deep Dive	Full state machine specification, failure pattern database, recovery strategies
Cache Architecture	Key generation algorithm, TTL strategies, cache invalidation rules, cost optimization
Audit Trail Schema	Complete database schema for long-term audit storage, query replay, compliance reporting
Architecture Assessment	10 independent AI reviews of the consensus engine architecture
Multi-Provider Analysis	10 AI providers analyze 6 industries — 60 strategic analyses
Data Architecture Pack	Full technical specification verified from source code
Deployment Runbook	On-premise deployment guide, infrastructure requirements, scaling configuration

15. Getting Started — Proof of Concept

Step 1: Get API Access

Contact Lisa Russell at **lrussell@educationaltutorialservices.com** to request a proof-of-concept API key. Include your company name, primary use case, estimated query volume, and preferred integration language.

Step 2: Make Your First Call

```
curl -X POST https://your-endpoint.com/api/ai/quad-consensus \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer your_poc_key" \
  -d '{"prompt": "What is 2 + 2?", "taskType": "question"}'
```

Step 3: Replace Your Existing AI Call

Before (single provider):

```
response = openai.chat.completions.create(
    model="gpt-4",
    messages=[{"role": "user", "content": prompt}]
)
answer = response.choices[0].message.content
```

After (consensus-verified by 4 providers):

```
response = requests.post(
    "https://your-endpoint.com/api/ai/quad-consensus",
    json={"prompt": prompt, "taskType": "question"},
    headers={"Authorization": "Bearer your_api_key"}
)
data = response.json()
answer = data["result"]       # Consensus-verified by 4 providers
trace_id = data["traceId"]    # Full audit trail
confidence = data["confidence"]  # How many providers agreed
```

> **One API call replaced. Four providers verifying. Full audit trail. Deployed in minutes.**

Step 4: Evaluate

- **Accuracy:** Consensus answers vs. your current single-provider answers
- **Latency:** Typical 2–4 seconds for full consensus
- **Audit quality:** Trace IDs, per-provider breakdown, confidence scores
- **Error handling:** Circuit breaker behavior during provider outages

Step 5: Scale or Acquire

- Continue as API service (per-query pricing)

- License for your region or portfolio
- Acquire the engine and deploy in your own infrastructure

Live Demo: https://edtutorial.services/quad-ai/ | **AI Sandbox:** https://www.edtutorial.services/ai-sandbox/