

Yestoryd rAI System - Final Design Document

Intelligent Chat for Child Reading Development

Version: 2.0 (Optimized)
Date: December 21, 2025
Authors: Amit Kumar Rai & Claude
Status: Ready for Implementation

Document History

Version	Date	Changes
1.0	Dec 21, 2025	Initial design
2.0	Dec 21, 2025	Added 5 optimizations: Parent Caching, HNSW Index, Hybrid Search, Coach CoT, Tier 0 Regex Router
2.1	Dec 21, 2025	Added: Capped Keyword Boost, Speaker Diarization, Streaming UI

Table of Contents

- 1. [Mission & Philosophy](#)
- 2. [System Architecture](#)
- 3. [Intent Classification](#)
- 4. [Role-Based Access & Boundaries](#)
- 5. [Data Sources & Handlers](#)
- 6. [Conversation Memory](#)
- 7. [Caching Strategies](#)
- 8. [Admin Weekly Insights](#)
- 9. [Technical Specifications](#)
- 10. [Cost Analysis](#)
- 11. [Implementation Roadmap](#)

1. Mission & Philosophy

1.1 Core Principle

┆ "Every response should contribute to a child's reading development."

The rAI system exists to serve one primary purpose: **improving children's reading abilities**. All features, queries, and responses must ultimately support this mission.

1.2 Value Proposition by Role

Role	rAI Value
Parents	Understand your child's learning journey. Participate actively in their development. Know what's working and what needs attention.
Coaches	Deliver better sessions through preparation. Track what teaching methods work for each child. Get AI-powered pedagogical reasoning and lesson planning assistance.
Admin	Spot platform-wide learning patterns. Identify at-risk children early. Make data-driven decisions that impact Yestoryd's educational mission.

1.3 What rAI is NOT

- ✗ Not a financial reporting tool
- ✗ Not a replacement for dashboards
- ✗ Not a general-purpose chatbot
- ✗ Not a support ticket system

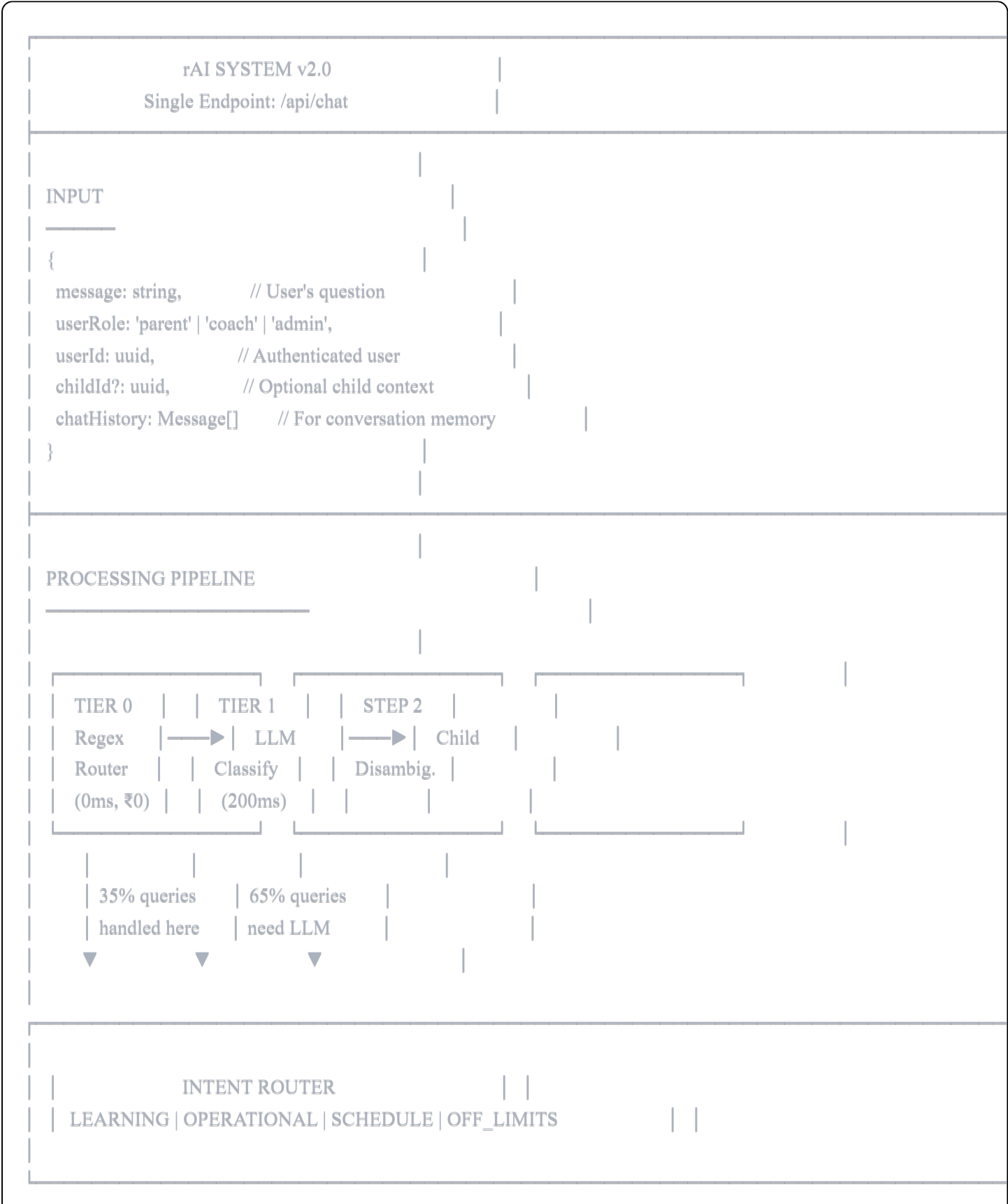
1.4 Design Principles

- Learning-Focused:** Every feature must serve child development
- Fast Operational Access:** Simple questions get instant SQL answers
- Role-Appropriate:** Users only see data relevant to their role
- Anti-Hallucination:** Never invent data - only use verified information
- Conversation-Aware:** Maintain context within a chat session
- Cost-Conscious:** Optimize for scale (10,000 children target)

- 7. **Cache-First:** Use cached summaries before expensive RAG calls
- 8. **Accuracy-First:** Use HNSW + Hybrid Search for precise results

2. System Architecture

2.1 High-Level Architecture (Optimized)





HANDLER SELECTION

LEARNING (Parent):

└─ CHECK CACHE: Is last_session_summary fresh (<24hrs)?

└─ YES → Return cached (₹0)

└─ NO → Hybrid RAG Search

LEARNING (Coach):

└─ Hybrid RAG Search (Keyword + Semantic)

└─ Chain of Thought Reasoning (Pedagogical)

LEARNING (Admin):

└─ Check admin_insights cache

└─ Fallback to platform-wide RAG

OPERATIONAL: Direct SQL with templates

SCHEDULE: Direct SQL on scheduled_sessions

OFF_LIMITS: Polite redirect + support offer

OUTPUT

```
{
  response: string,      // Natural language answer
  intent: string,        // Classified intent
  source: 'cache' | 'rag' | 'sql', // How response was generated
  needsChildSelection?: bool, // If disambiguation needed
  children?: [{id, name}], // Available children
}
```

2.2 Component Overview

Component	Technology	Purpose
Tier 0 Router	Regex patterns	Zero-latency classification for obvious queries
Tier 1 Classifier	Gemini Flash Lite	LLM classification for ambiguous queries
Session Cache	SQL column	Cached parent summaries from Recall.ai
Hybrid Search	SQL + pgvector HNSW	Keyword filtering + semantic search
CoT Reasoning	Gemini Flash Lite	Pedagogical reasoning for coaches
Admin Insights	Weekly cron (QStash)	Pre-computed platform insights

2.3 Data Flow Diagram



LEARNING (Parent)

CACHE CHECK (FIRST!)

Is query about recent session?

("How did it go?", "What happened today?", etc.)

SELECT last_session_summary, last_session_date

FROM children WHERE id = :child_id

IF summary exists AND age < 24 hours:

→ Return cached summary immediately (₹0)

ELSE:

→ Continue to Hybrid RAG



HYBRID SEARCH

1. EXTRACT FILTERS from query:

• Date range ("last Tuesday", "this week")

• Event type ("assessment", "session")

• Keywords ("phonics", "homework", "vowels")

2. SQL PRE-FILTER:

WHERE event_date BETWEEN :from AND :to

AND event_type = :type

3. VECTOR SEARCH with HNSW index:

ORDER BY embedding <=> query_embedding

4. KEYWORD BOOST:

+0.1 score for each keyword match



RESPONSE GENERATOR (Parent-Friendly)

Simple, encouraging, actionable

LEARNING (Coach)

HYBRID SEARCH (same as above)



CHAIN OF THOUGHT REASONING

System Prompt instructs Gemini to:

1. ANALYZE learning trajectory across retrieved events
2. IDENTIFY patterns in errors and successes
3. CONNECT to pedagogical principles
4. FORMULATE specific teaching recommendations

Output: Reasoned recommendation, not just summary

LEARNING (Admin)

CHECK WEEKLY INSIGHTS CACHE

SELECT * FROM admin_insights WHERE insight_type = :type
AND valid_until > NOW()

OPERATIONAL

Predefined SQL templates (see Section 5.2)

SCHEDULE

Direct SQL on scheduled_sessions (see Section 5.3)

OFF_LIMITS

Polite redirect + "Would you like to connect with support?"

