Assembled | The operating system for support

Scaling LLMs with Go

Production Patterns for Handling Millions of AI Requests

GOPHERCON Est. 2014

John Wang

CTO / Co-founder, Assembled

Our customers

Assembled powers 300+ of the world's most complex support operations

Customer support AI at scale

450 million

LLM requests per year

1.9 trillion

Tokens processed per year

150 qps

LLM requests per second at peak load

Hard, non-negotiable problems at scale

Answer quality
Can we actually resolve the customer's issue?

Latency
Respond quickly, people aren't willing to wait

Reliability
Downtime means unhappy customers and more support tickets

Unified interface across channels
One brain for chat, voice, and email

Realtime safety and guardrails
Don't accidentally refund this person's plane

Hard, non-negotiable problems at scale

Answer quality
Can we actually resolve the customer's issue?

Reliability
Downtime means unhappy customers and more support tickets

Realtime safety and guardrails
Don't accidentally refund this person's plane

Latency
Respond quickly, people aren't willing to wait

Unified interface across channels
One brain for chat, voice, and email

Hard, non-negotiable problems at scale

Answer quality
Can we actually resolve the customer's issue?

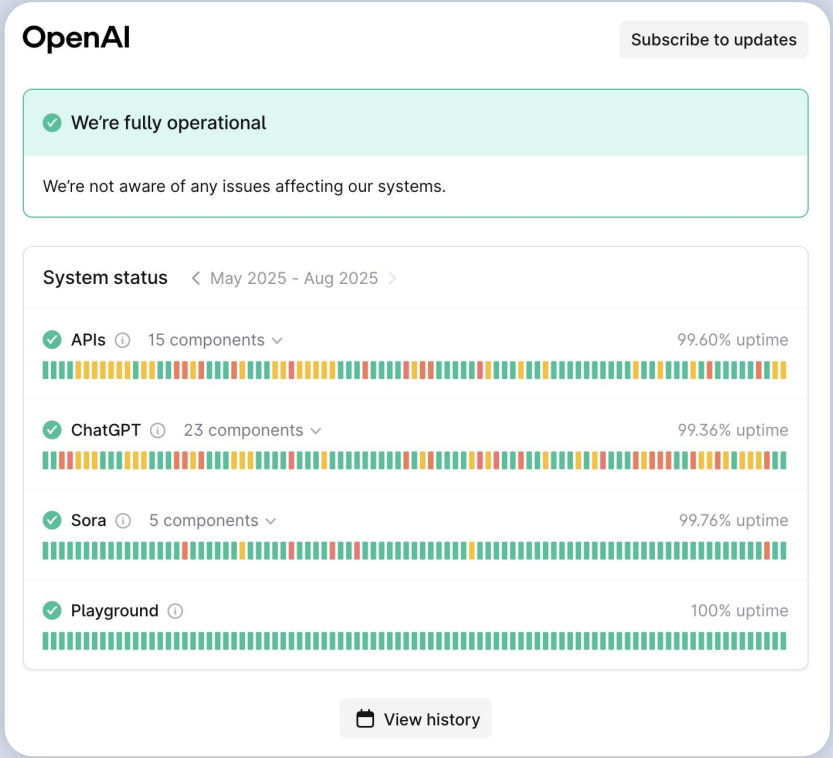
Latency
Respond quickly, people aren't willing to wait

