Chapter 1

Model Context Protocol (MCP)

1.1 Introduction to MCP

The Model Context Protocol (MCP) is an open protocol standardized by Anthropic that enables seamless integration between AI applications and external data sources. In the context of our project, MCP serves as the foundational layer that allows the AI assistant to access and interact with the knowledge library, providing contextual information to enhance conversational responses.

1.1.1 The Problem MCP Solves

AI-enabled tools are powerful, but they're often limited to the information manually provided or require bespoke integrations. Traditional AI applications face several challenges :

- **Limited Context** : AI models can only work with information provided in prompts
- **Manual Data Provision** : Users must manually copy and paste relevant information
- **Bespoke Integrations** : Each data source requires custom, hard-coded integrations
- **No Standardization** : Different AI applications implement data access differently

Whether it's reading files from your computer, searching through an internal or external knowledge base, or updating tasks in a project management tool, MCP provides a **secure, standardized, and simple** way to give AI systems the context they need.

1.2 MCP Architecture Overview

1.2.1 Client-Server Architecture

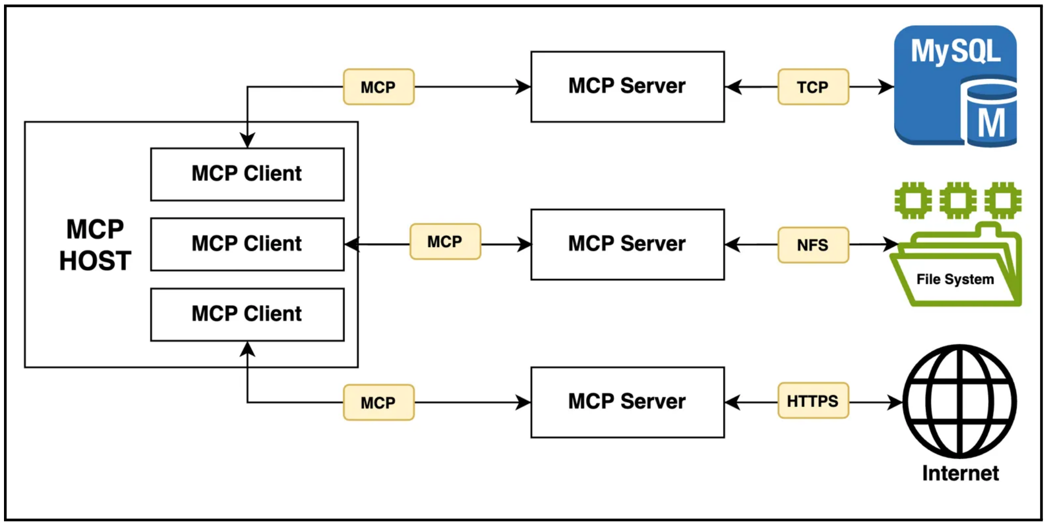

Figure 1.1 – MCP Client-Server Architecture

1.2.2 Key Participants

The MCP architecture involves three primary participants :

1. **MCP Host** : The AI application that coordinates and manages one or multiple MCP clients (e.g., Visual Studio Code, Claude Desktop, One Place Chat)
2. **MCP Client** : A component that maintains a connection to an MCP server and obtains context from an MCP server for the MCP host to use
3. **MCP Server** : A program that provides context to MCP clients. Servers can run locally (using STDIO transport) or remotely (using HTTP transport)

1.3 MCP Primitives

MCP defines primitives that specify what context can be shared between clients and servers. These are the core building blocks of the protocol.

1.3.1 Server Primitives

Servers expose three main types of primitives to provide context to AI applications :

Primitive	**Description**
Tools	Executable functions that AI applications can invoke to perform actions (e.g., file operations, API calls, database queries). Tools are discovered via `tools/list` and executed via `tools/call`.
Resources	Data sources that provide contextual information to AI applications (e.g., file contents, database records, API responses). Accessed via `resources/list` and `resources/read`.
Prompts	Reusable templates that help structure interactions with language models (e.g., system prompts, few-shot examples). Retrieved via `prompts/list` and `prompts/get`.

