Pain Point to Solution Agent

Introduction

In the field of customer experience and service management, businesses often struggle to identify the right solution or capability to address specific operational or user pain points. Given a wide array of tools such as customer surveys, journey tracking, conversation analytics, and automated support systems, selecting the most relevant solution can be challenging.
This document outlines the design of a **Pain Point to Solution Agent** – an intelligent assistant that:

- Accepts a user-described business pain point in natural language
- Analyzes and compares it with a structured knowledge base of available features
- Returns a set of recommended solutions that are most relevant to the problem, along with explanations

The design focuses on three key aspects: **input understanding**, **feature knowledge representation**, and **intelligent matching logic**. The following sections elaborate on each of these components.

1. Define the Agent's Input

1.1 Required Information

To effectively understand and respond to a business user's pain point, the agent requires the following primary input:

- **Pain Point Description** *(required)*: A natural language sentence or paragraph where the user describes a specific problem or challenge related to customer experience or customer service .

The more specific and context-rich the pain point description, the more accurate the agent's recommendations will be.

1.2 Input Format and Structure

To standardize and facilitate parsing, we propose that the input be submitted in **JSON** format, with the following structure:

```
{
  "pain_point": "We're getting low survey response rates from customers after checkout.",
  "industry": "E-commerce",
  "company_size": "Medium",
  "language": "en"
}
```

Fields:

- **pain_point** *(string, required)*: Free-text input describing the business issue or concern.
- **industry** *(string, optional)*: The sector the business operates in (e.g, e-commerce, finance, hospitality). Helps with domain-specific recommendations.
- **company_size** *(string, optional)*: Approximate business size (e.g., Small, Medium, Enterprise). Can influence the scale or complexity of the suggested solution.
- **language** *(string, optional)*: Indicates the preferred language for responses. Enables multilingual support if needed (e.g., "en", "vi").

1.3 Rationale for This Design

JSON is structured and developer-friendly, making it ideal for both API-based and form-based frontends.
Including optional fields like **industry** or **company_size** allows the agent to deliver more personalized and relevant recommendations, especially in future iterations.
Maintaining **language** as a field prepares the agent for multilingual support, which is often critical in customer experience platforms.
Keeping the input flexible yet structured enables integration with:

- Web interfaces (forms)
- Chatbots
- Backend ticketing or CRM systems

1.4 Example Inputs

Example 1:

```Json
{
  "pain_point": "Our support team is overwhelmed by repeated customer questions about delivery status.",
  "industry": "Logistics",
  "company_size": "Enterprise",
  "language": "en"
}
```

Example 2:
```json
{
  "pain_point": "Khách hàng thường xuyên bỏ cuộc giữa chừng khi điền khảo sát nhưng không rõ lý do.",
  "industry": "Retail",
  "company_size": "Small",
  "language": "vi"
}

```

2. Define the Agent's Output

2.1 Objective

The agent’s output is a structured list of recommended solutions (i.e., features or capabilities) that best match the user’s described pain point. Each recommendation should be:

- **Actionable** – the user can understand what to do next
- **Relevant** – aligned with the specific issue described
- **Informative** – clearly explains how the feature helps solve the problem

2.2 Propose a Clear Structure and Format

Each solution is represented as a **JSON object**, and all solutions are returned as a **list (array)**. This makes the output:

- Easy to parse by frontend clients or chatbot interfaces
- Human-readable and machine-readable
- Flexible for adding new fields later

```json

[

  {

    "feature_name": "Surveys",

    "category": "Voice of Customer (VoC)",

    "description": "Design and deploy multi-channel customer surveys.",
```

"how_it_helps": "Trigger post-checkout surveys via email or SMS to collect consistent customer feedback.",

"relevance_score": 0.91,

"docs_link": "https://docs.example.com/features/surveys"

