Elastic - Agentic RAG 构建之路

李捷：首席解决方案架构师

July 27, 2025

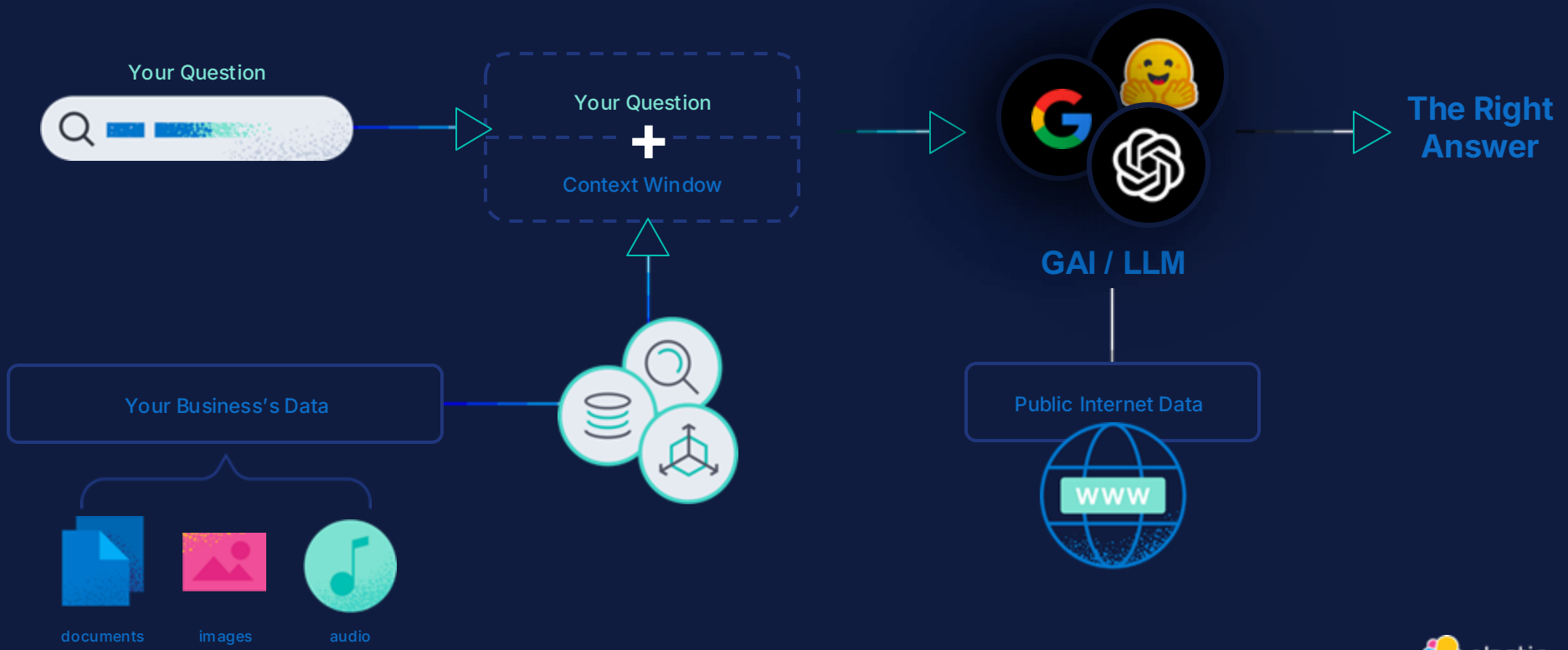
Agenda

我们将在今天分享:

- RAG 的限制： Agentic RAG Over RAG
- 构建 Agentic RAG 需要的 4 个关键能力
- Elastic - Agentic RAG 构建之路

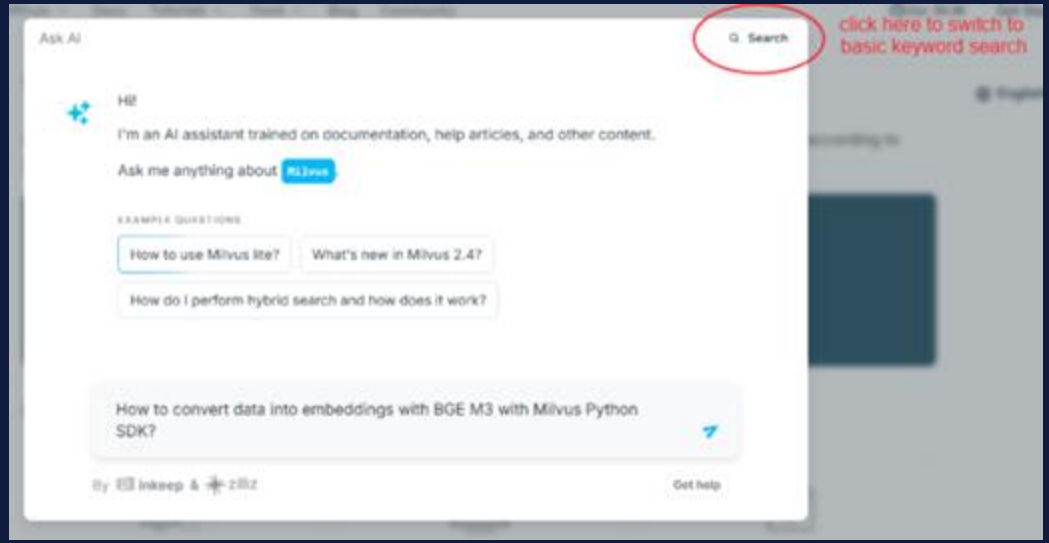
elastic

企业都在寻找与 AI 的结合点



elastic

RAG 能处理什么问题？




```
for method in chunk_methods:
    print(f"分块方法: {method}")
    # 获取参考信息
    ref_info = get_ref_info(query, search_params, limit, output_fields, method)
    # 生成用户提示词
    user_prompt = (
        f"请你根据提供的参考信息，查找是否有与问题语义相似的内容。参考信息：{ref_info}。问题：{query}。\
        f"如果找到了相似的内容，请回复"鲁迅的确说过类似的话。原文是[原文内容]，这句话来自[文章标题]"。\
        f"[原文内容]是参考信息中ref字段的值，[文章标题]是参考信息中title字段的值。如果引用它们，请引用
        f"如果参考信息没有提供和问题相关的内容，请回答"抱我所知，鲁迅并没有说过类似的话。""
    )
    # 生成响应
    generate_response(system_prompt, user_prompt, model, temperature)
    print("\n" + "*" * 50 + "\n")
```

好啦，一切准备就绪，让我们看看使用不同分块方法的 RAG，究竟有什么区别。先看第一句话，"世上本没有路，走的人多了，也便成了路。"

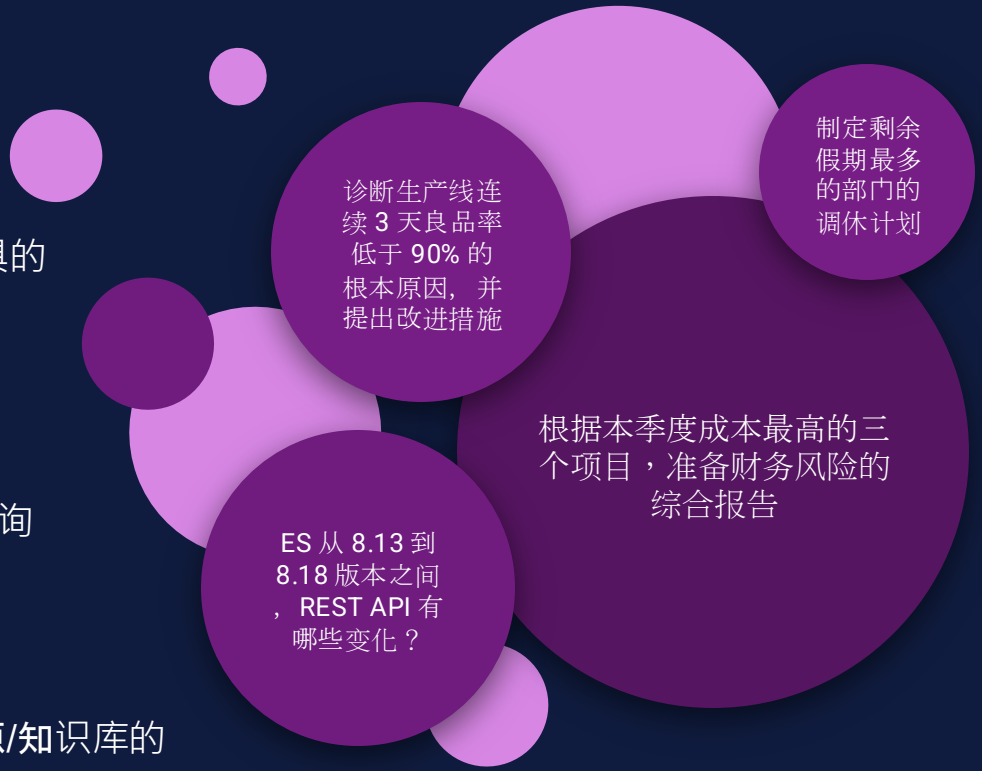
```
分块方法：fixed_chunk
鲁迅的确说过类似的话。原文是"的人多了，也便成了路。 一九二一年一月。"，这句话来自《故乡》。
**************************************************

分块方法：semantic_chunk
鲁迅的确说过类似的话。原文是"跨过了灭亡的人们向前进，什么是路？就是从没路的地方践踏出来的，从只有荆棘的地方
**************************************************

分块方法：sentence_window
鲁迅的确说过类似的话。原文是"我想：希望本是无所谓有，无所谓无的。 这正如地上的路：其实地上本没有路，走的人
```

主动环境感知

先聚合数据，再分析？



实时数据或外部工具的查询

多步推理与动态任务规划？

迭代优化和反馈的查询

跨多个数据源/知识库的协作检索

复杂格式或结构化数据的处理？

elastic

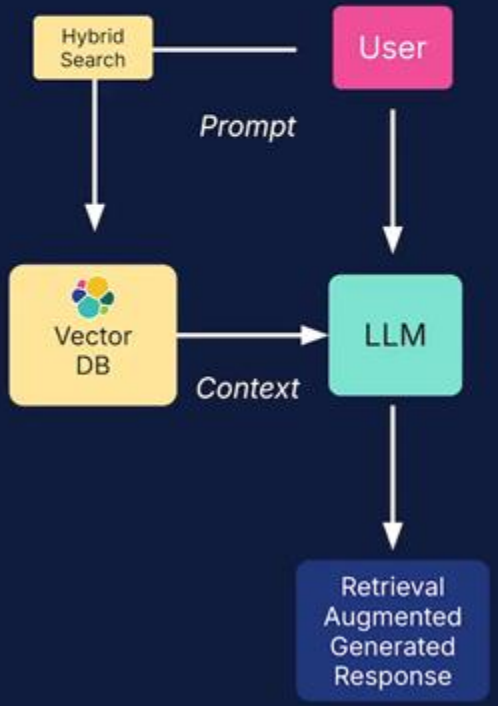
From Text Generation to Decision Making

Prompting an LLM

Prompt the LLM and get a response. No other tools or components needed.