3. Intent Classification

3.1 Two-Tier Classification System

Tier 0: Regex Router (Zero Latency, Zero Cost)

Handles 35% of queries instantly without any AI call.

typescript

```
function tier0Router(message: string): Intent | null {  
  const lowerMessage = message.toLowerCase();
```

```
// =====
```

```
// SCHEDULE patterns - very high confidence
```

```
// =====
```

```
const schedulePatterns = [  
  /when is (my|the|our) (next|upcoming) (session|class|meeting)/,  
  /what('s| is) (my|the|today'?s?) schedule/,  
  /what time is/,  
  /show me my (calendar|sessions)/,  
  /do i have (any )?(sessions?|classes?) (today|tomorrow|this week)/,  
  /next (session|class|meeting)/,  
];  
if (schedulePatterns.some(p => p.test(lowerMessage))) {  
  return 'SCHEDULE';  
}
```

```
// =====
```

```
// OPERATIONAL patterns - high confidence
```

```
// =====
```

```
const operationalPatterns = [  
  /how many (children|students|kids) do i have/,  
  /how many sessions? (have i|did i|completed)/,  
  /who is my coach/,  
  /what('s| is| did) (the )?(program )?(cost|price|fee)/,  
  /how (long|many months) is the program/,  
  /how many sessions (are )?(included|in the program)/,  
  /is my (enrollment|payment) (active|complete)/,  
];  
if (operationalPatterns.some(p => p.test(lowerMessage))) {  
  return 'OPERATIONAL';  
}
```

```
// =====
```

```
// OFF_LIMITS patterns - high confidence
```

```
// =====
```

```
const offLimitsPatterns = [  
  /what are my earnings/,  
  /how much (have i|did i) (earn|make)/,  
  /show me (my )?payout/,  
  /(my|the) (revenue|income)/,  
  /other (children|students|coaches)/,  
  /platform (revenue|stats|metrics)/,  
  /how much does (the )?coach (earn|make)/,  
];
```

```
if (offLimitsPatterns.some(p => p.test(lowerMessage))) {
  return 'OFF_LIMITS';
}

// No match - proceed to Tier 1 (LLM classification)
return null;
}
```

Tier 1: LLM Classifier (For Ambiguous Queries)

Handles the remaining 65% of queries that need semantic understanding.

```
typescript

const INTENT_CLASSIFICATION_PROMPT = `
You are an intent classifier for a children's reading education platform.

Classify the following query into exactly ONE category:

LEARNING: Questions about child progress, development, teaching strategies,
          session summaries, recommendations, what to work on next

OPERATIONAL: Counts, coach info, program info, payment status, enrollment status

SCHEDULE: Session times, calendar, when is next session, schedule today/week

OFF_LIMITS: Earnings, payouts, other users' data, platform revenue metrics

User role: {role}
Query: {message}

Respond ONLY with JSON:
{"intent": "LEARNING|OPERATIONAL|SCHEDULE|OFF_LIMITS", "entities": ["extracted names or topics"], "confidence":
`;
```

3.2 Intent Categories

Intent	Description	Handler	Cost
LEARNING	Child progress, development, teaching strategies	Cache → Hybrid RAG	₹0 - ₹0.015
OPERATIONAL	Counts, coach info, program details	Direct SQL	₹0.005
SCHEDULE	Session times, calendar queries	Direct SQL	₹0.005
OFF_LIMITS	Earnings, other users' data	Redirect	₹0.001

3.3 Classification Examples

Query	Tier	Intent	Entities
"When is my next session?"	0 (Regex)	SCHEDULE	[]
"How many children do I have?"	0 (Regex)	OPERATIONAL	[]
"What are my earnings?"	0 (Regex)	OFF_LIMITS	[]
"How is Sima doing?"	1 (LLM)	LEARNING	["Sima"]
"Prepare me for tomorrow's session with Aarav"	1 (LLM)	LEARNING	["Aarav"]
"What phonics strategies work for struggling readers?"	1 (LLM)	LEARNING	["phonics"]

4. Role-Based Access & Boundaries

4.1 Parent Access

Allowed Queries

Category	Query Type	Data Source	Example
LEARNING	Recent session summary	Cache first	"How did today's session go?"
LEARNING	Child progress	Hybrid RAG	"How is my child doing overall?"
LEARNING	Recommendations	Hybrid RAG	"What books should my child read?"
LEARNING	Home practice tips	Hybrid RAG + General	"How can I help at home?"
OPERATIONAL	Coach information	SQL	"Who is my coach?"
OPERATIONAL	Program details	SQL (site_settings)	"How many sessions are included?"
OPERATIONAL	Payment status	SQL	"Is my enrollment active?"
SCHEDULE	Next session	SQL	"When is the next session?"
SCHEDULE	Session history	SQL	"How many sessions completed?"

Off-Limits

Topic	Response
Other children's data	"I can only share information about your enrolled children."
Coach earnings	"I don't have access to that information."
Platform statistics	"I focus on your child's learning journey."

Multi-Child Handling

If parent has multiple children enrolled and asks about "my child":

 "I see you have Sima and Aarav enrolled. Which child are you asking about?"

4.2 Coach Access

Allowed Queries

Category	Query Type	Data Source	Example
LEARNING	Student progress	Hybrid RAG + CoT	"How is Aarav doing?"
LEARNING	Session preparation	Hybrid RAG + CoT	"Prepare me for tomorrow's session with Sima"
LEARNING	Teaching strategies	Hybrid RAG + CoT	"What phonics games work for struggling readers?"
LEARNING	What works analysis	Hybrid RAG + CoT	"What teaching methods work best with Sima?"
OPERATIONAL	Student count	SQL	"How many children do I have?"
OPERATIONAL	Session count	SQL	"How many sessions this month?"
OPERATIONAL	Program structure	SQL	"How many sessions per enrollment?"
SCHEDULE	Today's schedule	SQL	"What's my schedule today?"
SCHEDULE	Week's schedule	SQL	"How many sessions this week?"

Off-Limits

Topic	Response
Earnings/payouts	"For earnings information, please check your Earnings dashboard."
Other coaches' data	"I can only share information about your assigned students."
Unassigned children	"I don't have access to that student's information."

Chain of Thought Reasoning (Coach-Only)

For LEARNING queries, coaches receive pedagogically-reasoned responses:

Simple summary (NOT what coaches get):

█ "Aarav struggled with 'th' sounds in the last session."

Chain of Thought reasoning (what coaches GET):

█ "Aarav struggled with 'th' sounds in the last session. Looking at his learning trajectory: he mastered 'sh' sounds quickly (2 sessions), but 'th' is taking longer. This pattern suggests the voiced/voiceless distinction is his challenge. Since he already knows 'sh' well, I recommend using the 'sh' vs 'th' contrast method - building from his existing anchor point should accelerate his progress. Try the minimal pairs activity: 'ship/this', 'she/the'."

4.3 Admin Access

Allowed Queries

Category	Query Type	Data Source	Example
LEARNING INSIGHTS	At-risk children	Cached Insights	"Which children are at risk?"
LEARNING INSIGHTS	Common struggles	Cached Insights	"What are children struggling with?"
LEARNING INSIGHTS	Coach effectiveness	Cached Insights	"Which coaches have best outcomes?"
LEARNING INSIGHTS	Learning trends	Cached Insights	"How are scores trending?"
OPERATIONAL	Enrollment metrics	SQL	"How many enrollments this month?"
OPERATIONAL	Session metrics	SQL	"What's the session completion rate?"
OPERATIONAL	Coach workload	SQL	"How many sessions per coach?"
OPERATIONAL	Assessment metrics	SQL	"How many assessments this week?"
SCHEDULE	Platform schedule	SQL	"How many sessions today?"

Off-Limits

Topic	Response
Financial/Revenue data	"For revenue and payout information, please check the Revenue dashboard."
Personal contact details	Only when needed for follow-up on learning issues

5. Data Sources & Handlers

5.1 LEARNING Handler (Hybrid RAG)

Purpose: Answer questions about child development, progress, and learning

5.1.1 Parent Flow (With Caching)

typescript

```

async function handleParentLearning(
  message: string,
  childId: string,
  chatHistory: ChatMessage[]
): Promise<ChatResponse> {

  // =====
  // STEP 1: CHECK CACHE FIRST
  // =====

  if (isRecentSessionQuery(message)) {
    const child = await getChild(childId);
    const hoursSinceSession = getHoursSince(child.last_session_date);

    if (child.last_session_summary && hoursSinceSession < 24) {
      // Return cached summary - NO AI CALLS!
      return {
        response: formatCachedSummary(
          child.last_session_summary,
          child.last_session_date,
          child.child_name
        ),
        intent: 'LEARNING',
        source: 'cache'
      };
    }
  }

  // =====
  // STEP 2: HYBRID SEARCH
  // =====

  const events = await hybridSearch(message, childId, 'parent');

  // =====
  // STEP 3: GENERATE RESPONSE (Parent-friendly)
  // =====

  const response = await generateParentResponse(events, message, chatHistory);

  return {
    response,
    intent: 'LEARNING',
    source: 'rag'
  };
}

function isRecentSessionQuery(message: string): boolean {
  const patterns = [

```

```

    /how did (it|the session|today'?s? session) go/i,
    /what happened (today|in the session|in today'?s session)/i,
    /how was (the|today'?s?) (session|class)/i,
    /tell me about (the )?(last|recent|today'?s?) session/i,
    /session (summary|update|recap)/i,
  ];
  return patterns.some(p => p.test(message));
}

function formatCachedSummary(summary: string, date: Date, childName: string): string {
  const timeAgo = formatTimeAgo(date);
  return `Here's what happened in ${childName}'s session ${timeAgo}:\n\n${summary}`;
}

```

5.1.2 Coach Flow (With Chain of Thought)

typescript

```
async function handleCoachLearning(
```

```
  message: string,
```

```
  childId: string | null,
```

```
  coachId: string,
```

```
  chatHistory: ChatMessage[]
```

```
): Promise<ChatResponse> {
```

```
//=====
```

```
// STEP 1: HYBRID SEARCH
```

```
//=====
```

```
const events = await hybridSearch(message, childId, 'coach', coachId);
```

```
//=====
```

```
// STEP 2: CHAIN OF THOUGHT REASONING
```

```
//=====
```

```
const response = await generateCoachResponse(events, message, chatHistory);
```

```
return {
```

```
  response,
```

```
  intent: 'LEARNING',
```

```
  source: 'rag'
```

```
};
```

```
}
```

```
const COACH_SYSTEM_PROMPT = `
```

```
You are rAI, a reading development assistant for coaches at Yestoryd.
```

CRITICAL: Before generating your response, you MUST follow this reasoning process:

STEP 1: ANALYZE THE LEARNING TRAJECTORY

Look across ALL the retrieved session data and answer:

- What patterns do you see in this child's progress?
- Where have they improved? Where are they stuck?
- What teaching methods worked before? What didn't?