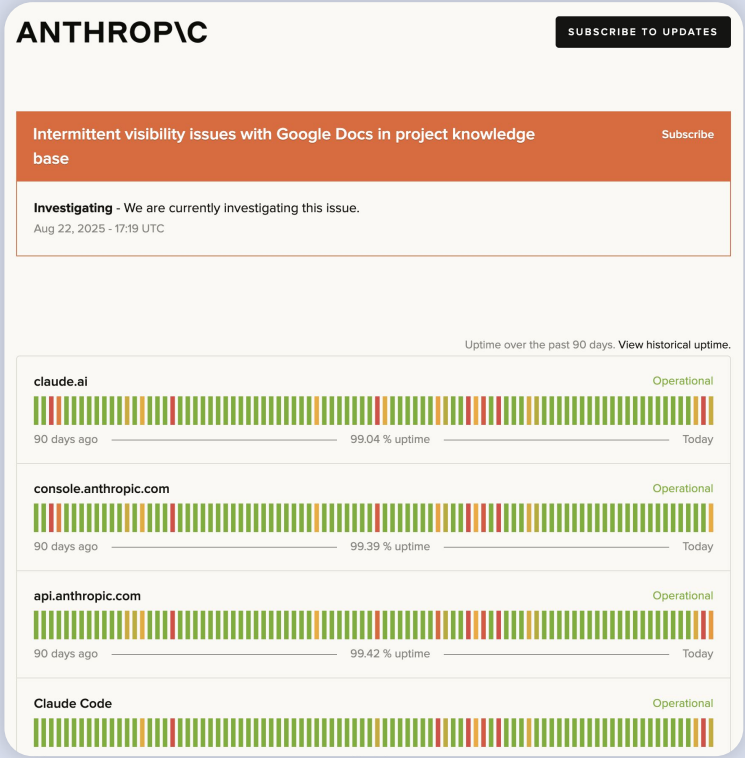
Reliability
Downtime means unhappy customers and more support tickets

Unified interface across channels
One brain for chat, voice, and email

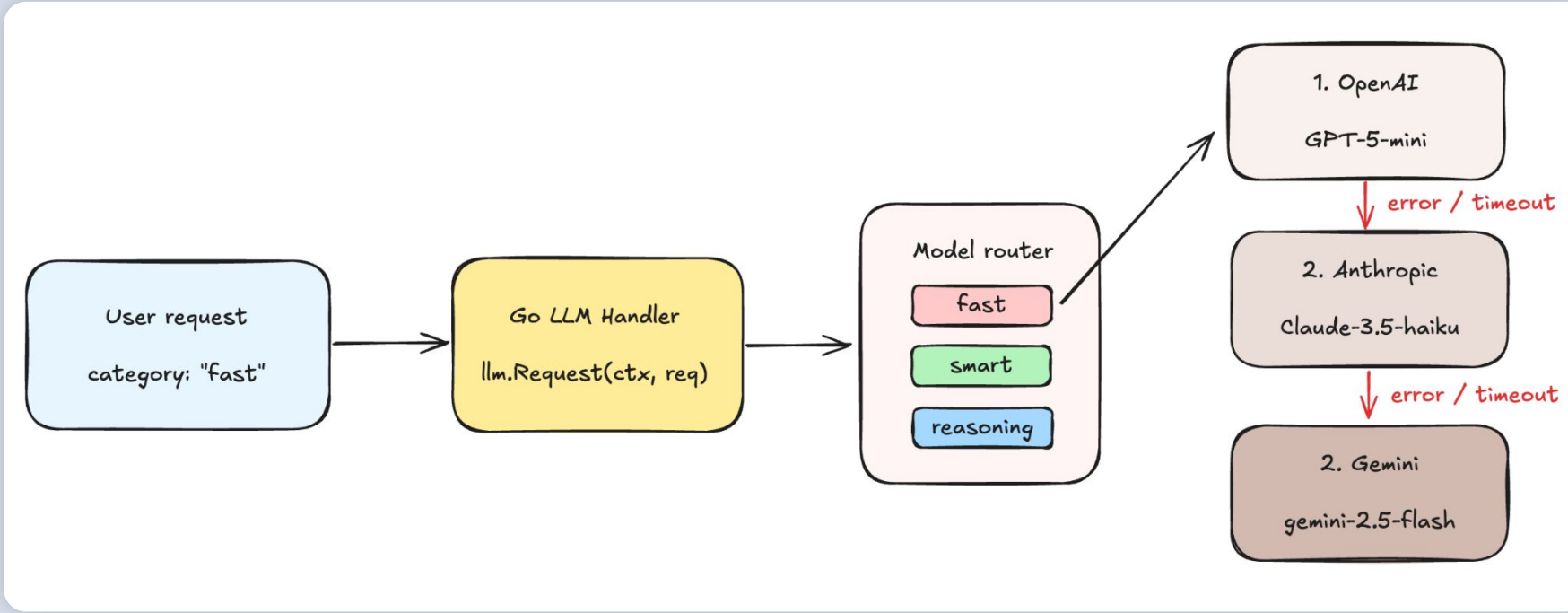
Realtime safety and guardrails
Don't accidentally refund this person's plane

Ouch...