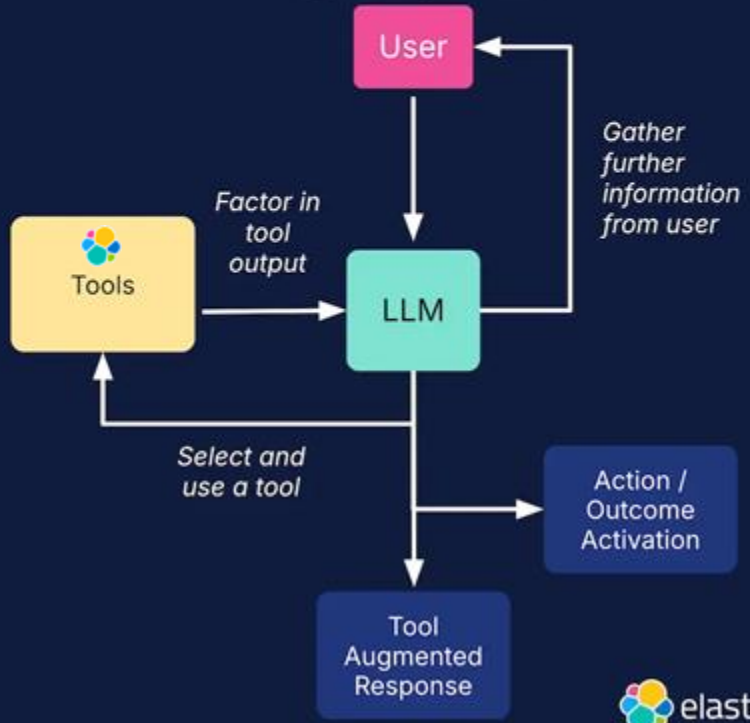
Retrieval Augmented Generation

Add a knowledge base to enhance accuracy and add novel information to the LLM



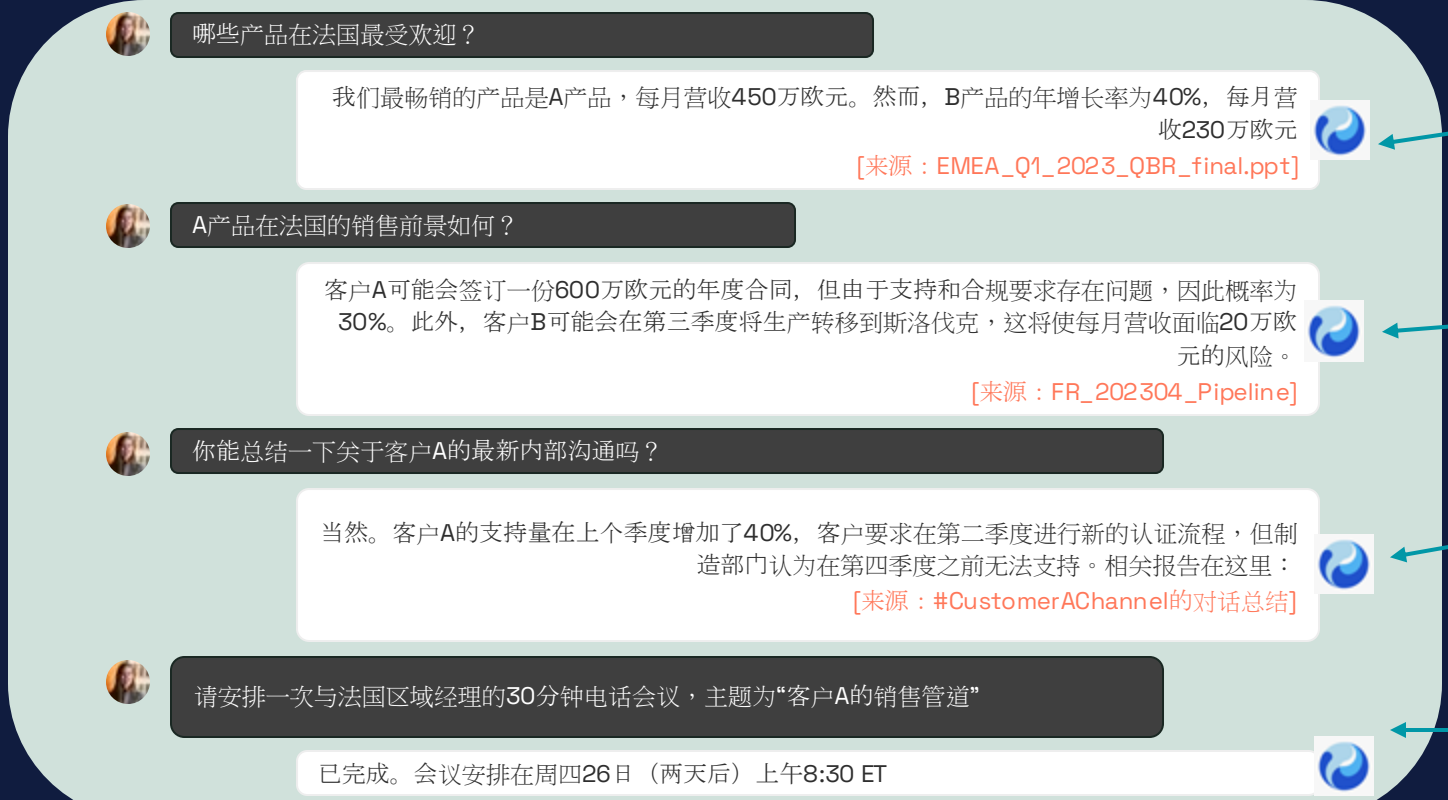
Agentic Flow

Decision Making enabled. LLM can prompt user for information, choose to use tools, interact with other agents, and affect the real world (ie. Triggering alerts, sending messages, etc...)