}

]

```

To make each suggestion actionable, understandable, and useful, here’s a breakdown of what each field means and why it’s important:

| Field | Required | Type | Purpose and Why it matter? |
|---|---|---|---|
| feature_name | yes | string | Identifies the exact feature the user can look up or activate. Clear labeling improves user trust and next-step decision-making. |
| category | yes | string | Groups features under logical product pillars like “VoC” or “Customer 360” to help users understand the ecosystem. |
| description | yes | string | Gives a short, general summary of the feature. Useful when users are unfamiliar with the full product suite. |
| how_it_helps | yes | string | Explains in natural language how this feature directly addresses the pain point. This is the most important bridge between user needs and technical solutions. |
| relevance+score | recommended | float (0-1) | Indicates how strongly the feature matches the pain point. Enables result ranking and helps users prioritize actions. |
| docs_link | optional | string (URL) | Gives users a way to learn more or take action immediately. Adds credibility and follow-through. |
```

2.3 Example – Realistic Use Case

Input:

```json
{
  "pain_point": "Our agents spend too much time replying to the same customer questions repeatedly.",
  "language": "en"
}

```

Output:

```json
[
  {
    "feature_name": "AI Inbox",
    "category": "AI Customer Service",
    "description": "A collaborative inbox where human and AI agents handle customer conversations together.",
    "how_it_helps": "The AI agent can automatically handle repetitive questions and FAQs, reducing load on human agents.",
    "relevance_score": 0.94,
    "docs_link": "https://docs.example.com/ai-inbox"
  },
  {
    "feature_name": "Knowledge Base Integration",
    "category": "AI & Automation",
    "description": "Integrate FAQs and standard responses for instant AI replies.",
    "how_it_helps": "Feeds the AI agent with content to instantly answer common customer queries.",
    "relevance_score": 0.85
  }
]
```

3. Design the Feature Knowledge Base Structure

3.1 Objective

To enable the agent to recommend relevant Filum.ai features based on a user's pain point, we need a structured and machine-consumable representation of the product's capabilities. This is known as the Feature Knowledge Base (KB).

The KB must allow the agent to:

- Understand the semantics of a user's input (written in natural language)
- Match pain points to feature descriptions, use cases, and problem-solving patterns
- Rank the most relevant features with confidence and explainability

To accomplish this, the KB must be designed in a way that supports both **efficient information retrieval** (e.g., keyword match) and **semantic reasoning** (e.g., via embeddings or LLMs).

3.2 Data Structure Options for Feature Representation

We evaluated several approaches for representing and storing the feature knowledge base:

Option	Description	Pros	Cons	Suitable for
Flat JSON file	A structured list of feature objects stored in a single .json file	- Easy to edit - Portable - Compatible with keyword + embedding search	- No scalability for large-scale - No real-time querying	Prototypes, lightweight matching
Relational Database (e.g., SQLite, PostgreSQL)	Structured tables (features, categories, tags, etc.)	- Powerful queries (SQL) - Scalable - Join/filter capabilities	- More setup - Not semantic-aware	Mid-to-large apps, production backends
Vector Store (e.g., FAISS, Pinecone)	Stores embeddings for semantic search	- Excellent for meaning-based retrieval - Fast similarity queries	- Requires pre-embedding - No structured filtering without hybrid logic	LLM/RAG pipelines, semantic agent

Hybrid RAG Corpus (e.g., text + metadata)	Combines unstructured documents (e.g., Markdown/paragraphs) with LLM-based retrieval	- Human-like understanding - Great for GPT-4 reasoning	- Lacks structure - Slower, needs prompt engineering	Fully AI-driven agents

3.3 Comparison of Matching Techniques for Pain Point

To recommend the most relevant feature(s) based on a user's pain point, the agent must **match the natural language input to entries in the Feature Knowledge Base**.