STEP 2: IDENTIFY PEDAGOGICAL CONNECTIONS

- How does their current struggle relate to past successes?
- What prerequisite skills are solid? What's missing?
- What does research suggest for this type of learner?

STEP 3: FORMULATE ACTIONABLE RECOMMENDATIONS

-
- Suggest SPECIFIC teaching strategies based on patterns
 - Reference what worked before ("The phonics game worked in session 3")
 - Suggest progression paths ("Since they mastered X, they're ready for Y")
-

Your response should demonstrate this reasoning, not just summarize facts.

Retrieved learning events:

{events}

Conversation history:

{chatHistory}

Coach's question: {message}

`;

5.1.3 Hybrid Search Implementation

typescript

```

async function hybridSearch(
  query: string,
  childId: string | null,
  userRole: string,
  userId?: string
): Promise<LearningEvent[]> {

  // =====
  // STEP 1: EXTRACT STRUCTURED FILTERS
  // =====

  const filters = extractQueryFilters(query);

  // =====
  // STEP 2: GENERATE QUERY EMBEDDING
  // =====

  const queryEmbedding = await generateEmbedding(query);

  // =====
  // STEP 3: EXECUTE HYBRID SEARCH
  // =====

  const results = await supabase.rpc('hybrid_match_learning_events', {
    query_embedding: queryEmbedding,
    filter_child_id: childId,
    filter_coach_id: userRole === 'coach' ? userId : null,
    filter_date_from: filters.dateRange?.from,
    filter_date_to: filters.dateRange?.to,
    filter_event_type: filters.eventType,
    filter_keywords: filters.keywords,
    match_threshold: 0.4,
    match_count: 15
  });

  return results.data;
}

function extractQueryFilters(query: string): QueryFilters {
  const filters: QueryFilters = {};
  const now = new Date();

  // =====
  // DATE EXTRACTION
  // =====

  if (/today/i.test(query)) {
    filters.dateRange = {
      from: startOfDay(now),
      to: endOfDay(now)
    };
  }
}

```

```

};
} else if (/yesterday/i.test(query)) {
  const yesterday = subDays(now, 1);
  filters.dateRange = {
    from: startOfDay(yesterday),
    to: endOfDay(yesterday)
  };
} else if (/last week/i.test(query)) {
  filters.dateRange = {
    from: subDays(now, 7),
    to: now
  };
} else if (/last month/i.test(query)) {
  filters.dateRange = {
    from: subMonths(now, 1),
    to: now
  };
} else if (/last (monday|tuesday|wednesday|thursday|friday|saturday|sunday)/i.test(query)) {
  const dayMatch = query.match(/last (monday|tuesday|wednesday|thursday|friday|saturday|sunday)/i);
  if (dayMatch) {
    const targetDay = getLastWeekday(dayMatch[1]);
    filters.dateRange = {
      from: startOfDay(targetDay),
      to: endOfDay(targetDay)
    };
  }
}
}

```

```

// EVENT TYPE EXTRACTION

```

```

if (/assessment|test|score|reading level/i.test(query)) {
  filters.eventType = 'assessment';
} else if (/session|class|lesson|coaching/i.test(query)) {
  filters.eventType = 'session';
} else if (/quiz|homework|practice/i.test(query)) {
  filters.eventType = 'quiz';
}

```

```

// KEYWORD EXTRACTION (for boosting)

```

```

const keywordPatterns = [
  { pattern: /phonics?/i, keyword: 'phonics' },
  { pattern: /vowels?/i, keyword: 'vowel' },
  { pattern: /consonants?/i, keyword: 'consonant' },
  { pattern: /blends?/i, keyword: 'blend' },

```

```
{ pattern: /digraphs?/i, keyword: 'digraph' },
{ pattern: /fluency/i, keyword: 'fluency' },
{ pattern: /comprehension/i, keyword: 'comprehension' },
{ pattern: /homework/i, keyword: 'homework' },
{ pattern: /reading speed|wpm/i, keyword: 'wpm' },
{ pattern: /pronunciation/i, keyword: 'pronunciation' },
];

filters.keywords = keywordPatterns
  .filter(({ pattern }) => pattern.test(query))
  .map(({ keyword }) => keyword);

return filters;
}
```

5.2 OPERATIONAL Handler (Direct SQL)

Purpose: Fast answers to structured data questions

SQL Templates:

```
sql
```

-- *PARENT QUERIES*

-- *parent_coach_info*

```
SELECT c.name as coach_name, c.email as coach_email
FROM coaches c
JOIN children ch ON ch.assigned_coach_id = c.id
WHERE ch.parent_id = $1 AND ch.status = 'enrolled'
LIMIT 1;
```

-- *parent_payment_status*

```
SELECT
  p.amount,
  p.status,
  p.created_at,
  e.status as enrollment_status
FROM payments p
JOIN enrollments e ON p.enrollment_id = e.id
WHERE p.parent_id = $1
ORDER BY p.created_at DESC
LIMIT 1;
```

-- *parent_program_info*

```
SELECT setting_key, setting_value
FROM site_settings
WHERE setting_key IN ('program_price', 'program_duration', 'session_count');
```

-- *parent_session_progress*

```
SELECT
  COUNT(*) FILTER (WHERE ss.status = 'completed') as completed,
  COUNT(*) as total,
  c.child_name
FROM scheduled_sessions ss
JOIN children c ON ss.child_id = c.id
WHERE c.parent_id = $1 AND c.status = 'enrolled'
GROUP BY c.id, c.child_name;
```

-- *COACH QUERIES*

-- *coach_child_count*

```
SELECT COUNT(*) as count
FROM children
```

```
WHERE assigned_coach_id = $1 AND status = 'enrolled';
```

```
-- coach_session_count_month
```

```
SELECT
```

```
  COUNT(*) as total,
```

```
  COUNT(*) FILTER (WHERE status = 'completed') as completed
```

```
FROM scheduled_sessions
```

```
WHERE coach_id = $1
```

```
  AND scheduled_date >= date_trunc('month', NOW());
```

```
-- coach_session_count_all
```

```
SELECT
```

```
  COUNT(*) FILTER (WHERE status = 'completed') as completed,
```

```
  COUNT(*) FILTER (WHERE status = 'scheduled' AND scheduled_date > NOW()) as upcoming,
```

```
  COUNT(*) as total
```

```
FROM scheduled_sessions
```

```
WHERE coach_id = $1;
```

```
-- ADMIN QUERIES
```

```
-- admin_enrollment_count
```

```
SELECT
```

```
  COUNT(*) FILTER (WHERE created_at >= date_trunc('month', NOW())) as this_month,
```

```
  COUNT(*) FILTER (WHERE created_at >= date_trunc('week', NOW())) as this_week,
```

```
  COUNT(*) as total
```

```
FROM enrollments
```

```
WHERE status = 'active';
```

```
-- admin_session_metrics
```

```
SELECT
```

```
  COUNT(*) as total_sessions,
```

```
  COUNT(*) FILTER (WHERE status = 'completed') as completed,
```

```
  COUNT(*) FILTER (WHERE status = 'no_show') as no_shows,
```

```
  ROUND(100.0 * COUNT(*) FILTER (WHERE status = 'completed') / NULLIF(COUNT(*), 0), 1) as completion_rate
```

```
FROM scheduled_sessions
```

```
WHERE scheduled_date >= date_trunc('month', NOW())
```

```
  AND scheduled_date <= NOW();
```

```
-- admin_coach_workload
```

```
SELECT
```

```
  c.name as coach_name,
```

```
  COUNT(ss.id) FILTER (WHERE ss.status = 'completed') as completed_sessions,
```

```
  COUNT(DISTINCT ss.child_id) as active_students
```

```
FROM coaches c
```

```
LEFT JOIN scheduled_sessions ss ON c.id = ss.coach_id
AND ss.scheduled_date >= date_trunc('month', NOW())
WHERE c.status = 'active'
GROUP BY c.id, c.name
ORDER BY completed_sessions DESC;

-- admin_assessment_metrics
SELECT
COUNT(*) as total_assessments,
COUNT(*) FILTER (WHERE created_at >= date_trunc('week', NOW())) as this_week,
COUNT(*) FILTER (WHERE created_at >= date_trunc('month', NOW())) as this_month
FROM children
WHERE assessment_completed = true;
```