Table 1.1 – MCP Server Primitives

1.3.2 Client Primitives

Clients expose primitives that allow servers to build richer interactions :
- **Sampling** : Allows servers to request language model completions from the client's AI application using `sampling/complete`
- **Elicitation** : Enables servers to request additional information from users via `elicitation/request`
- **Logging** : Allows servers to send log messages to clients for debugging and monitoring

1.4 MCP Communication Flow

The typical MCP interaction follows a well-defined sequence :

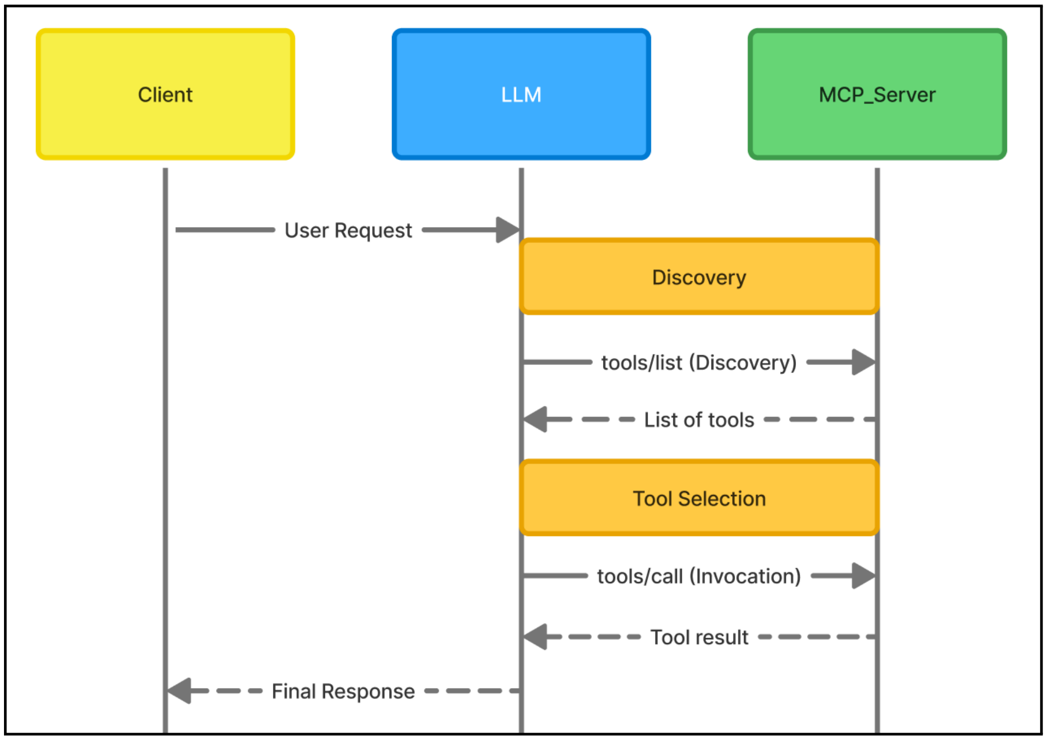

FIGURE 1.2 – MCP Communication Workflow

1.4.1 Understanding the MCP Workflow

1.4.1.1 The Discovery Phase

When a user sends a request to the AI application through the client interface, the Large Language Model (LLM) initiates a discovery phase :

1. The LLM sends a `tools/list` request to the MCP server
2. The MCP server responds with a comprehensive list of all available tools that have been previously implemented and stored
3. Each tool in the list includes its name, description, and input schema (parameters it accepts)

1.4.1.2 The Tool Selection Phase

Once the LLM receives the list of available tools, it enters the tool selection phase :

1. The LLM analyzes the user's request against the available tools
2. Based on the tool descriptions and schemas, the LLM intelligently selects the most appropriate tool(s) to fulfill the user's request

3. The LLM then invokes the selected tool using `tools/call` with appropriate arguments
4. The MCP server executes the pre-coded tool logic and returns the result
5. The LLM incorporates the tool result into its response to the user

1.4.1.3 Limitations and Design Implications

This architecture has important implications for system design :

— **Pre-implementation Required** : Every capability the AI needs must be implemented as a tool beforehand

— **Schema Definition** : Each tool must have a well-defined input schema so the LLM knows how to invoke it correctly

— **Limited Flexibility** : The AI cannot create new tools or capabilities on the fly ; it can only use what has been pre-coded

— **Deterministic Behavior** : Since tools are pre-defined, their behavior is predictable and can be tested thoroughly

This design ensures security, reliability, and predictability, as the AI can only perform actions that have been explicitly programmed and authorized by developers.