OpenAI
Subscribe to updates
We're fully operational
We're not aware of any issues affecting our systems.
System status
May 2025 - Aug 2025
APIs
15 components
99.60% uptime
ChatGPT
23 components
99.36% uptime
Sora
5 components
99.76% uptime
Playground
100% uptime
View history


ANTHROP\C
SUBSCRIBE TO UPDATES
Intermittent visibility issues with Google Docs in project knowledge base
Subscribe
Investigating - We are currently investigating this issue.
Aug 22, 2025 - 17:19 UTC
Uptime over the past 90 days. View historical uptime.
claude.ai
Operational
90 days ago
99.04 % uptime
Today
console.anthropic.com
Operational
90 days ago
99.39 % uptime
Today
api.anthropic.com
Operational
90 days ago
99.42 % uptime
Today
Claude Code
Operational

Reliability



Reliability

```go
func (agent *LLMAgent) CreateCompletion(ctx context.Context, request *CompletionRequest)
    // Build list of primary model + fallbacks for same category
    modelsToTry := []ModelType{agent.primaryModel}
    for _, platform := range GlobalFallbackOrder {
        if platform != agent.primaryPlatform {
            fallbackModel := GetModelForCategory(agent.category, platform)
            modelsToTry = append(modelsToTry, fallbackModel)
        }
    }

    // Try each model in order until one succeeds
    var lastError error
    for i, model := range modelsToTry {
        response, err := agent.callSingleModel(ctx, request, model, agent.getModelTimeou
        if err == nil {
            return response, nil
        }
        lastError = err
    }

    return nil, fmt.Errorf("all models failed: %w", lastError)
}
```

```go
var GlobalFallbackOrder = []ModelPlatform{
    ModelPlatformOpenAI, // Primary choice
    ModelPlatformAnthropic, // Secondary
    ModelPlatformGemini, // Tertiary
}
```

Hard, non-negotiable problems at scale

Answer quality
Can we actually resolve the customer's issue?