elastic

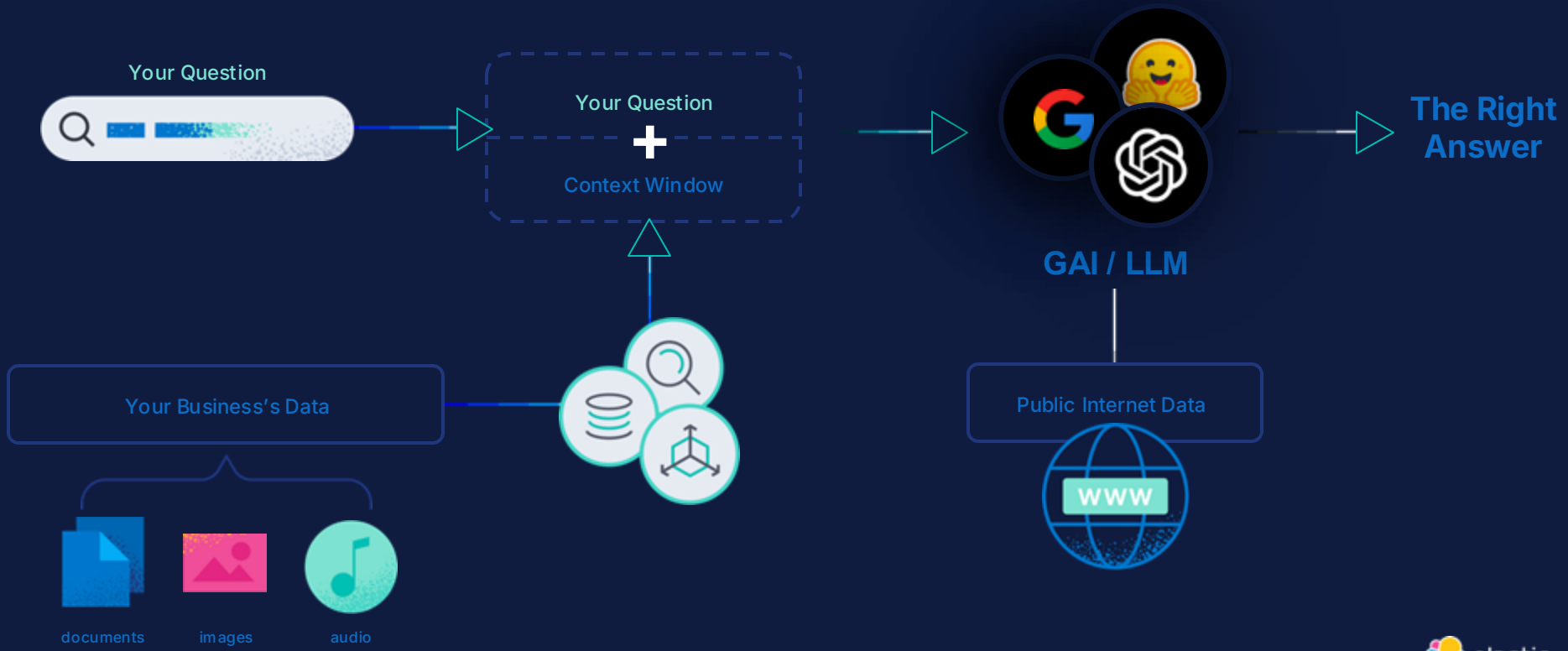
从简单 RAG 到 Agentic RAG



02

Agentic RAG 需要什么样的引擎？

GAI 应用 != 简单的概念验证



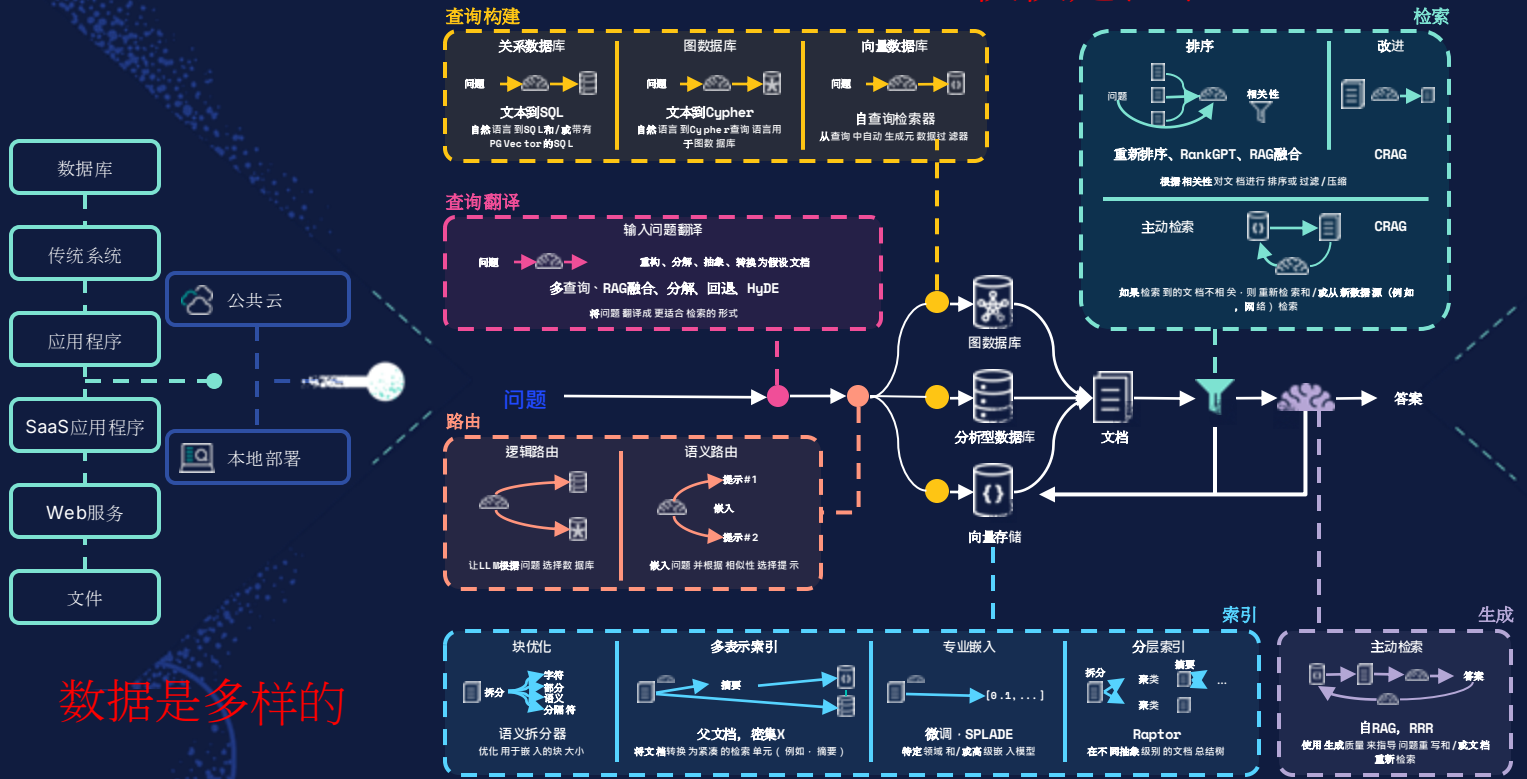
elastic

从"流水线"到"智能大脑"

Agentic RAG的架构，更像一个具备规划、记忆、执行和反思能力的智能大脑。

GAI 应用远非想的那么简单

LLM 生成的不仅仅是答案



数据是多样的

链路调试复杂，
并且不安全

业务成果

客户和员工满意度提高**69%**

总拥有成本降低**50%**

收入中断减少**62%**

风险降低**60%**

来源：LangChain

不仅仅是查询

elastic

Agentic RAG 引擎需要的进阶能力

01

数据融合

不仅能够查询知识库

还能融合业务数据，运营数据，运维数据， 安全数据

根据用户的问题，跟知识与数据对话

避免数据孤岛

02

查询与分析能力

不仅仅是召回知识

还能进行统计分析，数据处理， 数据挖掘

强大的数据分析能力与富于表达的查询语言

避免复杂的异构系统

03

LLM 友好的设计

能够被大模型理解和使用

生成 Query， 分析， ETL 等任务

多种任务支撑，管道语言

避免可生成性差的系统集成

04

可靠， 安全

整个链路的可见，可调试，可监控，可保障

具备可观测和必要的 LLM 安全特性

避免因盲点带来的各种问题

elastic

全域数据融合能力 (Holistic Data Fusion)

核心理念： 打破数据孤岛，实现与企业所有数据的统一对话。

AI应用不应只局限于非结构化的“知识”，更要成为洞察企业全景的入口。这意味着平台必须具备强大的数据融合能力，无缝连接并理解不同类型、不同来源的数据。

- **融合范围：**
  - **知识数据 (Knowledge Data):** 如文档、PDF、音视频、网页等，通过向量、文本、图谱等形式管理。
  - **业务数据 (Business Data):** 来自CRM、ERP、BI系统中的结构化数据，反映企业的核心商业活动。
  - **运营数据 (Operational Data):** 来自监控系统、日志平台的时序数据和指标，反映IT系统的健康状况 (AIOps)
  - **安全数据 (Security Data):** 来自SIEM等安全平台的数据和事件，反映企业安全态势。
- **设计要求：**
  - **统一查询入口:** 用户可以用自然语言提出一个横跨多个数据域的问题，如“上季度销售额最高的产品的用户反馈和相关的IT系统稳定性报告”，平台能自动理解并查询相应的BI、知识库和运维系统。
  - **避免数据孤岛:** 在架构设计之初就考虑数据联邦或统一数据模型的可能性，而不是事后进行复杂的集成。

elastic

深度查询与分析能力 (In-depth Query & Analysis)

核心理念： 超越简单的信息“召回”， 提供深度的“洞察”

平台不仅要能“找到”信息，更要能对信息进行计算、处理和挖掘，从而揭示隐藏的模式和价值。

- **能力要求：**
  - **统计分析与数据处理: 能**够执行聚合、排序、过滤、关联分析等操作。例如，用户问“统计一下所有关于A产品的功能缺陷类工单，并按模块分类”， **平台**应能直接完成统计分析，而非仅仅返回所有相关工单
  - **数据挖掘: 具**备执行更复杂算法的能力，如趋势预测、异常检测、聚类分析等
  - **富有表达力的查询语言: 平台需要提供一种**强大且统一的查询语言或框架。这种语言既能被人类分析师使用，**也能被**LLM轻松生成。它屏蔽了底层异构系统的复杂性，让复杂的分析任务可以通过简洁的表达式来定义
- **设计要求：**
  - **内置分析引擎: 平台内部集成或**紧密耦合数据处理和分析引擎，避免在多个异构系统之间来回切换数据。
  - **一体化设计: 将数据存**储、查询、分析功能整合在一个平台上，最大化性能和易用性。

elastic

LLM原生友好设计 (LLM-Native & Friendly Design)

核心理念： 让LLM成为平台的使用者和编排者，而非被复杂性困扰的“调用方”

平台的每一部分都应该思考“LLM如何才能更好地理解和使用我？”。目标是降低LLM执行复杂任务的“认知负荷”。

- **能力要求：**
  - **可被LLM理解的工具集:** 平台提供的所有能力（查询、分析、ETL，挖掘）都应封装成清晰、原子化的“工具”，并有详尽的、机器可读的描述（Tool/API Specification）。
  - **强大的管道/工作流语言 (Pipeline Language):** 提供一种高级的、声明式的语言，允许LLM通过生成一段简洁的工作流脚本来编排一系列复杂的任务。例如，LLM只需生成 `load_data | filter_by_date | run_sentiment_analysis | visualize_as_bar_chart`，而无需关心每一步的具体实现细节。这极大地减轻了LLM在多步推理中对上下文长度和逻辑复杂度的依赖。
  - **LLM友好的API:** 所有暴露给LLM的接口都应遵循简单、一致的原则，避免不直观的参数设计或需要复杂状态管理的交互。
- **设计要求：**
  - **以LLM为中心的设计范式:** 在API和SDK设计时，始终把“这是否便于LLM生成和调用”作为核心考量。
  - **解耦LLM与执行:** LLM负责“规划”（生成工作流），平台负责“执行”。这种解耦使得系统更加稳定和可预测。

企业级可靠性与安全性 (Enterprise-Grade Reliability & Security)

核心理念： 在复杂链路中建立信任，确保系统的透明、可控和安全

Agentic RAG的链路长而复杂，任何一个环节的“盲点”都可能导致系统性风险。因此，企业级的可观测性和安全性是不可或缺的基石。

- **能力要求：**
  - **端到端可观测性 (End-to-End Observability):**
    - **日志(Logging):** 记录每一次LLM的思考链（Chain-of-Thought）、工具调用、数据召回和最终响应。
    - **追踪(Tracing):** 能够完整追踪一个用户请求从输入到输出的全过程，清晰地看到每一步的耗时、输入和输出。
    - **监控(Monitoring):** 对关键指标（如召回准确率、响应时间、Token消耗、工具调用成功率）进行实时监控和告警。
  - **全面的LLM安全保障 (LLM Security - LLM Guard):**
    - **输入防护:** 检测并防范提示词注入（Prompt Injection）、越狱攻击等。
    - **输出防护:** 扫描和过滤生成内容，防止数据泄露（PII）、有害信息、偏见和歧视性言论的产生。
    - **内容审核与溯源:** 所有交互都应可审计，所有生成的结论都能追溯到其所依赖的数据源。
- 设计要求：
  - **内置的可观测性框架:** 将可观测性作为平台的一等公民，而非事后附加的功能。
  - **可插拔的安全组件:** 提供灵活的安全策略配置，允许企业根据自身合规要求定制输入输出的防护规则。