5.3 SCHEDULE Handler (Direct SQL)

```
sql
```

```
-- PARENT SCHEDULE
```

```
-- parent_next_session
```

```
SELECT
    ss.scheduled_time,
    ss.session_type,
    ss.google_meet_link,
    c.child_name,
    co.name as coach_name
FROM scheduled_sessions ss
JOIN children c ON ss.child_id = c.id
JOIN coaches co ON ss.coach_id = co.id
WHERE c.parent_id = $1
    AND ss.scheduled_time > NOW()
    AND ss.status = 'scheduled'
ORDER BY ss.scheduled_time
LIMIT 1;
```

```
-- parent_upcoming_sessions
```

```
SELECT
    ss.scheduled_time,
    ss.session_type,
    ss.google_meet_link,
    c.child_name
FROM scheduled_sessions ss
JOIN children c ON ss.child_id = c.id
WHERE c.parent_id = $1
    AND ss.scheduled_time > NOW()
    AND ss.status = 'scheduled'
ORDER BY ss.scheduled_time
LIMIT 5;
```

```
-- COACH SCHEDULE
```

```
-- coach_today_schedule
```

```
SELECT
    c.child_name,
    ss.scheduled_time,
    ss.session_type,
    ss.google_meet_link
FROM scheduled_sessions ss
```

```

JOIN children c ON ss.child_id = c.id
WHERE ss.coach_id = $1
    AND DATE(ss.scheduled_time) = CURRENT_DATE
    AND ss.status = 'scheduled'
ORDER BY ss.scheduled_time;

-- coach_week_schedule
SELECT
    DATE(ss.scheduled_time) as session_date,
    COUNT(*) as session_count
FROM scheduled_sessions ss
WHERE ss.coach_id = $1
    AND ss.scheduled_time >= date_trunc('week', NOW())
    AND ss.scheduled_time < date_trunc('week', NOW()) + INTERVAL '7 days'
    AND ss.status = 'scheduled'
GROUP BY session_date
ORDER BY session_date;

-- coach_tomorrow_sessions
SELECT
    c.child_name,
    ss.scheduled_time,
    ss.session_type,
    ss.google_meet_link
FROM scheduled_sessions ss
JOIN children c ON ss.child_id = c.id
WHERE ss.coach_id = $1
    AND DATE(ss.scheduled_time) = CURRENT_DATE + INTERVAL '1 day'
    AND ss.status = 'scheduled'
ORDER BY ss.scheduled_time;

-- =====
-- ADMIN SCHEDULE
-- =====

-- admin_today_platform_schedule
SELECT
    COUNT(*) as total_sessions,
    COUNT(DISTINCT coach_id) as active_coaches,
    COUNT(DISTINCT child_id) as children_with_sessions
FROM scheduled_sessions
WHERE DATE(scheduled_time) = CURRENT_DATE
    AND status = 'scheduled';

```

5.4 OFF_LIMITS Handler

Purpose: Polite redirects for out-of-scope queries

typescript

```
const OFF_LIMITS_RESPONSES = {  
  // _____  
  // EARNINGS (Coach, Admin)  
  // _____  
  earnings: {  
    coach: "For earnings and payout information, please check your Earnings dashboard. Is there anything about your students'  
    admin: "For revenue and financial data, please check the Revenue dashboard. Would you like to know about learning metri  
  },  
  
  // _____  
  // OTHER USERS' DATA  
  // _____  
  other_users: {  
    parent: "I can only share information about your enrolled children. Is there something specific about {childName} you'd lik  
    coach: "I can only provide information about students assigned to you. Would you like to know about one of your students?  
    admin: "I can help with platform-wide learning insights, but not individual user details. What learning metrics would help?  
  },  
  
  // _____  
  // SUPPORT OFFER (Always included)  
  // _____  
  support_offer: "Would you like me to connect you with support? You can reach us on WhatsApp: 918976287997",  
  
  // _____  
  // UNKNOWN / UNCLEAR  
  // _____  
  unknown: "I'm not sure I can help with that. I'm best at answering questions about children's reading development. Is there s  
};
```

6. Conversation Memory

6.1 Purpose

Enable natural multi-turn conversations where users can refer to previous messages:

User: "How is Sima doing?"

rAI: "Sima has improved her fluency to 58 WPM..."

User: "What about her reading speed?"

rAI: (*Knows "her" = Sima*) "Sima's reading speed has increased by 13 WPM over the last 3 sessions..."

6.2 Data Structure

typescript

```
interface ChatMessage {  
  role: 'user' | 'assistant';  
  content: string;  
  timestamp: Date;  
  childId?: string; // Track which child was discussed  
  intent?: string; // Track query type  
}  
  
interface ChatRequest {  
  message: string;  
  userRole: 'parent' | 'coach' | 'admin';  
  userId: string;  
  childId?: string;  
  chatHistory: ChatMessage[]; // Last 10 messages from frontend  
}
```

6.3 Context Management

typescript

```

function buildConversationContext(history: ChatMessage[]): string {
  // Take last 5 exchanges (10 messages)
  const recent = history.slice(-10);

  return recent.map(msg =>
    `${msg.role} === 'user' ? 'User' : 'rAI': ${msg.content}`
  ).join('\n');
}

function extractChildFromHistory(history: ChatMessage[]): string | null {
  // Find most recently mentioned child (reverse search)
  for (let i = history.length - 1; i >= 0; i--) {
    if (history[i].childId) {
      return history[i].childId;
    }
  }
  return null;
}

function resolvePronouns(message: string, context: ConversationContext): string {
  // If message uses pronouns and we have child context, resolve them
  if (/^(\b(he|she|they|their|his|her|them)\b/i.test(message) && context.lastChildMentioned) {
    // Pass child context to LLM for proper resolution
    return message; // LLM handles with context
  }
  return message;
}

```

6.4 Memory Limits

Limit	Value	Rationale
History depth	10 messages	Balance context vs. token cost
Session timeout	30 minutes	After inactivity, start fresh
Child context	Persists in session	Don't ask "which child" repeatedly

7. Caching Strategies

7.1 Parent Session Summary Cache

Problem: 80% of parent queries are "How did the session go?" — triggering expensive RAG calls.

Solution: Generate and cache a parent-friendly summary immediately after each session ends.

7.1.1 Trigger: Recall.ai Webhook (Single Gemini Call, Two Outputs)

typescript

// In /api/webhooks/recall/route.ts

async function handleBotDone(payload: RecallWebhookPayload) {

// =====

// STEP 1: BUILD SPEAKER-LABELED TRANSCRIPT

// =====

// CRITICAL: Must use Recall.ai's speaker diarization for accurate analysis

// Without speaker labels, Gemini can't distinguish Coach vs Child speech

const transcript = buildTranscriptWithSpeakers(payload.data.transcript, {
 coachEmail: session.coach_email, // From scheduled_sessions
 childName: session.child_name
});

// =====

// SINGLE GEMINI CALL - TWO OUTPUTS

// =====

const analysisPrompt = `

You are analyzing a reading coaching session transcript for Yestoryd.

IMPORTANT: The transcript is formatted with speaker labels:

- COACH: statements from the reading coach
- CHILD: statements from the child being coached

Generate TWO outputs in JSON format:

1. COACH_ANALYSIS: Detailed analysis for internal use

- focus_area: What was the main focus?
- skills_worked_on: List of skills practiced
- progress_rating: 'declined' | 'same' | 'improved' | 'significant_improvement'
- engagement_level: 'low' | 'medium' | 'high'
- breakthrough_moment: Any notable achievements?
- concerns: Any issues to flag?
- homework_assigned: true/false
- homework_description: What was assigned?
- next_session_focus: Recommended focus for next session
- coach_talk_ratio: Percentage of time coach spoke (aim for <40%)
- child_reading_samples: Array of notable reading attempts by child

2. PARENT_SUMMARY: A warm, encouraging 2-3 sentence summary for parents

- Should be positive and actionable
- Mention what child practiced
- Include one specific thing parent can do at home
- Do NOT include scores or technical terms

Child's name: {childName}

Session type: {sessionType}

Transcript:

{transcript}

Respond with JSON:

```
{
  "coach_analysis": { ... },
  "parent_summary": "string"
}
```

```
const response = await gemini.generateContent(analysisPrompt);
```

```
const { coach_analysis, parent_summary } = JSON.parse(response.text());
```

```
// ... rest of saving logic
```

```
}
```

```
// =====
// SPEAKER DIARIZATION HANDLER
```

```
interface TranscriptWord {
  text: string;
  start_timestamp: number;
  end_timestamp: number;
  speaker_id: string; // Recall.ai provides this
}
```

```
interface SpeakerMapping {
  coachEmail: string;
  childName: string;
}
```

```
function buildTranscriptWithSpeakers(
  transcriptData: TranscriptWord[],
  mapping: SpeakerMapping
): string {
  // =====
  // STEP 1: Identify speakers from Recall.ai's diarization
  // =====
  // Recall.ai assigns speaker_id like "speaker_0", "speaker_1"
  // We need to map these to COACH and CHILD
```

```
const speakerCounts: Record<string, number> = {};
```

```
const speakerFirstWords: Record<string, string[]> = {};
```

```

for (const word of transcriptData) {
  speakerCounts[word.speaker_id] = (speakerCounts[word.speaker_id] || 0) + 1;
  if (!speakerFirstWords[word.speaker_id]) {
    speakerFirstWords[word.speaker_id] = [];
  }
  if (speakerFirstWords[word.speaker_id].length < 20) {
    speakerFirstWords[word.speaker_id].push(word.text);
  }
}

// -----
// STEP 2: Heuristic speaker identification
// -----

// Coach typically:
// - Speaks first ("Hello", "Let's start")
// - Uses more words overall
// - Uses instructional language ("read", "try", "good job")
// Child typically:
// - Speaks less
// - More hesitations ("um", "uh")
// - Reading attempts (spelling out words)

const speakerIds = Object.keys(speakerCounts);
let coachSpeakerId: string;
let childSpeakerId: string;

if (speakerIds.length === 2) {
  // Two speakers detected - most common case
  const [speaker1, speaker2] = speakerIds;

  // Coach usually speaks more
  if (speakerCounts[speaker1] > speakerCounts[speaker2]) {
    coachSpeakerId = speaker1;
    childSpeakerId = speaker2;
  } else {
    coachSpeakerId = speaker2;
    childSpeakerId = speaker1;
  }
}

// Validate with first words (coach likely says "hello", "let's", etc.)
const coachWords = speakerFirstWords[coachSpeakerId].join(' ').toLowerCase();
const childWords = speakerFirstWords[childSpeakerId].join(' ').toLowerCase();

// If child's first words seem more instructional, swap
const instructionalPatterns = /\b(hello|hi|let's|today|read|start|open)\b/;
if (instructionalPatterns.test(childWords) && !instructionalPatterns.test(coachWords)) {