1.5 Conclusion

The Model Context Protocol represents a significant advancement in how AI applications interact with external data sources. By providing a standardized, secure framework for context exchange, MCP eliminates the need for bespoke integrations while maintaining strict control over AI capabilities through pre-defined tools.

Chapter 2

Problem Statement and Proposed Solution

2.1 Introduction

Integrating AI assistants with external APIs presents significant challenges in traditional implementations. This chapter examines these challenges and presents One Place Chat's innovative solution that enables natural language API interactions through intelligent tool generation and semantic matching.

2.2 Traditional API Integration Challenges

2.2.1 The Manual Integration Problem

Traditional approaches to integrating AI systems with APIs face several fundamental challenges that limit scalability and user experience.

2.2.1.1 Manual Tool Definition

The traditional workflow requires developers to manually write code for each API endpoint the AI needs to access. This includes defining request structures, parameter schemas, response parsing, and error handling. For an API with dozens or hundreds of endpoints, this becomes a massive development bottleneck.

2.2.2 Identified Limitations

This manual approach introduces several practical limitations :

— **Development Bottleneck** : Each new API endpoint requires separate implementation, testing, and deployment cycles. Adding support for a new API service can take days or weeks of development time.

— **Maintenance Burden** : As APIs evolve and endpoints change, each integration must be manually updated. API versioning requires duplicate implementations or complex version management logic.

— **Limited Scalability** : Supporting multiple APIs with hundreds of endpoints becomes impractical when each requires manual coding. The codebase grows linearly with the number of supported endpoints.

— **Poor User Experience** : Users must learn specific command syntax or interact with rigid interfaces. Natural language understanding is limited or requires extensive prompt engineering.

2.2.3 Real-World Impact

Consider a practical scenario : An organization wants to integrate their AI assistant with three different API services (user management, inventory, and analytics), each with 50+ endpoints. In a traditional implementation, this would require :

— Manual coding of 150+ individual endpoint handlers
— Custom parameter validation logic for each endpoint
— Separate error handling for each API service

This approach quickly becomes unsustainable as the number of APIs and endpoints grows.

2.3 Proposed Solution : One-Place-Chat

One Place Chat introduces an innovative three-stage approach that transforms API integration from a manual coding process into an automated, intelligent system :

1. **Tool Generation** : Convert OpenAPI specifications into executable **tool definitions**
2. **Vector-Based Storage** : Store tools with semantic embeddings for intelligent retrieval
3. **Natural Language Processing** : Use LLM-powered semantic matching and parameter extraction

2.3.0.1 Core Innovation

The key innovation lies in shifting from **manual API coding** to **automated semantic tool discovery** :

— **Traditional Approach** : Manual coding → Fixed endpoints → Limited scalability
— **One Place Chat** : OpenAPI parsing → Vector embeddings → Semantic matching → Natural language processing

2.3.1 Architecture of the Solution

2.3.1.1 Stage 1 : Automated Tool Generation

Instead of manually coding API integrations, One Place Chat automatically generates tool definitions from OpenAPI specifications :

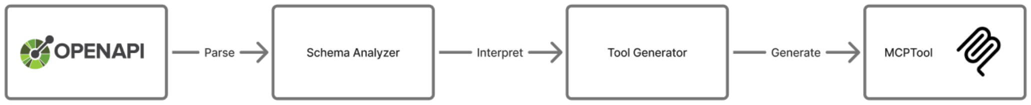

FIGURE 2.1 – MCP Communication Workflow

— Extracts all API endpoints and operations
— Generates input schemas from parameter definitions
— Creates tool descriptions from API documentation
— Produces standardized `MCPTool` objects

2.3.1.2 Stage 2 : Vector-Based Storage with ChromaDB

Generated tools are stored in a vector database with semantic embeddings for intelligent retrieval :