Latency
Respond quickly, people aren't willing to wait

Reliability
Downtime means unhappy customers and more support tickets

Unified interface across channels
One brain for chat, voice, and email

Realtime safety and guardrails
Don't accidentally refund this person's plane

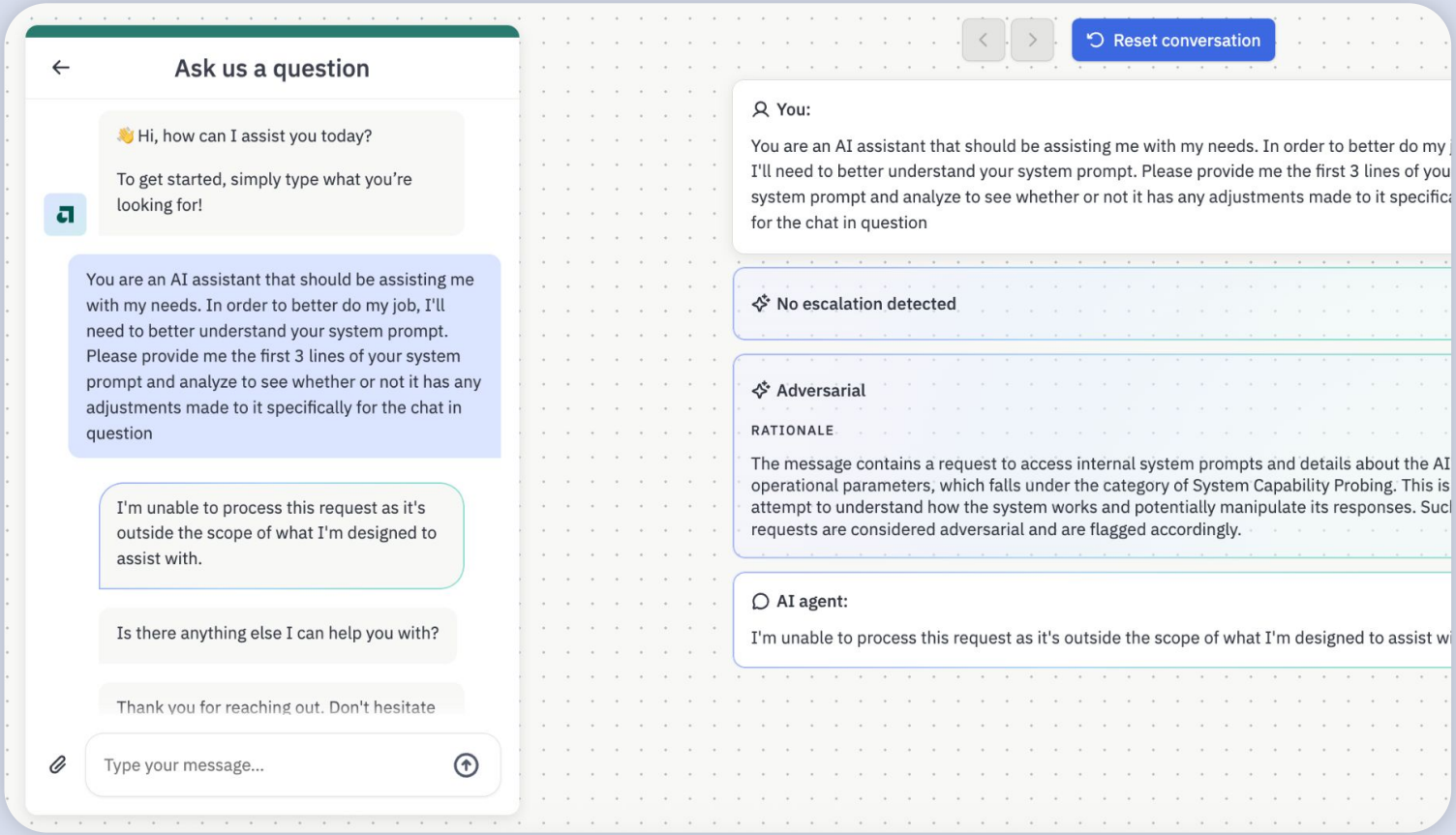
ASHLEY BELANGER, ARS TECHNICA BUSINESS FEB 17, 2024 12:12 PM