```

```

    [coachSpeakerId, childSpeakerId] = [childSpeakerId, coachSpeakerId];
  }
} else if (speakerIds.length === 1) {
  // Only one speaker detected - likely diarization issue
  // Assume it's mixed and label as MIXED
  coachSpeakerId = speakerIds[0];
  childSpeakerId = ""; // Will show as MIXED
} else {
  // More than 2 speakers (parent might be present)
  // Take top 2 by word count
  const sorted = speakerIds.sort((a, b) => speakerCounts[b] - speakerCounts[a]);
  coachSpeakerId = sorted[0];
  childSpeakerId = sorted[1] || "";
}

```

```

// STEP 3: Build formatted transcript
//

```

```

const lines: string[] = [];
let currentSpeaker = "";
let currentLine = "";

for (const word of transcriptData) {
  const speaker = word.speaker_id === coachSpeakerId
    ? 'COACH'
    : word.speaker_id === childSpeakerId
    ? 'CHILD'
    : 'OTHER';

  if (speaker !== currentSpeaker) {
    // Speaker changed - save current line and start new one
    if (currentLine) {
      lines.push(`${currentSpeaker}: "${currentLine.trim()}"`);
    }
    currentSpeaker = speaker;
    currentLine = word.text + ' ';
  } else {
    currentLine += word.text + ' ';
  }
}

// Don't forget the last line
if (currentLine) {
  lines.push(`${currentSpeaker}: "${currentLine.trim()}"`);
}

```

```
return lines.join('\n');
}

//=====
// EXAMPLE OUTPUT
//=====
/*
COACH: "Hello Aarav! How are you today? Let's start with our warm-up reading."
CHILD: "I'm good."
COACH: "Great! Can you read this word for me? It says 'through'."
CHILD: "Th... thr... through?"
COACH: "Excellent! You got the 'th' sound right. Let's try another one."
CHILD: "Um... th-rough. Through!"
COACH: "Perfect! You're getting faster. Now let's try a sentence."
*/
```

7.1.2 Database Saving (After Analysis)

typescript

```
// Continuation of handleBotDone after Gemini analysis
```

```
//=====
// SAVE TO DATABASE
//=====
```

```
// 1. Update children table with cached parent summary
```

```
await supabase
  .from('children')
  .update({
    last_session_summary: parent_summary,
    last_session_date: new Date().toISOString(),
    last_session_focus: coach_analysis.focus_area
  })
  .eq('id', childId);
```

```
// 2. Save detailed analysis to learning_events (for RAG)
```

```
await supabase
  .from('learning_events')
  .insert({
    child_id: childId,
    coach_id: coachId,
    session_id: sessionId,
    event_type: 'session',
    event_date: new Date().toISOString(),
    event_data: coach_analysis,
    ai_summary: parent_summary,
    content_for_embedding: buildSearchableContent(coach_analysis, childName),
    embedding: await generateEmbedding(buildSearchableContent(coach_analysis, childName))
  });
```

```
// 3. Update scheduled_sessions
```

```
await supabase
  .from('scheduled_sessions')
  .update({
    status: 'completed',
    tldv_ai_summary: JSON.stringify(coach_analysis),
    recall_transcript: transcript
  })
  .eq('id', sessionId);
}
```

```
function buildSearchableContent(analysis: CoachAnalysis, childName: string): string {
  return `
    ${childName} coaching session
    Focus: ${analysis.focus_area}
  `;
}
```

```
Skills: ${analysis.skills_worked_on.join(', ')}
Progress: ${analysis.progress_rating}
Engagement: ${analysis.engagement_level}
Coach talk ratio: ${analysis.coach_talk_ratio}%
${analysis.breakthrough_moment ? `Breakthrough: ${analysis.breakthrough_moment}` : ''}
${analysis.concerns ? `Concerns: ${analysis.concerns}` : ''}
${analysis.homework_assigned ? `Homework: ${analysis.homework_description}` : ''}
Next session: ${analysis.next_session_focus}
Child reading samples: ${analysis.child_reading_samples?.join(', ') || 'N/A'}
`.trim();
}
```

7.1.2 Database Schema Addition

```
sql

-- Add caching columns to children table
ALTER TABLE children ADD COLUMN IF NOT EXISTS last_session_summary TEXT;
ALTER TABLE children ADD COLUMN IF NOT EXISTS last_session_date TIMESTAMP WITH TIME ZONE;
ALTER TABLE children ADD COLUMN IF NOT EXISTS last_session_focus TEXT;
```

7.1.3 Cache Retrieval in Chat Handler

```
typescript
```

```
async function handleParentLearning(message: string, childId: string): Promise<ChatResponse> {  
  
    // =====  
    // CHECK CACHE FIRST - Before any AI calls  
    // =====  
  
    if (isRecentSessionQuery(message)) {  
        const { data: child } = await supabase  
            .from('children')  
            .select('last_session_summary, last_session_date, child_name')  
            .eq('id', childId)  
            .single();  
  
        if (child?.last_session_summary) {  
            const hoursSince = (Date.now() - new Date(child.last_session_date).getTime()) / (1000 * 60 * 60);  
  
            if (hoursSince < 24) {  
                // CACHE HIT - Return immediately, no AI calls  
                return {  
                    response: formatCachedSummary(  
                        child.last_session_summary,  
                        child.last_session_date,  
                        child.child_name  
                    ),  
                    intent: 'LEARNING',  
                    source: 'cache'  
                };  
            }  
        }  
    }  
  
    // CACHE MISS - Proceed to full Hybrid RAG  
    // ... rest of handler  
}
```

7.2 Admin Insights Cache

See Section 8 for weekly pre-computed insights.

7.3 Cache Summary

Cache Type	Trigger	TTL	Savings
Parent Session Summary	Recall.ai webhook	24 hours	40% of parent RAG calls
Admin Insights	Weekly cron (QStash)	7 days	90% of admin insight queries

8. Admin Weekly Insights

8.1 Overview

Pre-computed insights reduce cost and improve response time for complex Admin queries.

Schedule: Every Sunday at 2:00 AM IST (via QStash)

Storage: `admin_insights` table

8.2 Database Schema

sql

```
CREATE TABLE IF NOT EXISTS admin_insights (  
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),  
  insight_type TEXT NOT NULL UNIQUE,  
  insight_data JSONB NOT NULL,  
  computed_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
  valid_until TIMESTAMP WITH TIME ZONE,  
  
  CONSTRAINT valid_insight_type CHECK (  
    insight_type IN ('at_risk_children', 'common_struggles',  
      'coach_effectiveness', 'learning_trends')  
  )  
);  
  
CREATE INDEX IF NOT EXISTS idx_admin_insights_type ON admin_insights(insight_type);  
CREATE INDEX IF NOT EXISTS idx_admin_insights_valid ON admin_insights(valid_until);
```

8.3 Insight Definitions

8.3.1 At-Risk Children

Purpose: Identify children showing signs of disengagement

sql

-- Query

```
SELECT
  c.id,
  c.child_name,
  p.name as parent_name,
  p.email as parent_email,
  p.phone as parent_phone,
  co.name as coach_name,
  MAX(ss.scheduled_date) as last_session_date,
  EXTRACT(DAY FROM NOW() - MAX(ss.scheduled_date)) as days_since_session
FROM children c
JOIN parents p ON c.parent_id = p.id
LEFT JOIN coaches co ON c.assigned_coach_id = co.id
LEFT JOIN scheduled_sessions ss ON c.id = ss.child_id AND ss.status = 'completed'
WHERE c.status = 'enrolled'
GROUP BY c.id, c.child_name, p.name, p.email, p.phone, co.name
HAVING MAX(ss.scheduled_date) < NOW() - INTERVAL '14 days'
      OR MAX(ss.scheduled_date) IS NULL
ORDER BY days_since_session DESC NULLS FIRST;
```

Stored Format:

json

```
{
  "insight_type": "at_risk_children",
  "insight_data": {
    "children": [
      {
        "id": "uuid-1",
        "child_name": "Sima",
        "parent_name": "Priya",
        "parent_email": "priya@email.com",
        "coach_name": "Rucha",
        "days_since_session": 18,
        "risk_reason": "No session in 18 days"
      }
    ],
    "total_count": 5,
    "summary": "5 children haven't had sessions in 2+ weeks. Consider proactive parent outreach."
  }
}
```

8.3.2 Common Struggles

Purpose: Identify platform-wide learning challenges for curriculum improvement

sql

-- Query

```
SELECT
  event_data->>'focus_area' as focus_area,
  COUNT(*) as session_count,
  COUNT(DISTINCT child_id) as children_count,
  AVG(CASE
    WHEN event_data->>'engagement_level' = 'high' THEN 3
    WHEN event_data->>'engagement_level' = 'medium' THEN 2
    ELSE 1
  END) as avg_engagement
FROM learning_events
WHERE event_type = 'session'
  AND created_at > NOW() - INTERVAL '7 days'
  AND event_data->>'focus_area' IS NOT NULL
GROUP BY event_data->>'focus_area'
ORDER BY session_count DESC
LIMIT 10;
```