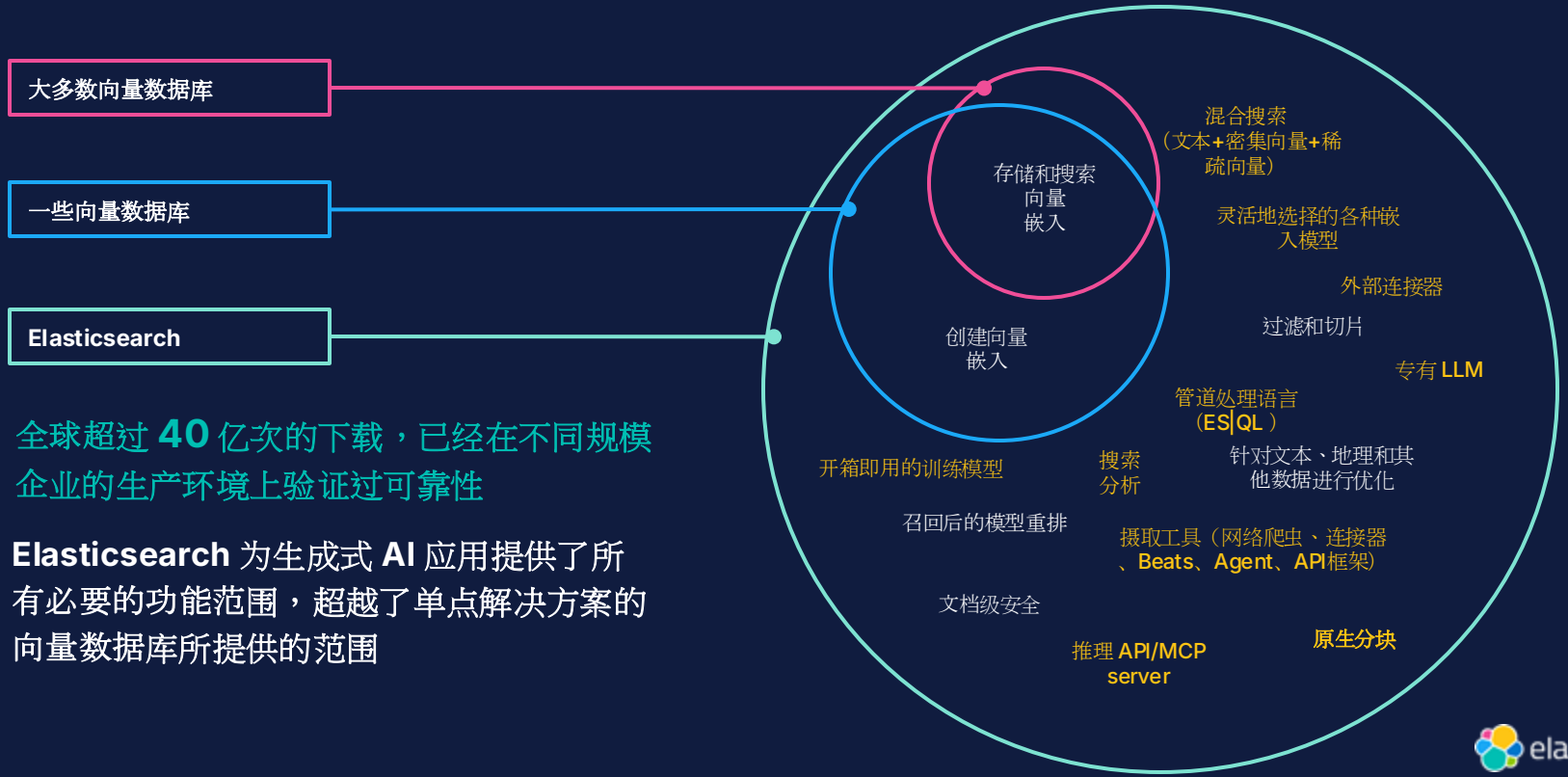
elastic

03

Elastic Agentic RAG 构建之路

elastic

Elastic 拥有您所需的所有功能

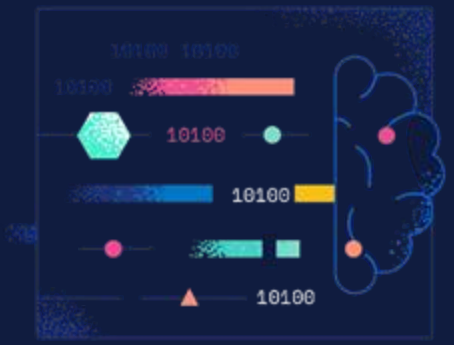


全球超过 40 亿次的下载，已经在不同规模企业的生产环境上验证过可靠性

Elasticsearch 为生成式 AI 应用提供了所有必要的功能范围，超越了单点解决方案的向量数据库所提供的范围

elastic

生成式 AI 不是 魔法



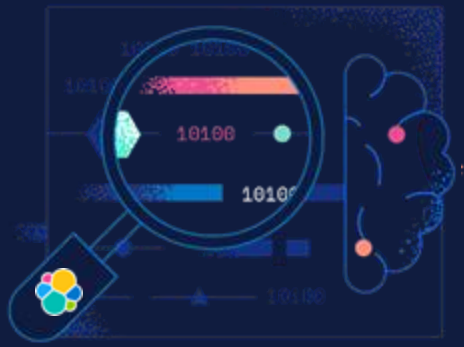
将最相关数据带给生成式AI的方法

生成式AI需要的不仅仅是一个向量数据库

没有一站式服务的生成式AI

elastic

Search AI 是正确的出路



Elastic 将搜索的精确性与 AI 的智能相结合。

只有 Elastic 提供你所需的所有功能。

Elastic 是生成 AI 生态系统中最开放的成员。

elastic

如何实现？依靠 Search AI Platform

构建你自己的

Elasticsearch

开箱即用的解决方案

可观测性

安全

Search AI 平台

数据摄取

处理

存储与复制

搜索

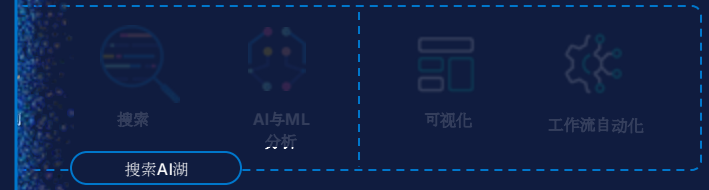
AI和ML分析

可视化

工作流自动化

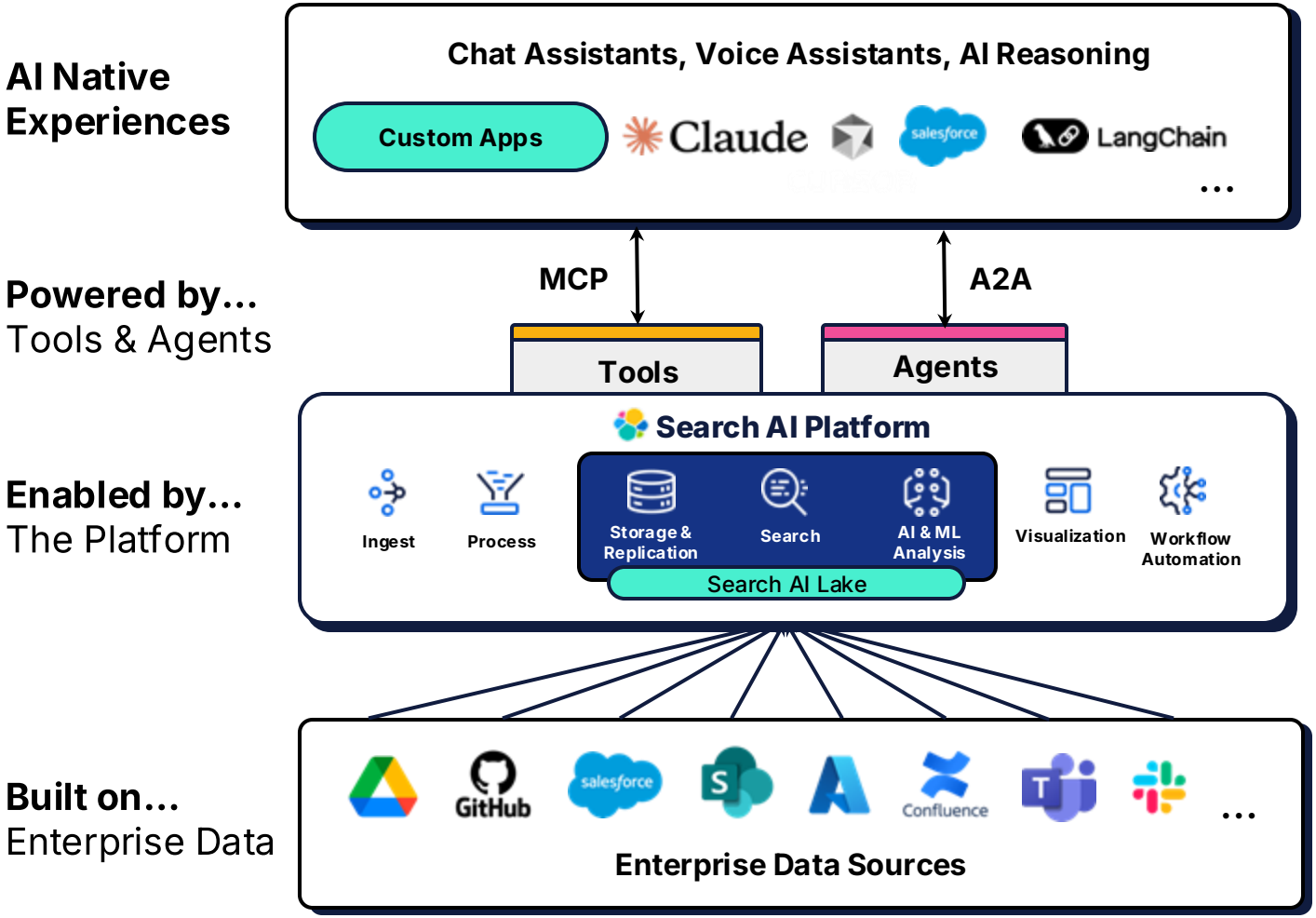
搜索AI湖

新功能将汇集您的所有数据



elastic

Elastic
通过 **500** 个集成
40 +连接器
将知识开放给智能主体生态系统

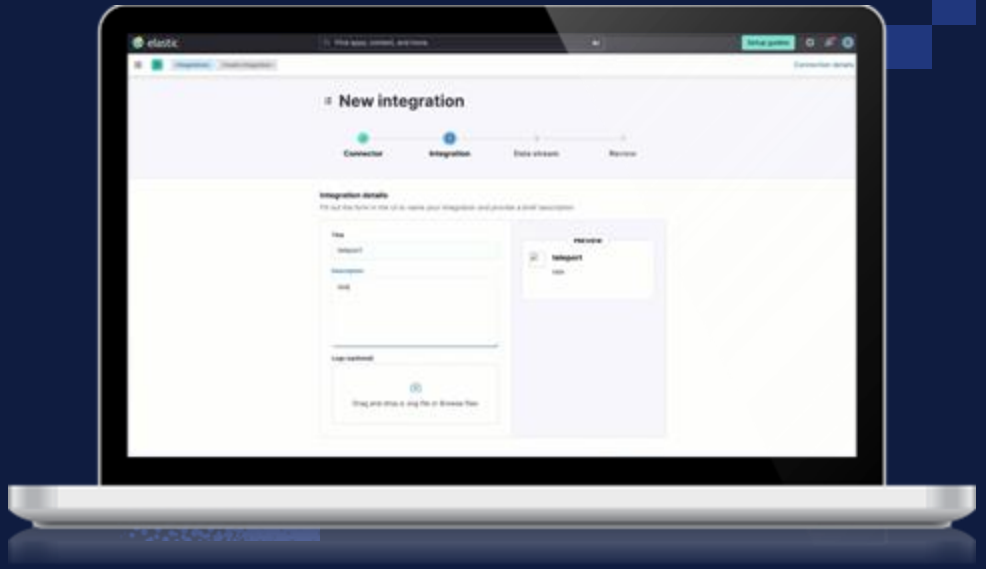


elastic

现已推出

自动导入

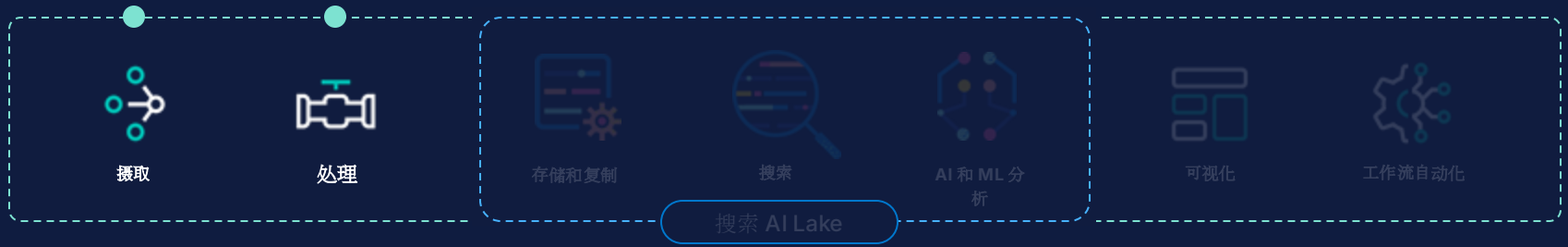
- 降低数据导入的成本、复杂性和压力
- 几分钟内创建自定义数据集成
- 超越Elastic 提供的 400 多种预构建数据集成，实现更广泛的可见性提升



立即可用

Elastic OpenTelemetry 发行版

- 唯一在所有可观测性和安全信号类型中具有纯 OTel 实现的供应商
- 在上游发布之前交付增强功能
- 来自 OTel 项目最大贡献者的商业支持服务

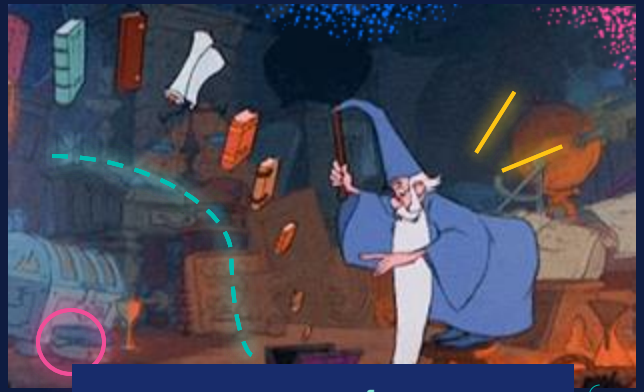


可用

数据超融合

LogsDB 索引模式

- 优化日志的索引排序和压缩
- 减少日志数据存储高达70%
- 存储更多数据并保持预算



```
"settins": {
"index.mode": "logsdb"
}
```

Standard
Time_series
Logsdb
metricdb

摄取 | 处理 | 存储和复制 | 搜索 | AI 和 ML 分析 | 可视化 | 工作流自动化

搜索 AI 湖

帮助你搜索所有数据的新功能

elastic

查询，过去三个季度 elastic 的财务报告。

Gemini
2.5 Pro

第 2 步：生成查询向量

您需要使用与索引数据时相同的**文本嵌入模型**（Embedding Model）来将您的查询文本转换为向量。

Python

```python
from sentence_transformers import SentenceTransformer

# 加载与您索引时相同的模型
# 这里的 'all-MiniLM-L6-v2' 是一个例子，请替换为您自己的模型
model = SentenceTransformer('all-MiniLM-L6-v2')

query_text = "Elastic's quarterly and annual financial reports and earnings statem
query_vector = model.encode(query_text).tolist()
```

提示：查询文本可以写得更丰富一些，包含同义词，以获得更好的语义搜索效果。

第 3 步：构建并执行 Milvus 查询

现在，我们将结合查询向量和过滤条件，对您可能存在的多个 Collection 进行搜索。

假设您有两个相关的 Collection：`reports_2024` 和 `reports_2025`。

Python

```python
from pymilvus import utility, Collection, connections

# --- 连接 Milvus ---
# 假设您已经配置了连接别名
connections.connect("default", host="localhost", port="19530")

# --- 准备查询参数 ---
collections_to_search = ["reports_2024", "reports_2025"]
top_k = 10 # 您想找出的最相关的 top K 个结果

# 构建强大的过滤表达式 (Boolean Expression)
filter_expression = f"(publication_date >= {start_timestamp} and publication_date

search_params = {
    "metric_type": "L2", # 或 "IP"，取决于您的索引类型
    "params": {"nprobe": 16}, # 示例参数，根据您的索引类型调整
}

all_results = {}

# --- 遍历所有需要查询的 Collection ---
for collection_name in collections_to_search:
    if not utility.has_collection(collection_name):
        print(f"Collection '{collection_name}' 不存在，跳过。")
        continue

    print(f"\n正在查询 Collection: '{collection_name}'...")
```

Gemini
2.5 Pro

Elasticsearch DSL 查询语句：

JSON

```
GET /my-enterprise-data-elser/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "text_expansion": {
            // 查询我们创建的 rank_features 字段
            "report_content_expansion": {
              // 提供原始的查询文本
              "model_text": "Elastic's financial report and earnings statement",
              // 指定在查询时使用的模型
              "model_id": ".elser_model_2"
            }
          }
        }
      ],
      "filter": [
        {
          "range": {
            "publication_date": {