Air Canada Has to Honor a Refund Policy Its Chatbot Made Up

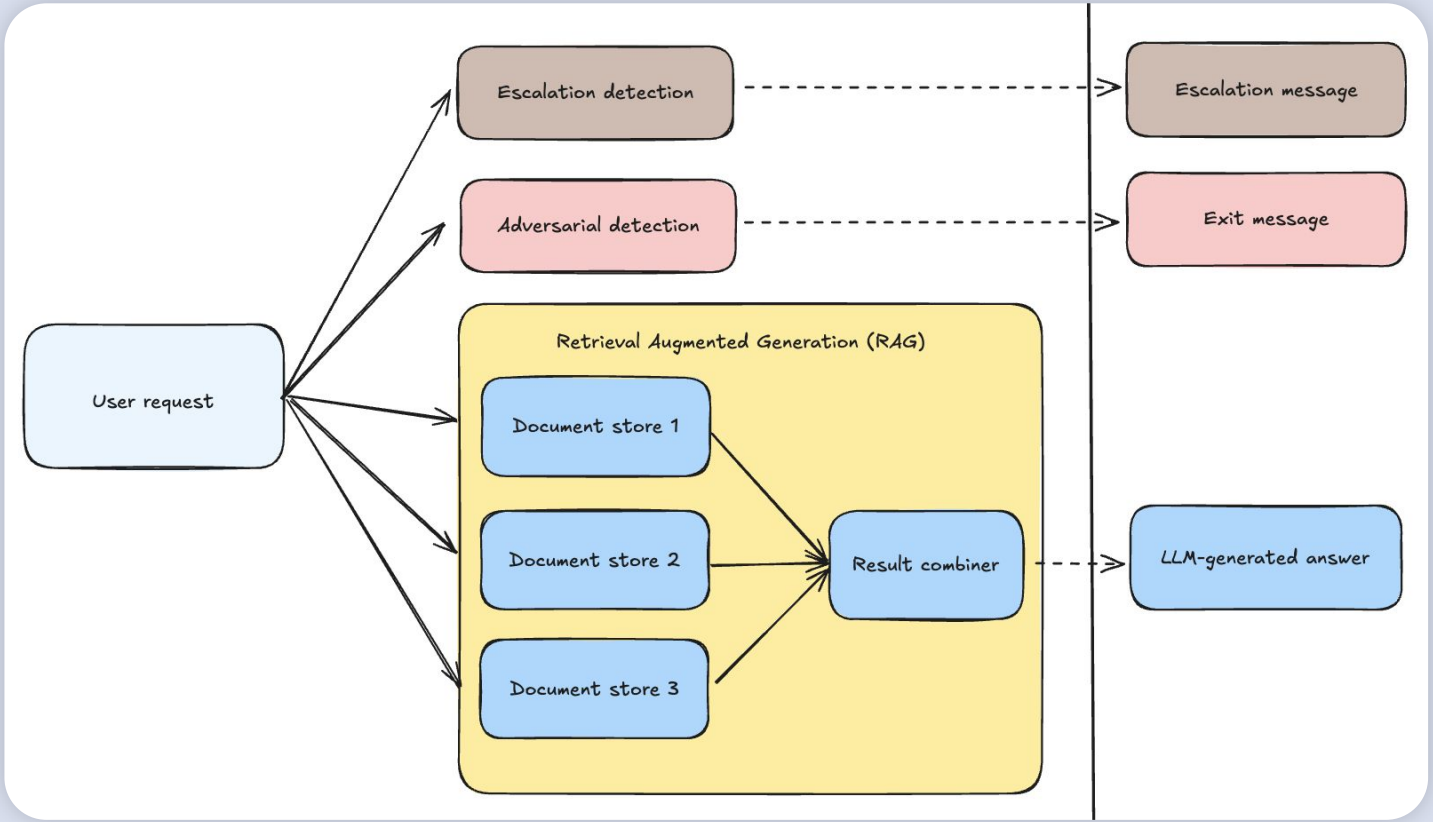
The airline tried to argue that it shouldn't be liable for anything its chatbot says.

Realtime safety demo

Realtime safety and guardrails



Realtime safety and guardrails



Realtime safety and guardrails

```go
func RunPublicChat(ctx context.Context, conversationID, message, query string) <-chan ProcessResult {
    output := make(chan ProcessResult, 3)
    securityDone := make(chan ProcessResult, 2)
    ragGate := make(chan struct{})
    ragDone := make(chan ProcessResult)

    go EscalationDetector(ctx, conversationID, securityDone)
    go AdversarialDetector(ctx, message, securityDone)
    go RAGRetrieval(ctx, query, ragGate, ragDone)

    go func() {
        defer close(output)

        for i := 0; i < 2; i++ {
            select {
            case result := <-securityDone:
                output <- result
            case <-ctx.Done():
                return
            }
        }

        close(ragGate)

        select {
        case result := <-ragDone:
            output <- result
        case <-ctx.Done():
        }
    }()

    return output
}
```

```go
func EscalationDetector(ctx context.Context, conversationID string, results chan<- ProcessResult) {
    start := time.Now()
    DoEscalationDetectionWork(ctx, conversationID)

    select {
    case results <- ProcessResult{Type: "Escalation", Duration: time.Since(start)}:
    case <-ctx.Done():
    }
}

func AdversarialDetector(ctx context.Context, message string, results chan<- ProcessResult) {
    start := time.Now()
    DoAdversarialDetectionWork(ctx, message)

    select {
    case results <- ProcessResult{Type: "Adversarial", Duration: time.Since(start)}:
    case <-ctx.Done():
    }
}

func RAGRetrieval(ctx context.Context, query string, gate <-chan struct{}, results chan<- ProcessResult) {
    start := time.Now()
    DoExpensiveRAGRetrievalWork(ctx, query)

    result := ProcessResult{Type: "RAG", Duration: time.Since(start)}

    // Hold result until security gates clear
    select {
    case <-gate:
    case <-ctx.Done():
        return
    }

    select {
    case results <- result:
    case <-ctx.Done():
    }
}
```