8.3.3 Coach Effectiveness

Purpose: Compare coach outcomes for quality monitoring

sql

-- Query

SELECT

```
co.id as coach_id,
co.name as coach_name,
COUNT(DISTINCT le.child_id) as student_count,
COUNT(le.id) as session_count,
AVG(CASE
  WHEN le.event_data->>'progress_rating' = 'significant_improvement' THEN 4
  WHEN le.event_data->>'progress_rating' = 'improved' THEN 3
  WHEN le.event_data->>'progress_rating' = 'same' THEN 2
  WHEN le.event_data->>'progress_rating' = 'declined' THEN 1
END) as avg_progress_score,
AVG(CASE
  WHEN le.event_data->>'engagement_level' = 'high' THEN 3
  WHEN le.event_data->>'engagement_level' = 'medium' THEN 2
  ELSE 1
END) as avg_engagement_score
FROM coaches co
LEFT JOIN learning_events le ON co.id = le.coach_id
AND le.event_type = 'session'
AND le.created_at > NOW() - INTERVAL '30 days'
WHERE co.status = 'active'
GROUP BY co.id, co.name
ORDER BY avg_progress_score DESC NULLS LAST;
```

8.3.4 Learning Trends

Purpose: Track platform-wide progress over time

sql

-- Query

SELECT

```
date_trunc('week', event_date) as week,
COUNT(*) as assessment_count,
AVG((event_data->>'score')::numeric) as avg_score,
AVG((event_data->>'wpm')::numeric) as avg_wpm,
AVG((event_data->>'fluency_rating')::numeric) as avg_fluency
FROM learning_events
WHERE event_type = 'assessment'
AND event_date > NOW() - INTERVAL '8 weeks'
GROUP BY week
ORDER BY week DESC;
```

8.4 QStash Cron Job

typescript

```
// /api/cron/weekly-insights/route.ts
```

```
import { verifySignatureAppRouter } from '@upstash/qstash/dist/nextjs';
```

```
async function handler(req: Request) {
```

```
  console.log('🔄 Starting weekly insights computation...');
```

```
  const insights = [
```

```
    { type: 'at_risk_children', query: AT_RISK_QUERY, aiSummary: true },
```

```
    { type: 'common_struggles', query: COMMON_STRUGGLES_QUERY, aiSummary: true },
```

```
    { type: 'coach_effectiveness', query: COACH_EFFECTIVENESS_QUERY, aiSummary: true },
```

```
    { type: 'learning_trends', query: LEARNING_TRENDS_QUERY, aiSummary: true },
```

```
  ];
```

```
  for (const insight of insights) {
```

```
    // Execute query
```

```
    const { data: rawData } = await supabase.rpc('run_insight_query', {  
      query_text: insight.query  
    });
```

```
    // Generate AI summary if needed
```

```
    let summary = "";
```

```
    if (insight.aiSummary) {  
      summary = await generateInsightSummary(insight.type, rawData);  
    }
```

```
    // Upsert to admin_insights
```

```
    await supabase
```

```
      .from('admin_insights')
```

```
      .upsert({  
        insight_type: insight.type,  
        insight_data: {  
          raw_data: rawData,  
          summary: summary,  
          generated_at: new Date().toISOString()  
        },  
        computed_at: new Date().toISOString(),  
        valid_until: new Date(Date.now() + 7 * 24 * 60 * 60 * 1000).toISOString() // 7 days  
      }, {
```

```
        onConflict: 'insight_type'  
      });
```

```
    console.log('✅ Computed: ${insight.type}');
```

```
  return Response.json({ success: true, computed: insights.length });
```

```
}  
  
export const POST = verifySignatureAppRouter(handler);
```

QStash Schedule Setup:

```
bash  
  
# Schedule weekly cron via QStash dashboard or API  
# URL: https://yestoryd.com/api/cron/weekly-insights  
# Schedule: 0 2 * * 0 (Every Sunday at 2 AM)
```

9. Technical Specifications

9.1 AI Models

Purpose	Model	Tokens	Cost per Call
Intent Classification (Tier 1)	Gemini 2.5 Flash Lite	~120	₹0.001
Embedding Generation	text-embedding-004	~200	₹0.002
Parent Response	Gemini 2.5 Flash Lite	~400	₹0.008
Coach Response (with CoT)	Gemini 2.5 Flash Lite	~650	₹0.012
Session Analysis (Recall.ai)	Gemini 2.5 Flash Lite	~800	₹0.015

9.2 Database Tables

Table	Role in rAI
learning_events	Primary RAG data source (with HNSW embeddings)
children	Child profiles + session summary cache
coaches	Coach profiles, assignments
parents	Parent accounts
scheduled_sessions	Session data for schedule queries
site_settings	Program info (price, duration)

Table	Role in rAI
admin_insights	Pre-computed weekly insights
recall_bot_sessions	Recall.ai bot tracking

9.3 Vector Search: HNSW Configuration

Why HNSW over IVFFlat:

Factor	IVFFlat	HNSW
Query accuracy	85-90% recall	95-99% recall
Incremental updates	Degrades over time	Handles well
Build time	Faster	Slower (one-time)
Maintenance	Needs periodic retraining	Self-maintaining

Index Creation:

```

sql

-- Drop old IVFFlat index if exists
DROP INDEX IF EXISTS idx_learning_events_embedding;

-- Create HNSW index for high-accuracy semantic search
CREATE INDEX idx_learning_events_embedding_hnsw
ON learning_events
USING hnsw (embedding vector_cosine_ops)
WITH (m = 16, ef_construction = 64);

-- Set search accuracy parameter (higher = more accurate, slightly slower)
-- Can be set per-session or globally
SET hnsw.ef_search = 100;

```

HNSW Parameters Explained:

- `m = 16`: Number of connections per layer (balance between accuracy and memory)
- `ef_construction = 64`: Build-time accuracy (higher = better index, slower build)
- `ef_search = 100`: Query-time accuracy (higher = better recall, slightly slower)

9.4 Hybrid Search Function

sql

```

CREATE OR REPLACE FUNCTION hybrid_match_learning_events(
  query_embedding vector(768),
  filter_child_id uuid DEFAULT NULL,
  filter_coach_id uuid DEFAULT NULL,
  filter_date_from timestamp DEFAULT NULL,
  filter_date_to timestamp DEFAULT NULL,
  filter_event_type text DEFAULT NULL,
  filter_keywords text[] DEFAULT NULL,
  match_threshold float DEFAULT 0.4,
  match_count int DEFAULT 15
) RETURNS TABLE (
  id uuid,
  child_id uuid,
  coach_id uuid,
  event_type text,
  event_date timestampz,
  event_data jsonb,
  ai_summary text,
  content_for_embedding text,
  similarity float,
  keyword_boost float,
  final_score float
) AS $$
BEGIN
  RETURN QUERY
  SELECT
    le.id,
    le.child_id,
    le.coach_id,
    le.event_type,
    le.event_date,
    le.event_data,
    le.ai_summary,
    le.content_for_embedding,
    1 - (le.embedding <=> query_embedding) as similarity,
    -- =====
    -- KEYWORD BOOST: Capped to prevent overwhelming semantic scores
    -- =====
    -- Vector similarity typically ranges 0.6-0.85 for relevant content
    -- Uncapped boost could overpower semantic meaning
    -- Solution: 0.05 per keyword match, max 0.25 total
    LEAST(
      COALESCE(
        (SELECT COUNT(*)::float * 0.05
         FROM unnest(filter_keywords) kw
         WHERE le.content_for_embedding ILIKE '%' || kw || '%'),

```

```
0
),
0.25 -- Hard cap: max boost is 0.25
) as keyword_boost,
-- Final score = similarity + capped keyword boost
(1 - (le.embedding <=> query_embedding)) +
LEAST(
  COALESCE(
    (SELECT COUNT(*)::float * 0.05
     FROM unnest(filter_keywords) kw
     WHERE le.content_for_embedding ILIKE '%' || kw || '%'),
    0
  ),
  0.25
) as final_score
FROM learning_events le
WHERE
  -- Role-based filtering (CRITICAL for security)
  (filter_child_id IS NULL OR le.child_id = filter_child_id)
  AND (filter_coach_id IS NULL OR le.coach_id = filter_coach_id)
  -- Date filtering
  AND (filter_date_from IS NULL OR le.event_date >= filter_date_from)
  AND (filter_date_to IS NULL OR le.event_date <= filter_date_to)
  -- Event type filtering
  AND (filter_event_type IS NULL OR le.event_type = filter_event_type)
  -- Similarity threshold
  AND 1 - (le.embedding <=> query_embedding) > match_threshold
ORDER BY final_score DESC
LIMIT match_count;
END;
$$ LANGUAGE plpgsql STABLE;
```

Keyword Boost Logic Explained:

Keywords Matched	Old Boost	New Boost (Capped)
1 keyword	0.10	0.05
2 keywords	0.20	0.10
3 keywords	0.30	0.15
5 keywords	0.50	0.25 (capped)
10 keywords	1.00	0.25 (capped)

This ensures semantic similarity remains the primary ranking factor while keywords provide helpful tie-breaking.

9.5 API Endpoint Structure

```
typescript

// POST /api/chat

interface ChatRequest {
  message: string;
  userRole: 'parent' | 'coach' | 'admin';
  userId: string;
  childId?: string;
  chatHistory?: ChatMessage[];
}

interface ChatResponse {
  response: string;
  intent: 'LEARNING' | 'OPERATIONAL' | 'SCHEDULE' | 'OFF_LIMITS';
  source: 'cache' | 'rag' | 'sql' | 'redirect';
  needsChildSelection?: boolean;
  children?: { id: string; name: string }[];
  debug?: {
    tier0Match: boolean;
    cacheHit: boolean;
    eventsRetrieved: number;
    latencyMs: number;
  };
}
```

9.6 Streaming UI for Slow Queries

Problem: Tier 1 queries (Hybrid Search + CoT Gemini) take 3-6 seconds. Users may think the app froze.