              "gte": "2024-10-01",
              "lte": "2025-06-30"
            }
          }
        },
        {
          "term": {
            "company_name": "Elastic"
          }
        },
        {
          "terms": {
            "doc_type": ["quarterly_report", "10-Q"]
          }
        }
      ]
    }
  },
  "size": 10,
  "_source": ["report_title", "publication_date", "doc_type"]
}
```

现在可用

语义文本字段类型

我选择什么类型的模型？

什么是零样本模型？

如何将内容转换为向量？

什么是向量数据库？

什么是分块？

semantic_text

如何用向量写查询？

密集与稀疏向量？

如果我的块太大怎么办？

有多少维度？

- 使将语义搜索构建到应用程序和服务中变得简单
- 自动生成和存储文本内容的向量嵌入
- 自动将长段落分成较小的部分

从 `text` 到 `semantic_text`

之前 ▶

```
PUT twitter
{
  "mappings": {
    "properties": {
      "user": {
        "type": "text"
      },
      "message": {
        "type": "text"
      }
    }
  }
}
```

索引创建

```
POST twitter/_doc/1
{
  "user": "kimchy",
  "postDate": "2009-11-15T13:12:00",
  "message": "尝试 Elasticsearch, so far is good？"
}
```

索引文档

```
GET twitter/_search
{
  "query": {
    "match": {
      "message": "good"
    }
  }
}
```

关键词搜索

elastic

从 `text` 到 `semantic_text`

之后 ▶

```
PUT twitter
{
  "mappings": {
    "properties": {
      "user": {
        "type": "text"
      },
      "message": {
        "type": "semantic_text"
      }
    }
  }
}
```

索引创建

```
POST twitter/_doc/1
{
  "user": "kimchy",
  "postDate": "2024-09-25T13:12:00",
  "message": "尝试 Elasticsearch, so far is good？"
}
```

索引文档

```
GET twitter/_search
{
  "query": {
    "semantic": {
      "message": "excellent"
    }
  }
}
```

语义搜索

elastic

从 LLM 友好度的角度看 Milvus 与 Elasticsearch 的核心区别

查询的“语言”本质 (The Nature of the "Query Language")

Elasticsearch:

- **声明式 & 自包含 (Declarative & Self-contained)**：Elasticsearch 的核心是其 **JSON 格式的查询语言 (Query DSL)** 和 **SQL 风格的查询语言 (ESQL)**。一个完整的查询请求就是一个**独立的、结构化的文本**（JSON对象或SQL字符串）。
- **LLM 的任务**: LLM 的任务是将用户的自然语言需求，直接翻译成一个或多个完整的、可以立即执行的JSON文档或ESQL语句。
- **示例**:
  - **用户**: "查找过去三个季度Elastic的财报"

LLM 生成目标 (JSON DSL):

```
JSON
{
  "query": {
    "bool": {
      "must": [{"match": {"content": "Elastic financial report"}}],
      "filter": [{"range": {"date": {"gte": "2024-10-01"}}}]
    }
  }

}
```

Milvus:

- **过程式 & 参数化 (Procedural & Parameterized)**：Milvus 的核心交互模式是**函数调用 (Function Call)**。一个查询不是一个独立的文本，而是对一个函数（如 `collection.search()`）的调用，需要分别准备和传入多个不同类型的参数（向量、过滤表达式字符串、搜索参数字典等）。
- **LLM 的任务**: LLM 的任务是理解用户的意图，然后生成一段**调用 SDK函数的代码**，或者为预定义的“**工具/函数**”**填充正确的参数**。
- **示例**:
  - **用户**: "查找过去三个季度Elastic的财报"

LLM 生成目标 (Python SDK):

```
Python
# LLM 需要知道有这个函数，并且知道每个参数的含义
collection.search(
    data=[query_vector], # 向量需要另行生成
    anns_field="report_vector",
    param={"nprobe": 16},
    limit=10,
    expr="publication_date >= 1727704800 and company_name ==
'Elastic'")
```

从 LLM 友好度的角度看 Milvus 与 Elasticsearch 的核心区别

接口的开放性与灵活性

Elasticsearch: 生成任务是 **“文本到JSON” (Text-to-JSON) 或 “文本到SQL” (Text-to-SQL)**。

- 这是当前所有主流LLM的**原生核心能力**。它们在大型代码和文本语料库上进行了训练，能够非常自然地将指令转换为结构化的JSON或SQL。你可以给它几个示例（few-shot prompting），它就能举一反三，生成任意复杂的查询。

Milvus: 生成任务更接近 **“文本到函数调用” (Text-to-Function-Call) 或 “文本到代码” (Text-to-Code)**。

- 这要求LLM遵循更严格的“工具使用”（Tool Use）或“函数调用”（Function Calling）范式。LLM不仅要知道函数名 (`search`)，还要知道每个参数的名称 (`data`, `expr`, `limit`) 和数据类型。
- **关键障碍**: `query_vector` 本身是浮点数数组，它不是文本，LLM无法直接“生成”它。它必须调用另一个工具（嵌入模型）来生成这个向量，然后才能填充到 `search` 函数的参数中。这引入了额外的步骤和复杂性。