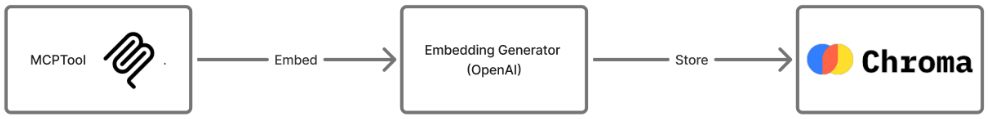

Figure 2.2 – MCP Communication Workflow

For each tool, the system :
— Creates a text representation combining name, description, path, and tags
— Generates vector embeddings using OpenAI's '`text-embedding-ada-002`' model
— Stores the tool definition and embedding in ChromaDB collections
— Enables semantic similarity search for intelligent tool matching

2.3.1.3 Stage 3 : Semantic Matching and Natural Language Processing

When users make requests in natural language, the system uses semantic search to find the right tool :

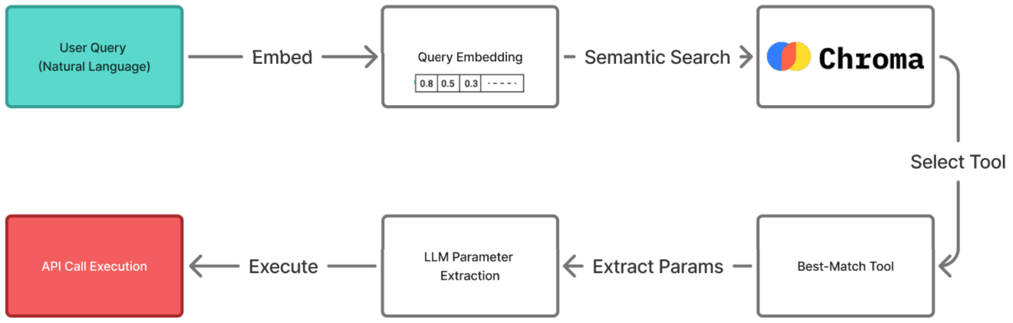

Figure 2.3 – MCP Communication Workflow

2.3.2 Implementation Strategy

Our solution implements the three-stage approach through specialized components :

1. **OpenAPI Tool Parser** : Automatically converts OpenAPI specifications into executable tool definitions with complete schemas and parameter mappings.
2. **ChromaDB Tool Loader** : Dynamically loads tools from the vector database without caching, ensuring fresh data and providing semantic search capabilities.
3. **ChromaDB Tool Matcher** : Generates OpenAI embeddings for tools and performs similarity-based matching using vector distances to find the most relevant tool.

2.3.3 Key Advantages

Aspect	**Traditional Approach**	**One Place Chat**
Tool Creation	Manual coding per endpoint	Automatic from OpenAPI specs
Scalability	Linear development effort	Handles unlimited endpoints
Intent Matching	Keyword-based, rigid	Semantic embeddings, flexible
Maintenance	Update each integration	Re-parse updated spec
Development Time	Days per API	Minutes per API
User Interface	Specific syntax required	Natural language

Table 2.1 – Comparison : Traditional Approach vs One Place Chat

2.4 Conclusion

One Place Chat fundamentally transforms API integration by introducing a three-stage architecture that eliminates manual coding entirely : OpenAPI specifications are automatically parsed into tool definitions, stored as vector embeddings in ChromaDB for semantic retrieval, and matched to user intent through natural language processing. This innovation shifts the paradigm from labor-intensive, endpoint-by-endpoint development to an intelligent, automated system where adding a new API takes minutes instead of weeks, where users interact conversationally instead of learning rigid syntax, and where semantic understanding replaces brittle keyword matching. The result is a scalable, maintainable solution that demonstrates how combining standardized API specifications with modern vector databases and large language models can create truly intelligent interfaces that understand intent, adapt dynamically, and deliver superior user experience without sacrificing technical accuracy or reliability.

Conclusion

Au terme de ce projet d'innovation mené dans le cadre de notre formation d'ingénieurs à l'École Normale Supérieure de l'Enseignement Technique de Mohammedia (ENSET-M), la réalisation de la plateforme JobAI représente bien plus qu'un simple aboutissement académique. Elle constitue une véritable expérience entrepreneuriale qui nous a permis d'appréhender concrètement les défis et les opportunités du développement technologique contemporain.