Solution: Implement streaming responses with progressive feedback.

9.6.1 Streaming API Route

```
typescript
```

```
// /api/chat/route.ts
```

```
import { StreamingTextResponse, Message } from 'ai';
```

```
export async function POST(req: Request) {
```

```
  const { message, userRole, userId, childId, chatHistory } = await req.json();
```

```
  // =====
```

```
  // TIER 0: Instant response (no streaming needed)
```

```
  // =====
```

```
  const tier0Intent = tier0Router(message);
```

```
  if (tier0Intent) {
```

```
    const result = await handleIntent(tier0Intent, { message, userRole, userId, childId });
```

```
    return Response.json(result);
```

```
  }
```

```
  // =====
```

```
  // TIER 1: Streaming response for slow queries
```

```
  // =====
```

```
  const encoder = new TextEncoder();
```

```
  const stream = new ReadableStream({
```

```
    async start(controller) {
```

```
      // Send immediate feedback
```

```
      controller.enqueue(encoder.encode(
```

```
        JSON.stringify({
```

```
          type: 'status',
```

```
          message: '🔍 Analyzing your question...'
```

```
        }) + '\n'
```

```
      ));
```

```
    // Step 1: Classify intent
```

```
    const { intent, entities } = await classifyIntent(message, userRole);
```

```
    controller.enqueue(encoder.encode(
```

```
      JSON.stringify({
```

```
        type: 'intent',
```

```
        intent,
```

```
        entities
```

```
      }) + '\n'
```

```
    ));
```

```
    // Step 2: Execute handler with progress updates
```

```
    if (intent === 'LEARNING') {
```


```
      // Check cache first
```

```
      if (isRecentSessionQuery(message) && childId) {
```


```
        const cached = await checkSessionCache(childId);
```

```
if (cached) {
  controller.enqueue(encoder.encode(
    JSON.stringify({
      type: 'response',
      response: cached,
      source: 'cache'
    }) + '\n'
  ));
  controller.close();
  return;
}
```

// RAG search

```
controller.enqueue(encoder.encode(
  JSON.stringify({
    type: 'status',
    message: ' Searching learning history...'
  }) + '\n'
));
```

```
const events = await hybridSearch(message, childId, userRole, userId);
```

```
controller.enqueue(encoder.encode(
  JSON.stringify({
    type: 'status',
    message: ` Found ${events.length} relevant events. Generating response...`
  }) + '\n'
));
```

// Stream Gemini response

```
const geminiStream = await generateStreamingResponse(events, message, userRole);
```

```
for await (const chunk of geminiStream) {
  controller.enqueue(encoder.encode(
    JSON.stringify({
      type: 'chunk',
      content: chunk
    }) + '\n'
  ));
}
```

```
controller.enqueue(encoder.encode(
  JSON.stringify({
    type: 'done',
    source: 'rag'
  }) + '\n'
));
```

```

));
} else {
  // OPERATIONAL, SCHEDULE, OFF_LIMITS - usually fast
  const result = await handleIntent(intent, { message, userRole, userId, childId });
  controller.enqueue(encoder.encode(
    JSON.stringify({
      type: 'response',
      ...result
    }) + '\n'
  ));
}

controller.close();
}
});

return new Response(stream, {
  headers: {
    'Content-Type': 'text/event-stream',
    'Cache-Control': 'no-cache',
    'Connection': 'keep-alive',
  },
});
}

// Gemini streaming helper
async function* generateStreamingResponse(
  events: LearningEvent[],
  message: string,
  userRole: string
): AsyncGenerator<string> {
  const model = genAI.getGenerativeModel({ model: 'gemini-1.5-flash-8b' });

  const prompt = userRole === 'coach'
    ? buildCoachCoTPrompt(events, message)
    : buildParentPrompt(events, message);

  const result = await model.generateContentStream(prompt);

  for await (const chunk of result.stream) {
    const text = chunk.text();
    if (text) {
      yield text;
    }
  }
}

```

9.6.2 Frontend Chat Component

typescript

```
// components/ChatWidget.tsx
```

```
import { useState, useRef } from 'react';

export function ChatWidget({ userRole, userId, childId }) {
  const [messages, setMessages] = useState<Message[]>([]);
  const [isLoading, setIsLoading] = useState(false);
  const [status, setStatus] = useState("");
  const [streamingContent, setStreamingContent] = useState("");

  async function handleSubmit(message: string) {
    setIsLoading(true);
    setStatus("");
    setStreamingContent("");

    // Add user message
    setMessages(prev => [...prev, { role: 'user', content: message }]);

    try {
      const response = await fetch('/api/chat', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ message, userRole, userId, childId }),
      });

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

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

        const lines = decoder.decode(value).split('\n').filter(Boolean);

        for (const line of lines) {
          const data = JSON.parse(line);

          switch (data.type) {
            case 'status':
              setStatus(data.message);
              break;

            case 'intent':
              // Optional: Show intent for debugging
              console.log('Intent:', data.intent);
              break;
          }
        }
      }
    }
  }
}
```

```

    case 'chunk':
      setStreamingContent(prev => prev + data.content);
      break;

    case 'response':
      // Non-streaming response (cache hit, SQL, etc.)
      setMessages(prev => [...prev, {
        role: 'assistant',
        content: data.response
      }]);
      break;

    case 'done':
      // Streaming complete - move accumulated content to messages
      if (streamingContent) {
        setMessages(prev => [...prev, {
          role: 'assistant',
          content: streamingContent
        }]);
        setStreamingContent("");
      }
      break;
  }
}

} catch (error) {
  console.error('Chat error:', error);
  setMessages(prev => [...prev, {
    role: 'assistant',
    content: 'Sorry, something went wrong. Please try again.'
  }]);
} finally {
  setIsLoading(false);
  setStatus("");
}
}

return (
  <div className="chat-widget">
    { /* Messages */ }
    <div className="messages">
      {messages.map((msg, i) => (
        <div key={i} className={`message ${msg.role}`}>
          {msg.content}
        </div>
      ))}
    </div>
  </div>
)

```

```

    { /* Streaming content */
    {streamingContent && (
      <div className="message assistant streaming">
        {streamingContent}
        <span className="cursor">█ </span>
      </div>
    )}

    { /* Status indicator */
    {status && (
      <div className="status-indicator">
        <span className="spinner">⌛ </span>
        {status}
      </div>
    )}
  </div>

  { /* Input */
  <ChatInput onSubmit={handleSubmit} disabled={isLoading} />
</div>
);
}

```

9.6.3 Status Messages by Query Type

Query Type	Status Sequence	Total Time
Cache Hit	(none - instant)	<500ms
Tier 0 SQL	(none - instant)	<500ms
Tier 1 RAG	"🔍 Analyzing..." → "📄 Searching..." → "📖 Found X events..." → streaming	3-6s
Admin Insights	"🔍 Analyzing..." → "📊 Loading insights..."	1-2s

9.7 Security: Role-Based Filtering

CRITICAL: All queries MUST filter by role at the SQL level. This is non-negotiable.

typescript

```
function buildRoleFilter(userRole: string, userId: string): RoleFilter {
  switch (userRole) {
    case 'parent':
      // Parent can ONLY see their own children
      return {
        type: 'child_ids',
        query: `child_id IN (SELECT id FROM children WHERE parent_id = '${userId})'`
      };

    case 'coach':
      // Coach can ONLY see assigned children
      return {
        type: 'coach_filter',
        query: `child_id IN (SELECT id FROM children WHERE assigned_coach_id = '${userId})'`
      };

    case 'admin':
      // Admin sees all (no filter)
      return {
        type: 'none',
        query: 'TRUE'
      };

    default:
      throw new Error('Invalid user role');
  }
}
```

10. Cost Analysis

10.1 Per-Query Cost Breakdown

Query Path	Components	Total Cost
Tier 0 → SQL	Regex (₹0) + SQL (₹0) + Format (₹0.005)	₹0.005
Tier 0 → Cache	Regex (₹0) + Cache lookup (₹0)	₹0
Tier 1 → Cache	LLM classify (₹0.001) + Cache lookup (₹0)	₹0.001
Tier 1 → RAG (Parent)	LLM classify (₹0.001) + Embed (₹0.002) + Generate (₹0.008)	₹0.011
Tier 1 → RAG (Coach CoT)	LLM classify (₹0.001) + Embed (₹0.002) + Generate (₹0.012)	₹0.015

Query Path	Components	Total Cost
Tier 1 → OFF_LIMITS	LLM classify (₹0.001) + Template (₹0)	₹0.001

10.2 Query Distribution (Estimated)

User Type	LEARNING	OPERATIONAL	SCHEDULE	OFF_LIMITS	Avg Queries/Week
Parent	60%	20%	15%	5%	5
Coach	50%	25%	20%	5%	10
Admin	40%	45%	10%	5%	20

10.3 Optimization Impact

Optimization	Before	After	Savings
Tier 0 Regex	100% queries → LLM	35% skip LLM	35% of classification costs
Parent Cache	100% parent LEARNING → RAG	40% from cache	40% of parent RAG costs
HNSW Index	~85% recall	~95% recall	Better quality (not cost)
Hybrid Search	Inaccurate temporal queries	Accurate results	Better quality (not cost)
Coach CoT	Generic summaries	Pedagogical reasoning	Better quality (not cost)

10.4 Monthly Cost Projections (Optimized)

At 100 Children

Component	Queries/Month	Cost
Parents ($5 \times 4 \text{ weeks} \times 100$)	2,000	
└─ Cache hits (40%)	800	₹0
└─ Tier 0 (35% of remaining)	420	₹2
└─ RAG (65% of remaining)	780	₹9
Coaches ($10 \times 4 \text{ weeks} \times 5$)	200	
└─ Tier 0 (35%)	70	₹0.35
└─ RAG with CoT (50%)	100	₹1.5
└─ SQL (15%)	30	₹0.15
Admin	80	₹1
Weekly Insights (4 runs)	4	₹4
TOTAL		~₹18

At 1,000 Children

Component	Queries/Month	Cost
Parents	20,000	~₹80
Coaches (50 coaches)	2,000	~₹25
Admin	200	~₹3
Weekly Insights	4	₹16
TOTAL		~₹125

At 10,000 Children

Component	Queries/Month	Cost
Parents	200,000	~₹800
Coaches (500 coaches)	20,000	~₹250
Admin	500	~₹10
Weekly Insights	4	₹40
Session Analysis (Recall.ai)	30,000 sessions	~₹450
TOTAL		~₹1,550

10.5 Comparison: Before vs After Optimizations

Scale	Original Design	Optimized Design	Savings
100 children	~₹235	~₹18	92%
1,000 children	~₹1,400	~₹125	91%
10,000 children	~₹5,200	~₹1,550	70%

Note: At higher scale, Session Analysis (Recall.ai) becomes a larger portion of cost. This is unavoidable as it's tied to actual sessions conducted.