Below is a comparison of key matching techniques:

Technique	Description	Pros	Cons	Suitability
TF-IDF	Converts text into vector based on term frequency–inverse document frequency	- Simple, interpretable - Fast for small corpus	- Ignores context - Struggles with synonyms and rephrasing	Good for keyword-heavy input and prototypes
Fuzzy Matching	Leverages string similarity (e.g., Levenshtein distance) to detect approximate matches	- Tolerates typos - Works well with short texts	- Not semantic-aware - Easily confused by similar terms	Quick filtering of feature names or tags
Embedding-based Semantic Search	Converts both pain point and feature data to embeddings and compares using cosine similarity	- Understands meaning - Handles paraphrasing - Ideal for vague inputs	- Requires precomputed embeddings - Costlier to compute	Best for LLM-based agents or user-facing tools
Hybrid Search (Keyword + Semantic)	Combine keyword filtering + semantic reranking	- High precision - Good fallback mechanism	- Adds pipeline complexity	Recommended for robust systems

LLM-based Reasoning (e.g., GPT-4 with RAG)	Use GPT-4 to directly select/reason over features from top-N candidates	- Powerful reasoning - Natural language explanation	- Slower - Requires careful prompt design	Advanced use cases, explainable agents

3.4 Feature

Based on the provided product documentation, we can organize Filum.ai's capabilities into a structured set of features. These features form the core knowledge base that the agent will reference to recommend solutions in response to user-submitted pain points. Each feature is described with its name, category, functionality, typical use cases, and examples of how it helps resolve specific business problems.

1. **Surveys**

```json
{
  "feature_name": "Surveys",
  "category": "Voice of Customer (VoC)",
  "description": "Design and deploy feedback surveys across multiple channels such as Web, Mobile App, Zalo, SMS, Email, QR code, and POS systems.",
  "keywords": ["survey", "feedback", "CSAT", "NPS", "questionnaire", "form"],
  "use_cases": [
    "Collect post-purchase feedback automatically",
    "Gather customer satisfaction ratings after support calls",
    "Launch targeted satisfaction surveys on mobile or POS"
  ],
  "how_it_helps_examples": [
    "Increases response rates by automating survey delivery across preferred customer channels",
    "Improves customer insight collection at key journey moments"
  ],
  "docs_link": "https://docs.filum.ai/surveys"
```

```
}
```

2. Journeys

```json
{
  "feature_name": "Journeys",
  "category": "Voice of Customer (VoC)",
  "description": "Visualize and manage customer journeys across touchpoints, identifying key friction or drop-off points.",
  "keywords": ["customer journey", "touchpoint", "funnel", "conversion", "experience mapping"],
  "use_cases": [
    "Identify which steps in the onboarding flow cause customer frustration",
    "Map customer paths before and after submitting a support ticket"
  ],
  "how_it_helps_examples": [
    "Highlights friction points in customer journeys through aggregated behavior and feedback",
    "Improves user flow by showing drop-off or delay steps"
  ],
  "docs_link": "https://docs.filum.ai/journeys"
}
```

3. Conversations

```json
{
  "feature_name": "Conversations",
```

"category": "Voice of Customer (VoC)",

"description": "Analyze customer interactions across chat, call transcripts, and email using AI-based topic and sentiment extraction.",

"keywords": ["conversation analysis", "chat", "calls", "emails", "voice of customer", "sentiment"],

"use_cases": [

"Identify recurring complaints in support chats",

"Analyze email tone changes before churn",

"Detect negative feedback in call transcripts"

],

"how_it_helps_examples": [

"Automatically detects trends and themes from unstructured customer messages",

"Reduces manual review time of large volumes of conversations"

],

"docs_link": "https://docs.filum.ai/conversations"

}

```

4. **AI Inbox**

```json
{
  "feature_name": "AI Inbox",

  "category": "AI Customer Service",

  "description": "A collaborative inbox where human and AI agents co-manage customer conversations, automating first-level replies.",

  "keywords": ["AI agent", "inbox", "automated response", "FAQ", "support ticket", "chatbot"],

  "use_cases": [

```
    "Deflect repetitive support questions with AI",
    "Suggest reply drafts to human agents in real-time",
    "Auto-tag and prioritize incoming messages"
  ],
  "how_it_helps_examples": [
    "Reduces agent workload by 30–50% through AI-driven answers",
    "Improves response speed for common issues"
  ],
  "docs_link": "https://docs.filum.ai/ai-inbox"
}
```