JobAI s'est imposé comme une solution pertinente face à la complexité croissante du marché de l'emploi. En intégrant l'intelligence artificielle au cœur du processus de recherche d'emploi, notre plateforme répond aux attentes des candidats modernes qui font face à la multiplication des plateformes de recrutement, aux exigences accrues de personnalisation des candidatures, et à la nécessité d'optimiser leur temps et leurs efforts. L'approche holistique adoptée, combinant génération automatisée de documents professionnels, analyse de compatibilité sémantique et agent de candidature autonome, démontre notre capacité à concevoir des solutions technologiques complètes et cohérentes.

Le développement de JobAI nous a offert une immersion authentique dans l'univers des technologies de pointe. La maîtrise des modèles de langage avancés, l'intégration de services cloud, l'automatisation web et le développement d'extensions navigateur ont enrichi notre expertise technique de manière substantielle. Cette expérience pratique a consolidé notre compréhension des enjeux actuels de l'intelligence artificielle appliquée aux contextes professionnels, nous préparant efficacement aux défis technologiques futurs.

JobAI s'inscrit dans la tendance actuelle où l'intelligence artificielle devient un levier stratégique pour repenser et optimiser les processus humains complexes. Notre projet démontre que l'innovation technologique peut apporter des réponses concrètes aux défis contemporains, tout en créant de la valeur pour les utilisateurs finaux. Cette réalisation marque une étape déterminante dans notre parcours de formation, témoignant de notre capacité à transformer des concepts théoriques en solutions pratiques et innovantes.

Nous exprimons notre reconnaissance envers nos superviseurs, AKEF Fatiha et BOUSSELHAM Abdelmajid, ainsi qu'à l'ensemble du corps enseignant de l'ENSET-M pour leur accompagnement et leur expertise qui ont rendu possible la concrétisation de ce projet ambitieux. Leur guidance a été déterminante dans notre capacité à mener à bien cette initiative d'innovation.

En conclusion, JobAI représente non seulement l'aboutissement de notre formation académique, mais également le point de départ d'une réflexion plus large sur les applications de l'intelligence artificielle dans le domaine professionnel. Ce projet nous a dotés des compétences, de la confiance et de la vision nécessaires pour contribuer efficacement au développement technologique de demain, en tant qu'ingénieurs capables de concevoir des solutions innovantes répondant aux besoins réels de la société.

Webographie

1. Documentation de Flask : `https://flask.palletsprojects.com/`
   Page consultée le 3 juin 2025.
2. Documentation de React : `https://react.dev/learn`
   Page consultée le 3 juin 2025.
3. Documentation de Firebase : `https://firebase.google.com/docs`
   Page consultée le 3 juin 2025.
4. Documentation de Manifest V3 (Chrome) : `https://developer.chrome.com/docs/extensions/mv3/intro/`
   Page consultée le 3 juin 2025.
5. Documentation de Tailwind CSS : `https://tailwindcss.com/docs`
   Page consultée le 3 juin 2025.
6. Zety Resume Statistics : Analyse des taux de réponse basée sur 133 000 candidatures.
   Disponible sur : `https://zety.com/blog/resume-statistics`
   Page consultée le 3 juin 2025.
7. ZipJob Application Ratio : Insights sur les taux de réponse des candidatures.
   Disponible sur : `https://zipjob.com/blog/application-to-interview-ratio/`
   Page consultée le 3 juin 2025.
8. ResumeGo Cover Letters : Étude sur l'impact des lettres de motivation personnalisées.
   Disponible sur : `https://www.resumego.net/research/cover-letters/`
   Page consultée le 3 juin 2025.
9. RecruitCRM ATS Stats : Statistiques sur les systèmes de suivi des candidats.
   Disponible sur : `https://recruitcrm.io/blogs/applicant-tracking-system-statistics/`
   Page consultée le 3 juin 2025.
10. Al Game Code : On State of Art #1 : Leaderboard of the Chatbot Arena — LMSYS : A Platform for Crowdsourced Evaluation.
    Disponible sur : `https://medium.com/al-game-code/on-state-of-art-1-leaderboard-of-the-`
    Page consultée le 3 juin 2025.