11. Implementation Roadmap

Phase 1: Data Pipeline (Day 1)

Goal: Get learning data flowing into `learning_events` with embeddings

Task	Priority	Status
Modify <code>/api/assessment/analyze</code> to save to <code>learning_events</code>	P0	<div></div>
Add embedding generation after assessment	P0	<div></div>
Update Recall.ai webhook to generate two outputs (coach + parent)	P0	<div></div>
Add <code>last_session_summary</code> column to children table	P0	<div></div>
Create HNSW index (replace IVFFlat)	P0	<div></div>
Create <code>hybrid_match_learning_events</code> function	P0	<div></div>
Test with real assessment → verify data appears	P0	<div></div>

Phase 2: Chat API Enhancement (Day 1-2)

Goal: Build the optimized intent-based routing system

Task	Priority	Status
Implement Tier 0 Regex Router	P0	<div></div>
Implement Tier 1 LLM Classifier	P0	<div></div>
Build parent cache check (before RAG)	P0	<div></div>
Build hybrid search with date/keyword filtering	P0	<div></div>
Build OPERATIONAL handler with SQL templates	P1	<div></div>
Build SCHEDULE handler	P1	<div></div>
Build OFF_LIMITS handler with redirects	P1	<div></div>
Add Coach Chain of Thought prompts	P1	<div></div>
Add conversation memory support	P1	<div></div>
Add multi-child disambiguation	P1	<div></div>

Phase 3: UI Integration (Day 2)

Goal: Working chat on all dashboards

Task	Priority	Status
Fix ChatWidget on parent dashboard	P0	<div></div>
Verify coach AI assistant page	P0	<div></div>
Add/verify chat on admin dashboard	P1	<div></div>
Test conversation memory in UI	P1	<div></div>
Add loading states and error handling	P1	<div></div>

Phase 4: Admin Insights (Day 3)

Goal: Weekly pre-computed insights for Admin

Task	Priority	Status
Create <code>admin_insights</code> table	P1	<div></div>
Build QStash cron endpoint	P1	<div></div>
Implement insight computation queries	P1	<div></div>
Add insight retrieval in chat handler	P1	<div></div>
Schedule cron in QStash dashboard	P1	<div></div>
Test insight queries	P1	<div></div>

Phase 5: Testing & Refinement (Day 3-4)

Task	Priority	Status
End-to-end test: Parent journey (cache hit)	P0	<div></div>
End-to-end test: Parent journey (RAG)	P0	<div></div>
End-to-end test: Coach journey (CoT)	P0	<div></div>
End-to-end test: Admin journey (insights)	P0	<div></div>
Test multi-child scenarios	P1	<div></div>
Test conversation memory	P1	<div></div>
Verify anti-hallucination	P0	<div></div>

Task	Priority	Status
Test Tier 0 regex patterns	P1	<div></div>
Test hybrid search date filtering	P1	<div></div>
Performance testing (latency)	P2	<div></div>

12. Appendix: Query Examples

12.1 Parent Query Examples

Query	Path	Source	Example Response
"How did today's session go?"	Tier 1 → Cache	Cached	"Here's what happened in Sima's session 2 hours ago: Sima practiced consonant blends today and showed great enthusiasm! Coach Rucha noted she's getting faster with 'bl' and 'cr' sounds. Try reading 'Blue's Clues' together this week to reinforce these skills."
"How is Sima doing overall?"	Tier 1 → RAG	Hybrid RAG	"Sima has been making excellent progress! Over the last 3 sessions, her fluency improved from 45 to 58 WPM. She's particularly strong with consonant blends now. The area to focus on next is 'th' sounds - you can practice with words like 'the', 'this', 'that' during reading time."
"When is the next session?"	Tier 0 → SQL	SQL	"Sima's next session is tomorrow (Monday) at 4:00 PM with Coach Rucha. Here's the Google Meet link: [link]"
"Who is my coach?"	Tier 0 → SQL	SQL	"Your coach is Rucha. You can connect with her during scheduled session times."
"What are my earnings?"	Tier 0 → Redirect	Template	"I don't have access to that information. Is there something about Sima's learning I can help with?"

12.2 Coach Query Examples

Query	Path	Source	Example Response
"How many students do I have?"	Tier 0 → SQL	SQL	"You have 8 active students assigned to you."
"Prepare me for tomorrow's	Tier 1 →	Hybrid RAG	"For tomorrow's session with Aarav:\n\n Learning Trajectory Analysis: \nAarav has been working on phonics for 4 sessions. He mastered 'sh' sounds quickly (2 sessions) but is taking longer with 'th' sounds (2 sessions, still struggling). This



Query	Path	Source	Example Response
session with Aarav"	RAG (CoT)		pattern suggests the voiced/voiceless consonant distinction is his specific challenge.\n\n What's Worked: \n- Visual cues (pointing to mouth position)\n- The minimal pairs game\n- Parent reported 4 days of practice this week\n\n Recommendation: \nSince Aarav already knows 'sh' well, use the 'sh' vs 'th' contrast method. Start with minimal pairs: 'ship/this', 'she/the', 'shore/Thor'. Build from his existing anchor point - this should accelerate progress.\n\n Homework assigned last session: Read 'The Three Bears' focusing on 'th' words."
"What's my schedule today?"	Tier 0 → SQL	SQL	"Today you have 3 sessions:\n• 10:00 AM - Sima (coaching) [Meet link]\n• 2:00 PM - Aarav (coaching) [Meet link]\n• 4:00 PM - Zara (coaching) [Meet link]"
"What phonics strategies work for struggling readers?"	Tier 1 → RAG (CoT)	Hybrid RAG + General	"Based on your students' data and general best practices:\n\n From your sessions: \n- Sima responded well to the phonics bingo game\n- Aarav learns faster with visual mouth positioning cues\n- Multi-sensory approaches (sand tray) worked for Zara\n\n General strategies for struggling readers: \n1. Orton-Gillingham approach: Systematic, multi-sensory\n2. Minimal pairs: Contrast similar sounds\n3. Decodable texts: Match reading level to phonics knowledge\n\nWould you like specific activities for any of your students?"

12.3 Admin Query Examples

Query	Path	Source	Example Response
"Which children are at risk?"	Tier 1 → Insights	Cached Insights	"5 children haven't had sessions in 2+ weeks:\n\n1. Sima (18 days) - Coach: Rucha, Parent: Priya\n2. Aarav (15 days) - Coach: Priya, Parent: Rahul\n3. Zara (14 days) - Coach: Meera, Parent: Sneha\n4. Arjun (14 days) - Coach: Rucha, Parent: Vikram\n5. Ananya (14 days) - Coach: Priya, Parent: Deepa\n\n Recommendation: Consider proactive parent outreach. 2 of these families also have pending payments."
"How many enrollments this month?"	Tier 0 → SQL	SQL	"Enrollment metrics for December 2025:\n• This month: 42 new enrollments\n• This week: 12 new enrollments\n• Total active: 156 children\n• Conversion rate: 8.2% (from assessments)"
"What are children struggling with most?"	Tier 1 → Insights	Cached Insights	"Top struggle areas this week (based on 89 sessions):\n\n1. 'th' sounds - 23% of sessions (most common)\n2. Long vowels - 18% of sessions\n3. Digraphs - 15% of sessions\n4. Consonant blends - 12% of sessions\n\n Insight: 'th' sounds have been the #1 struggle for 3 consecutive weeks. Consider adding more 'th' resources to the curriculum or a specialized mini-course."
"What's the revenue this	Tier 0 → Redirect	Template	"For revenue and financial data, please check the Revenue dashboard. Would you like to know about learning metrics instead?"

Query	Path	Source	Example Response
month?"			

Document Approval

Role	Name	Date	Signature
Product Owner	Amit Kumar Rai		
Technical Lead	Claude	Dec 21, 2025	

Summary of v2.0 Optimizations

#	Optimization	Impact
1	Parent Session Caching	40% reduction in parent RAG calls
2	HNSW Index	95%+ recall accuracy (vs 85% with IVFFlat)
3	Hybrid Search	Accurate temporal and keyword queries
4	Coach Chain of Thought	Pedagogical reasoning, not just summaries
5	Tier 0 Regex Router	35% of queries skip LLM classification
6	Single Gemini Call (Recall.ai)	Generate coach analysis + parent summary together

Total Cost Reduction: ~70-90% depending on scale

Document Version: 2.0 (Optimized)
Last Updated: December 21, 2025
Status: Ready for Implementation