总结对比表

特性	Elasticsearch	Milvus
核心交互模式	**声明式** (发送一个描述你想要什么的文档)	**过程式/命令式** (调用一个函数去做某件事)
LLM 生成目标	结构化文本 (JSON / ESQL)	**函数调用参数** 或 **SDK代码片段**
查询“语言”	Query DSL, ESQL (自包含、富有表现力)	过滤表达式 (`expr`) 字符串 + 函数参数
对LLM的友好度	**极高** (Text-to-JSON是LLM的核心优势)	**中等** (依赖于更复杂的Function Calling/Tool Use范式)
处理向量的方式	**内置 embedding 转换能力，无需传递向量**	向量是函数调用的一个核心**数据参数**，需提前准备
集成便利性	**极高** (通用REST API，无需特定SDK)	**中等** (SDK优先，需要特定客户端库)

elastic

Search and Query Operations

- `milvus_text_search` : Search for documents using full text search
  - Parameters:
    - `collection_name` : Name of collection to search
    - `query_text` : Text to search for
    - `limit` : The maximum number of results to return (default: 5)
    - `output_fields` : Fields to include in results
    - `drop_ratio` : Proportion of low-frequency terms to ignore (0.0-1.0)
- `milvus_vector_search` : Perform vector similarity search on a collection
  - Parameters:
    - `collection_name` : Name of collection to search
    - `vector` : Query vector
    - `vector_field` : Field name for vector search (default: "vector")
    - `limit` : The maximum number of results to return (default: 5)
    - `output_fields` : Fields to include in results
    - `filter_expr` : Filter expression
    - `metric_type` : Distance metric (COSINE, L2, IP) (default: "COSINE")
- `milvus_hybrid_search` : Perform hybrid search on a collection
  - Parameters:
    - `collection_name` : Name of collection to search
    - `query_text` : Text query for search
    - `text_field` : Field name for text search
    - `vector` : Vector of the text query
    - `vector_field` : Field name for vector search
    - `limit` : The maximum number of results to return
    - `output_fields` : Fields to include in results
    - `filter_expr` : Filter expression
- `milvus_query` : Query collection using filter expressions
  - Parameters:
    - `collection_name` : Name of collection to query
    - `filter_expr` : Filter expression (e.g. 'age > 20')
    - `output_fields` : Fields to include in results
    - `limit` : The maximum number of results to return (default: 10)

```rust
//------------------------------------------------------------------------------
/// Tool: ES|QL
#[tool(
    description = "Perform an Elasticsearch ES|QL query.",
    annotations(title = "Elasticsearch ES|QL query", read_only_hint = true)
)]
async fn esql(
    &self,
    req_ctx: RequestContext<RoleServer>,
    Parameters(EsqlQueryParams { query }): Parameters<EsqlQueryParams>,
) -> Result<CallToolResult, rmcp::Error> {
    let es_client = self.es_client.get(req_ctx);

    let request = EsqlQueryRequest { query };

    let response = es_client.esql().query().body(request).send().await;
    let response: EsqlQueryResponse = read_json(response).await?;

    // Transform response into an array of objects
    let mut objects: Vec<Value> = Vec::new();
    for row in response.values.into_iter() {
        let mut obj = Map::new();
        for (i, value) in row.into_iter().enumerate() {
            obj.insert(response.columns[i].name.clone(), value);
        }
        objects.push(Value::Object(obj));
    }

    Ok(CallToolResult::success(vec![
        Content::text("Results"),
        Content::json(objects)?,
    ]))
}
```

从 LLM 友好度的角度看 Milvus 与 Elasticsearch 的核心区别

知识库广度与 LLM 可学习性

一个引擎对 LLM 的友好程度，不仅取决于其技术设计，还取决于 LLM 在其庞大的训练语料中“学习”该引擎的难易程度。

- **LLM 的学习源泉**：LLM 的能力源于其对海量互联网文本的训练。因此，一个引擎的公开文档、技术博客、教程、论坛讨论（如 Stack Overflow）以及代码示例（如 GitHub）越丰富，LLM 就越有可能准确、深入地理解其查询语法、API 用法乃至运维和调优的最佳实践。

- **流行度作为可学习性的代理指标**：像 DB-Engines 这样的行业排名，其评分标准恰恰包含了搜索引擎提及率、技术论坛讨论热度、社交媒体活跃度和招聘需求等，这些都是衡量一个技术生态系统公共知识库规模的有效代理指标。一个在这些方面得分远超对手的引擎，意味着 LLM 在训练过程中接触到了数量级更多的相关知识。

- **从“知道”到“精通”**：这种知识广度的差异，决定了 LLM 在与引擎交互时，是从仅仅能生成基础查询，还是能够构建出复杂、高效、符合最佳实践的查询。对于后者，LLM 几乎将该引擎的查询语言视为一种“母语”，从而极大地降低了集成开发的门槛和成本。

- **对 LLM 的意义**：巨大的得分差距意味着，一个通用的 LLM 在其训练数据中见过 Elasticsearch 查询、配置和问题解决方案的次数，可能比见过其他所有向量数据库相关内容的次数加起来还要多。这使得 LLM 在生成 Elasticsearch 的 Query DSL 或 ES|QL 时，表现得更加“自信”和准确，能够处理更复杂的查询逻辑，因为这已经成为其知识库中一个被充分学习和理解的部分。

Ranking > Vector DBMS

DB-Engines Ranking of Vector DBMS

The DB-Engines Ranking ranks database management systems according to their popularity. The ranking is updated monthly.

This is a partial list of the complete ranking showing only vector DBMS.

Read more about the method of calculating the scores.

☐ include secondary database models　　22 systems in ranking, August 2025

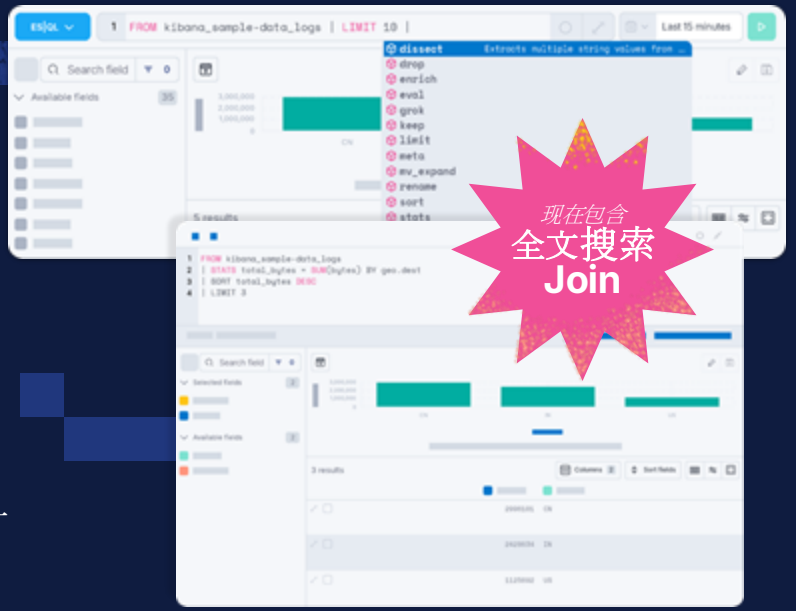
Rank			DBMS	Database Model	Score		
Aug 2025	Jul 2025	Aug 2024			Aug 2025	Jul 2025	Aug 2024
1.	1.	1.	Elasticsearch	Multi-model	114.27	-4.56	-15.56
2.	2.	2.	OpenSearch	Multi-model	18.48	+1.18	+2.01
3.	3.	3.	Couchbase	Multi-model	12.67	-0.29	-3.53
4.	4.	4.	Kdb	Multi-model	6.98	-0.12	-0.99
5.	5.	↑ 6.	Pinecone	Vector	4.95	+0.66	+1.81
6.	6.	↓ 5.	Milvus	Vector	3.83	+0.47	+0.45
7.	7.	↑ 10.	Qdrant	Vector	2.92	+0.39	+1.54
8.	8.	↑ 9.	OceanBase	Multi-model	2.82	+0.31	+0.93
9.	9.	↑ 11.	Weaviate	Vector	2.28	+0.13	+0.99
10.	↑ 11.	↓ 8.	Chroma	Vector	2.22	+0.21	+0.33

elastic

立即可用

ES | QL

- 跨结构化和非结构化数据的管道查询
- 来自日志、指标、跟踪等的实时洞察
- 跨域数据关联以更快解决问题
- 定位+上下文丰富+转换+聚合+洞察
- 允许LLM通过生成一段简洁的工作流脚本来编排一系列复杂的任务



摄取 处理 存储与复制 **搜索** AI和ML分析 **可视化** 工作流程自动化

搜索AI湖

可以使用所有的ES|QL运算符

分类	函数/运算名
数据分析与挖掘	AVG、COUNT、COUNT_DISTINCT、MAX、MEDIAN、MEDIAN_ABSOLUTE_DEVIATION、MIN、PERCENTILE、ST_CENTROID_AGG、SUM、TOP、VALUES、WEIGHTED_AVG、MV_AVG、MV_COUNT、MV_MAX、MV_MEDIAN、MV_MEDIAN_ABSOLUTE_DEVIATION、MV_MIN、MV_PERCENTILE、MV_PSERIES_WEIGHTED_SUM、MV_SUM、CATEGORIZE、LOOKUP JOIN
条件判断与转换	CASE、COALESCE、GREATEST、LEAST、TO_BOOLEAN、TO_DOUBLE、TO_INTEGER、TO_LONG、TO_STRING、TO_UNSIGNED_LONG、TO_CARTESIANPOINT、TO_CARTESIANSHAPE、TO_DATEPERIOD、TO_DATETIME、TO_DEGREES、TO_GEOPOINT、TO_GEOSHAPE、TO_IP、TO_RADIANS、TO_TIMEDURATION、TO_VERSION
文本与字符串处理	BIT_LENGTH、BYTE_LENGTH、CONCAT、ENDS_WITH、FROM_BASE64、LEFT、LENGTH、LOCATE、LTRIM、REPEAT、REPLACE、REVERSE、RIGHT、RTRIM、SPACE、SPLIT、STARTS_WITH、SUBSTRING、TO_BASE64、TO_LOWER、TO_UPPER、TRIM、MV_APPEND、MV_CONCAT、MV_ZIP
特定领域操作	DATE_DIFF、DATE_EXTRACT、DATE_FORMAT、DATE_PARSE、DATE_TRUNC、NOW、CIDR_MATCH、IP_PREFIX、ST_DISTANCE、ST_INTERSECTS、ST_DISJOINT、ST_CONTAINS、ST_WITHIN、ST_X、ST_Y、MATCH、QSTR、BUCKET
运算与逻辑操作	等于（=）、不等于（!=）、小于（<）、小于等于（<=）、大于（>）、大于等于（>=）、加（+）、减（-）、乘（*）、除（/）、取模（%）、否定（-）、AND、OR、NOT、IS NULL、IS NOT NULL、Cast（::）、IN、LIKE、RLIKE、match（:）、MV_DEDUPE、MV_SORT、MV_FIRST、MV_LAST、MV_SLICE