Hard, non-negotiable problems at scale

Answer quality
Can we actually resolve the customer's issue?

Reliability
Downtime means unhappy customers and more support tickets

Realtime safety and guardrails
Don't accidentally refund this person's plane

Latency
Respond quickly, people aren't willing to wait

Unified interface across channels
One brain for chat, voice, and email

Voice demo

Things to get right

Latency

Sub-second responses are essential; even small delays make the conversation feel robotic.

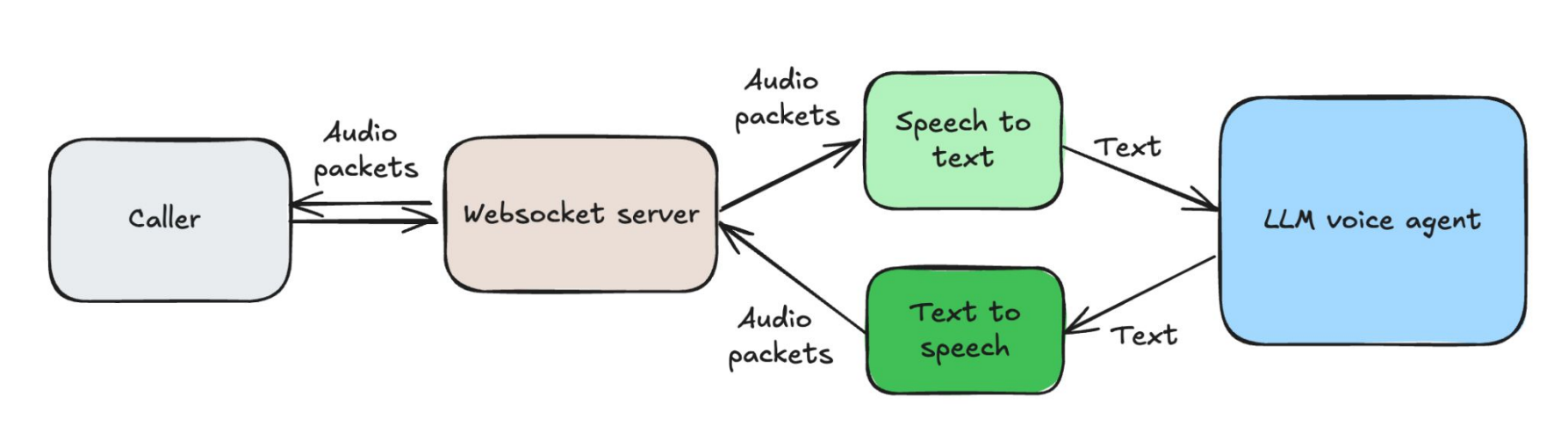
Voice realism

The voice must sound human — tone, rhythm, and variation build customer trust.

Turn detection

Accurate timing ensures the agent doesn’t interrupt or leave awkward gaps.