5. **Tickets**


```json
{
  "feature_name": "Tickets",
  "category": "AI Customer Service",
  "description": "Centralized ticket management system with customizable workflows, SLAs, and AI-based prioritization.",
  "keywords": ["support", "ticket system", "case management", "workflow", "SLA"],
  "use_cases": [
    "Track customer-reported issues from multiple channels",
    "Assign and escalate tickets based on urgency or topic",
    "Analyze ticket resolution time by agent or team"
  ],
  "how_it_helps_examples": [
    "Ensures no customer issues are missed or delayed",
    "Optimizes operations by prioritizing high-impact tickets"
```

```
  ],
  "docs_link": "https://docs.filum.ai/tickets"
}
```

6. **Experience Insights**

```json
{
  "feature_name": "Experience Insights",
  "category": "Insights",
  "description": "Analyze customer feedback and behavior across touchpoints to identify experience trends, topics, and friction zones.",
  "keywords": ["experience analysis", "touchpoint", "feedback trend", "topic extraction", "NLP"],
  "use_cases": [
    "Identify most common negative themes in survey responses",
    "Detect rising complaints linked to a new product update",
    "Correlate customer sentiment with support wait times"
  ],
  "how_it_helps_examples": [
    "Provides a big-picture view of customer experience drivers",
    "Drills down into actionable feedback across journeys"
  ],
  "docs_link": "https://docs.filum.ai/insights-experience"
}
```

7. **Customer Profiles**

```json
```

```
{
  "feature_name": "Customers (360 View)",
  "category": "Customer 360",
  "description": "Build complete profiles of individual customers, including demographics, interaction history, and segmentation.",
  "keywords": ["CRM", "360 view", "interaction history", "customer data", "segmentation"],
  "use_cases": [
    "View all messages, surveys, and purchases tied to a single customer",
    "Segment users by loyalty, behavior, or recent sentiment",
    "Export customer lists for personalized engagement"
  ],
  "how_it_helps_examples": [
    "Gives agents a full picture of every customer before responding",
    "Supports targeted campaigns with detailed segmentation"
  ],
  "docs_link": "https://docs.filum.ai/customer-360"
}
```

Each feature in the knowledge base is represented as a structured JSON object with carefully selected properties. These fields are derived directly from the official product documentation and designed to support both keyword-based and semantic reasoning. Below is the explanation of each property and its role:

- **feature_name**:
  The unique name of the feature. Acts as the identifier and is useful for human-readable output and UI display.

- **category**:
  Maps the feature to its product family (e.g., "VoC", "AI Customer Service", "Insights"). Helps narrow down recommendations when the user pain point hints at a specific product area.

- **description**:
  A concise functional summary of the feature. It is used in semantic matching and to give the

user an immediate understanding of what the feature does.

- **keywords**:
  A list of relevant terms and synonyms extracted from the documentation or common usage. Supports fast keyword filtering (e.g., TF-IDF or inverted index) and improves recall.

- **use_cases**:
  Describes when and how the feature is typically used. These sentences often align semantically with user pain points and are heavily weighted during embedding-based matching.

- **how_it_helps_examples**:
  Shows direct benefits in business terms — critical for both reasoning and **explainability**. These fields are often reused verbatim when the agent generates a natural language response.

- **ideal_industries** (optional):
  Identifies which industries benefit the most from the feature. Useful if future versions of the agent personalize results by domain.

- **docs_link**:
  Points to more information or documentation. Helps users quickly follow up on the recommendation and supports transparency.

4. Conclusion

This design outlines a complete plan for building a "Pain Point to Solution Agent" that maps user-described issues to the most relevant Filum.ai features.

We defined a clear input/output format, a structured JSON-based knowledge base, and proposed a hybrid matching approach that combines keyword filtering and semantic search. Each feature is enriched with use cases and business-focused descriptions to improve accuracy and explainability.

This foundation supports a scalable, intelligent agent that can help users discover the right solutions quickly — driving better customer experience and internal efficiency.