问题在发展...
搜索也应该如此

在单个ES|QL查询中，自然语言查询如“什么是ES|QL”是可组合的阶段

1. 初始结果检索
2. RRF合并 + 语义重排序结果
3. LLM对前n个结果进行总结

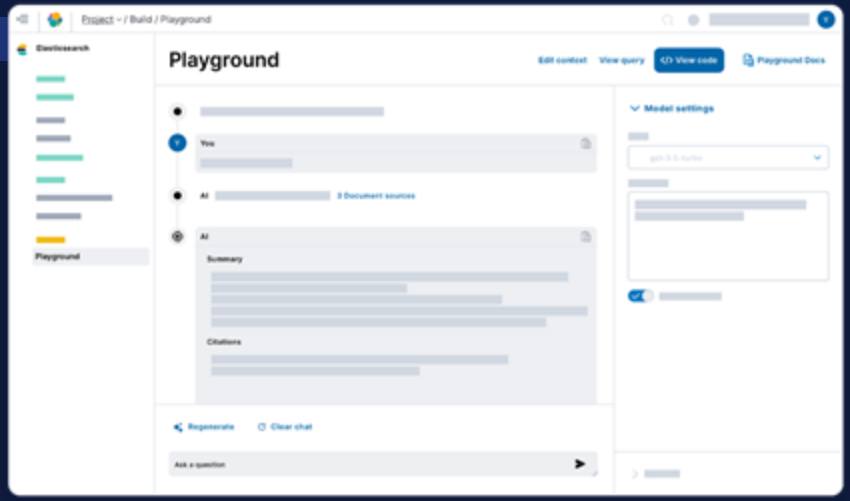
在Elasticsearch中，RAG就像一个简单的ES|QL查询


```
FROM search-index METADATA _score
| FORK (WHERE semantic_content:"what's esql?"
        | SORT _score DESC)
       (WHERE content:"what's esql?"
        | SORT _score DESC)
| RRF
| RERANK "what's esql" ON content WITH "elastic_rerank"
| LIMIT 6
| COMPLETION CONCAT("Summarize :", content) WITH
  "elasticllm" AS summary
```

现已推出

AI游乐场

- 快速原型检索增强生成（RAG）应用程序
- A/B测试各种大型语言模型（LLM）
- 集成专有数据以增强AI响应



现在可用

Better Binary Quantization (BBQ)

- 加速向量量化时间 20x - 30x
- 减少约 95% 的向量内存需求
- 加速向量搜索性能
- 保持高检索准确性

摄取

过程

存储和复制

搜索

AI 和 ML 分析

可视化

工作流自动化

搜索 AI 数据湖

帮助您充分利用所有数据的新功能

存储与复制

可视化

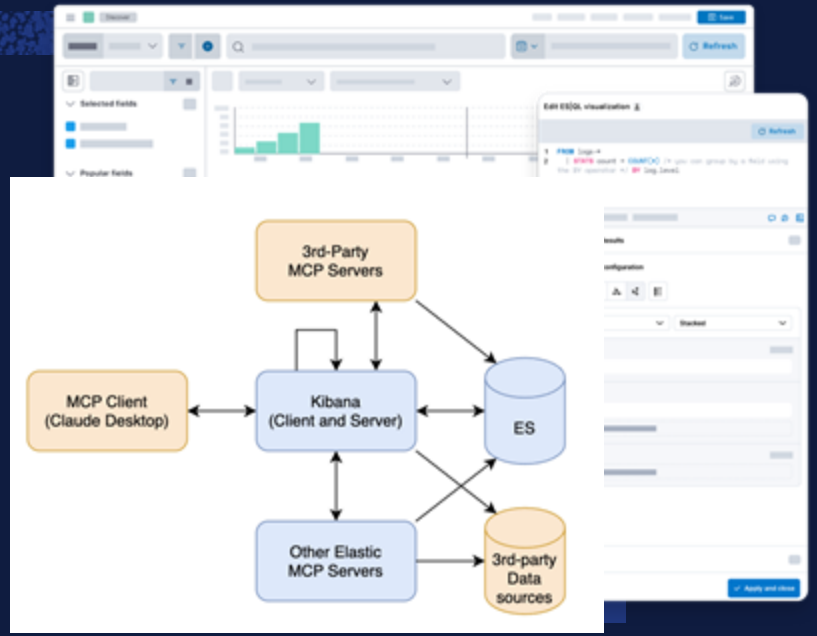
工作流动画

elastic

正在推出

MCP Server & Tools

- MCP 通过标准化应用程序如何向 LLM 提供上下文来增强 Elastic 的功能
- 它允许 LLM 根据上下文选择可用的工具，从而实现更智能的交互
- MCP 能够将 Elastic 功能和第三方功能无缝集成到 Elastic 中



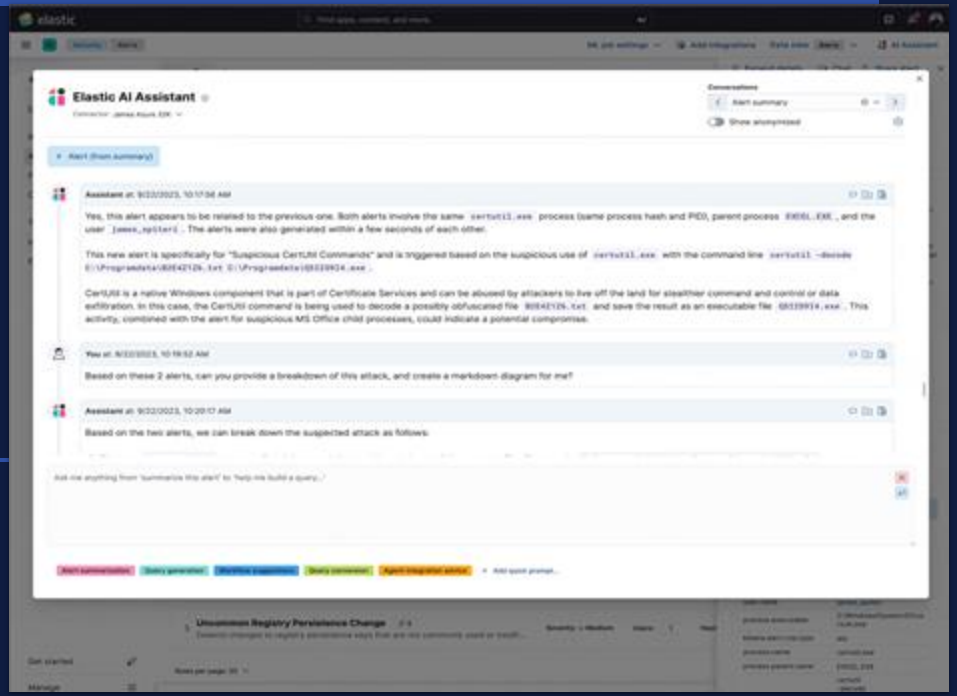
数据摄取　过程　存储与复制　**搜索**　AI和ML分析　可视化　**工作流自动化**

搜索AI湖

现已推出

AI 助手

- 通过分流、调查和响应指导加速事件解决
- 减少在例行任务上花费的时间
- 跨信号类型的标准化架构实现相关洞察
- 包括运行手册（和更多）以提供可操作的、上下文丰富的指导



企业应该如何技术选项？

一站式解决方案



优势: 快速验证 MVP | 降低初期成本 | 开箱即用

劣势: 技术支持不足 | 长期维护风险 | 性能瓶颈
缺乏深度定制 | 隐私和合规性 | 集成困难

优势: 快速验证 MVP | 降低初期成本 | 开箱即用

劣势: 模型优化受限 | 隐私和合规性 | 供应商锁定

企业级 AI 基础设施平台

优势: 企业级平台与服务 | 兼顾敏捷与规模
专注业务价值 | 安全、稳定、高效

完全自建



优势: 技术自主性高 | 降低长期成本 | 灵活，安全

劣势: 技术门槛高 | 集成复杂 | 维护负担大

通过端到端能力和功能更快地构建 RAG 应用程序

	数据摄取	推理	检索	LLM 集成	评估	LLM 可观察性	LLM 安全
现已提供	●语言客户端 ●连接器 ●网页爬虫 ●自动分块 ●更好的二进制量化 ●标量量化（默认int 8）	●ELSER（英文） ●E5（多语言） ●自定义模型 ●用于嵌入、重排和LLM的开放推理API ●OpenAI，Vertex AI，Hugging Face等	●向量搜索 ●混合搜索与简单RRF ●语义重排 ●应用业务标准与查询规则 ●原生学习排序(LTR) ●Elastic重排模型	●OpenAI ●Azure OpenAI ●Azure AI Studio ●Amazon Bedrock ●Cohere ●Mistral ●阿里云 ●Anthropic ●Google Vertex AI ●Google AI Studio ●Watsonx.ai	●行为分析 ●AI游乐场 ●AutoOps	●Azure OpenAI可观察性 ●Google Vertex AI可观察性 ●令牌追踪	●LLM检测规则
即将推出	●更多生态系统工具集成 ●连接器 SDK ●完全托管的无代理服务数据摄取	●用于 GPU 的 Elastic 推理服务 集成推理的 ES|QL	● 查询重写/理解	●Elastic LLM ●私有 LLM ●百度文心一言 ●Tsuzumi	●RAG 评估框架 ●相关性基准测试 ●A/B测试	●RAG 的 APM（追踪） ●查询日志和指标 ●成本追踪	●更多 LLM 检测规则 ●LLM 保护服务 ●workflow