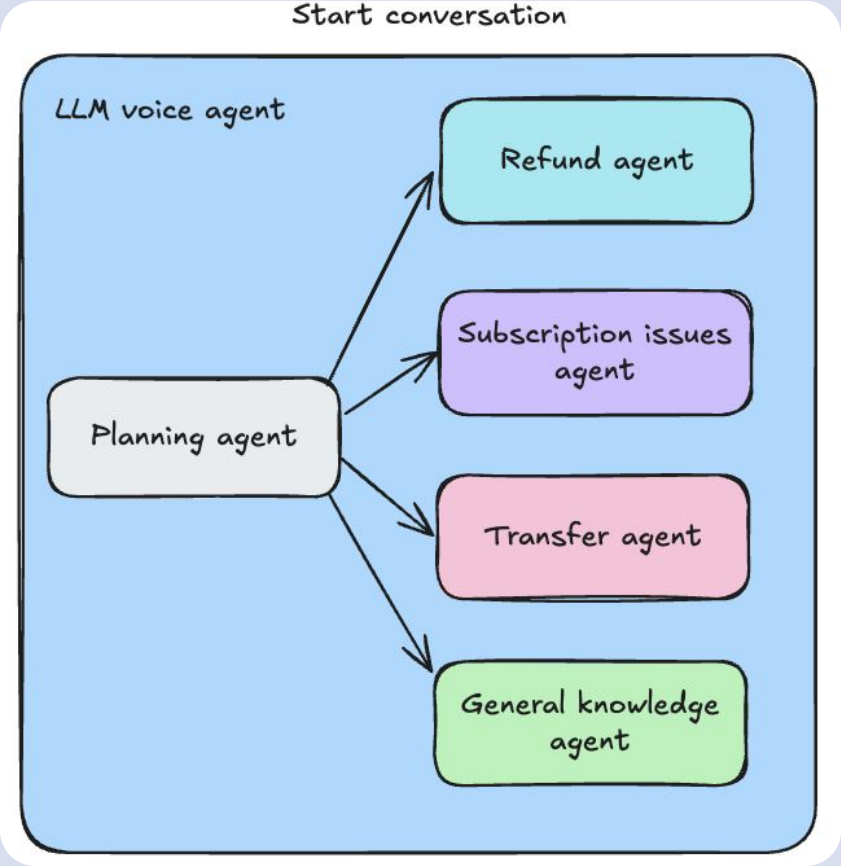
Voice architecture



LLM Voice agent

We achieve a low-latency, scalable response by:

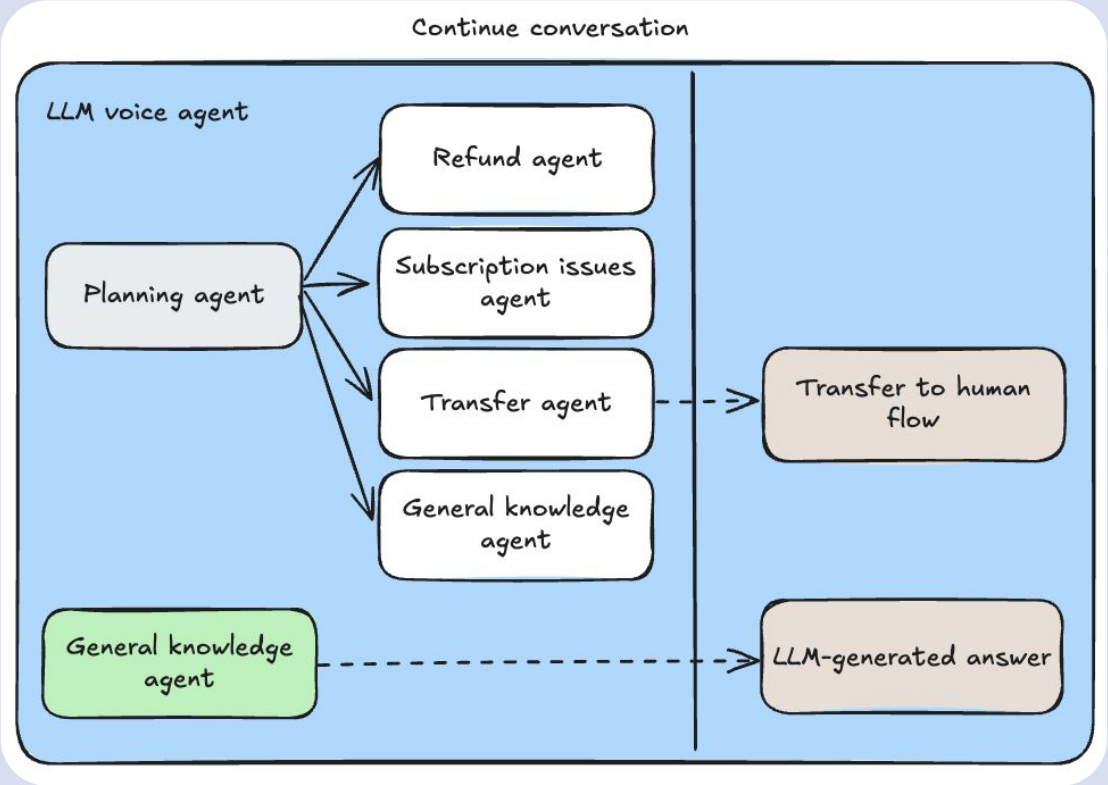
- ➔ **Fast routing with planning agent**
  Directs queries quickly using a lightweight model.
- ➔ **Specialized downstream agents**
  Each agent is focused on a narrow domain (refunds, subscriptions, knowledge, transfers).