elastic

任何以数据为中心的用例，一个搜索 AI 平台

可观测性

利用人工智能加速问题解决以提高性能

提高AI应用韧性

安全

使用AI自动化保护以预防和补救事件

降低AI应用风险

搜索

构建生成AI体验，以创新和自动化数据的实时洞察

提升AI应用体验

数据库

传统系统

应用程序

SaaS应用程序

Web服务

文件

公共云

本地部署

Elastic Search AI 平台

摄取

安全 & 可扩展存储

AI / ML

搜索

可视化

工作流自动化

从数据到结果只需（毫）秒

业务成果

客户和员工满意度提高**69%**

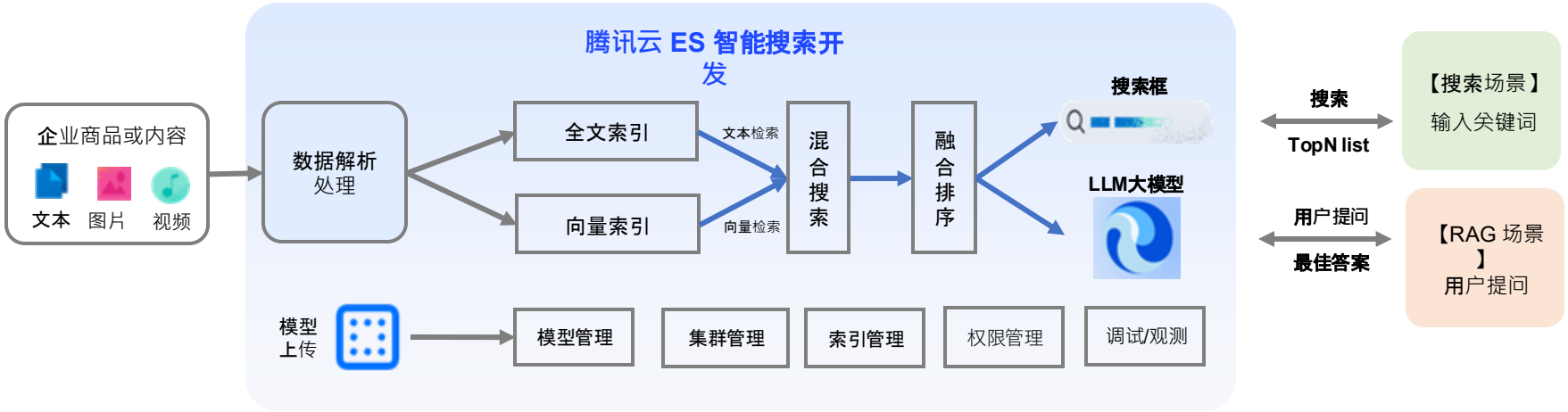
总拥有成本降低**50%**

收入中断减少**62%**

风险降低**60%**

elastic

以统一开放的架构支撑多样化场景



与腾讯云 AI生态联动

基础能力模型

数据解析 | chunk | embedding | query改写 | nlp | rerank

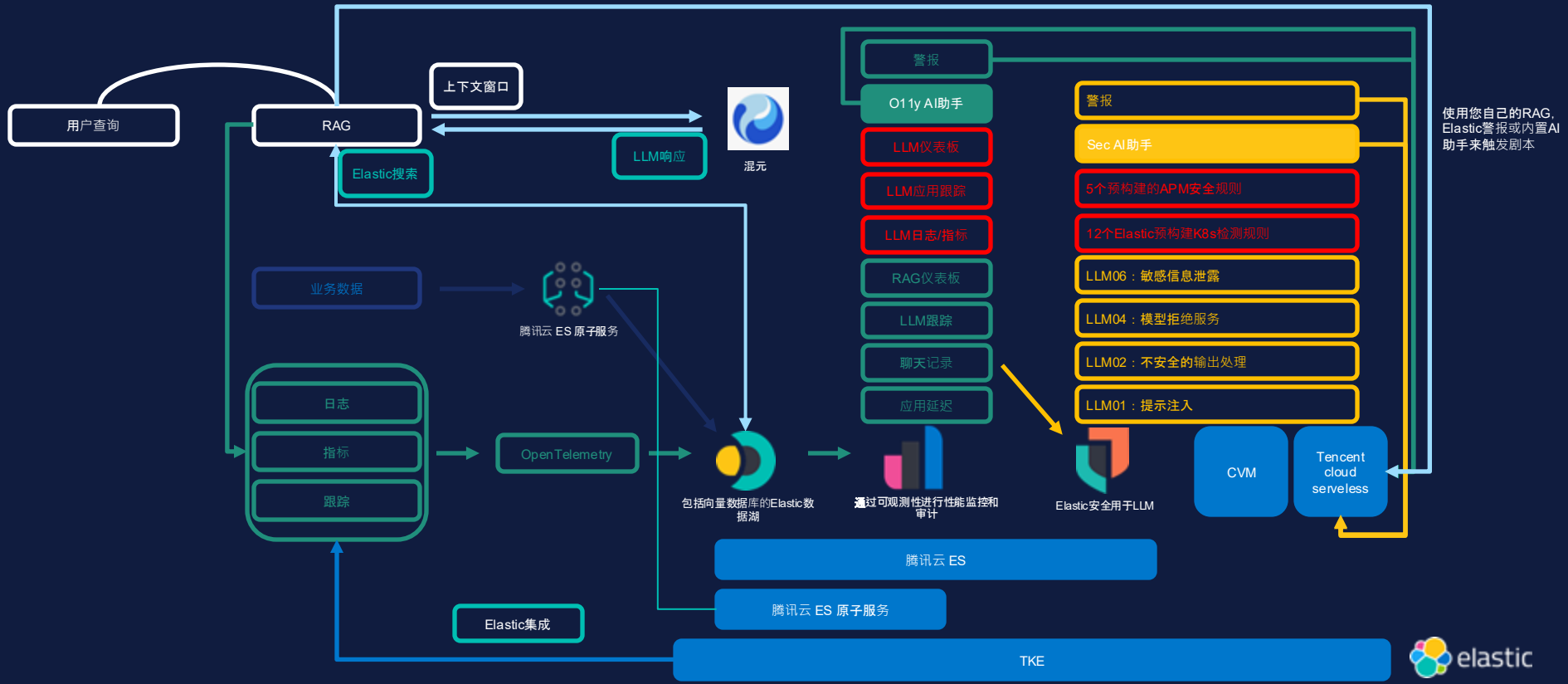
大语言模型

混元大模型 | DeepSeek

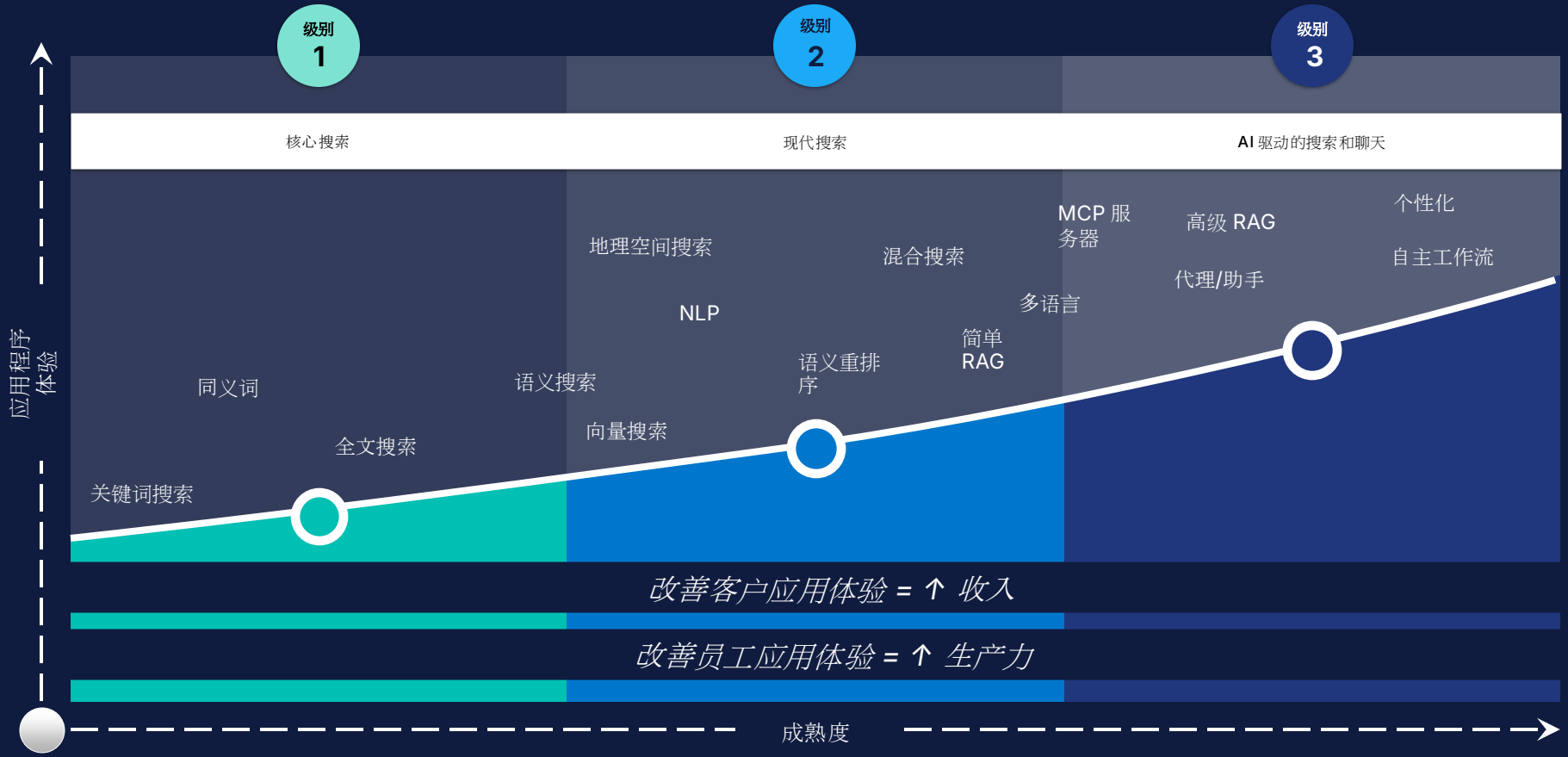
elastic

Elastic 与 Tencent Cloud 的联合解决方案

稳定基础设施 + 敏捷上层开发



客户旅程


级别 1
级别 2
级别 3
核心搜索
现代搜索
AI 驱动的搜索和聊天
应用程序体验
关键词搜索
同义词
全文搜索
语义搜索
地理空间搜索
向量搜索
NLP
语义重排序
混合搜索
简单 RAG
多语言
MCP 服务器
代理/助手
高级 RAG
自主工作流
个性化
改善客户应用体验 = ↑ 收入
改善员工应用体验 = ↑ 生产力
成熟度

产品路线图

向量数据库

- 开箱即用的高效量化（例如默认开启BBQ）
- 更快和更低内存索引，磁盘友好搜索（例如IVF）
- 更多底层算法优化（例如Acorn）
- 成本可预测性和透明性

使用AI搜索

- 第一方模型：ELSER，Elastic Rerank，计划更多
- ES|QL 用于搜索
- 混合搜索和检索器
- 原生GPU嵌入和LLM推理服务
- 简单的RAG实验和工具

在您的私有数据上聊天

- 原生MCP与LLM驱动的工具
- 原生聊天体验
- Elastic托管的关键业务数据源连接器
- 自动化相关性用于封闭域工作流和评估

Elasticsearch平台

- 仪表盘
- 发现
- 语言客户端
- 弹性推理
- 自动运维
- 无服务器

elastic