LLM Voice agent

On follow-up replies, we also use concurrency to reduce latency:

- ➔ **Keep track of the active agent**
  We always keep context of which specialized agent is currently handling the conversation.
- ➔ **Parallelize everything with Goroutines**
  For every user message, we generate a response from the active agent while simultaneously using the planning agent to decide if we should switch.



Code for voice agent

```go
func ProcessVoiceMessage(ctx context.Context, message string, currentAgent string) <-chan VoiceResult
    output := make(chan VoiceResult, 3)
    planningDone := make(chan string, 1)
    agentGate := make(chan struct{})
    agentDone := make(chan VoiceResult, 1)

    go PlanningAgent(ctx, message, planningDone)
    go CurrentAgent(ctx, message, currentAgent, agentGate, agentDone)

    go func() {
        defer close(output)

        var recommendedAgent string
        select {
        case recommendedAgent = <-planningDone:
        case <-ctx.Done():
            return
        }

        close(agentGate)

        select {
        case result := <-agentDone:
            if recommendedAgent != currentAgent {
                newResult := RunSpecializedAgent(ctx, message, recommendedAgent)
                output <- newResult
            } else {
                output <- result
            }
        case <-ctx.Done():
        }
    }()

    return output
}
```

```go
func SendFillerMessage(ctx context.Context, processing <-chan VoiceResult) <-chan string {
    filler := make(chan string, 1)

    go func() {
        defer close(filler)

        timer := time.NewTimer(750 * time.Millisecond)
        defer timer.Stop()

        select {
        case <-timer.C:
            select {
            case filler <- "One moment please...":
            case <-ctx.Done():
            }
        case <-processing:
            return
        case <-ctx.Done():
        }
    }()

    return filler
}

func ProcessVoiceWithFiller(ctx context.Context, message, currentAgent string) (VoiceResult, string) {
    processing := ProcessVoiceMessage(ctx, message, currentAgent)
    fillerCh := SendFillerMessage(ctx, processing)

    select {
    case result := <-processing:
        return result, ""
    case filler := <-fillerCh:
        result := <-processing
        return result, filler
    case <-ctx.Done():
        return VoiceResult{}, ""
    }
}
```

Recap

Answer quality
Can we actually resolve the customer's issue?

Reliability
Downtime means unhappy customers and more support tickets

Realtime safety and guardrails
Don't accidentally refund this person's plane

Latency
Respond quickly, people aren't willing to wait

Unified interface across channels
One brain for chat, voice, and email

Assembled

Questions?

Come find me afterwards or shoot me an email!

john@assembled.com