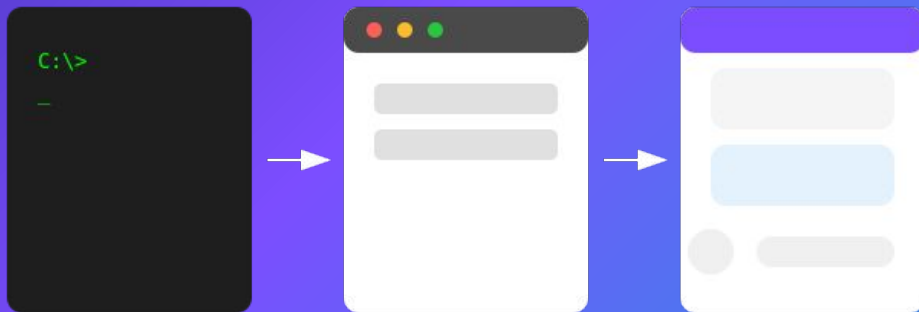


LLM Powered Applications

Evolution of Human-Computer Interaction

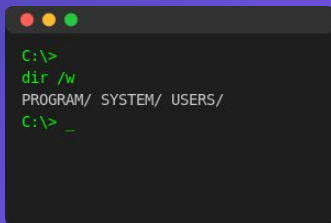


@knezevicdev, 2024

The Journey of Human-Computer Interface

Command Line Era

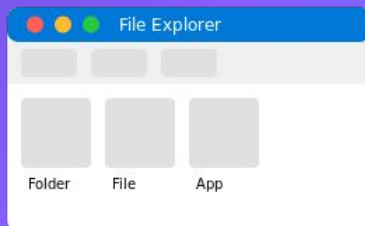
1960s-1980s



- Text-based interaction
- Expert users
- Precise control

Graphical UI Era

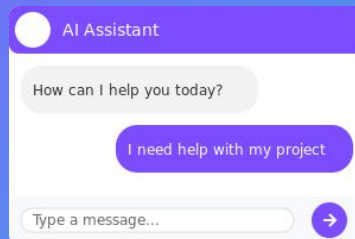
1980s-2020s



- Visual metaphors
- Mass adoption
- WIMP paradigm

LLM Interface Era

2020s+

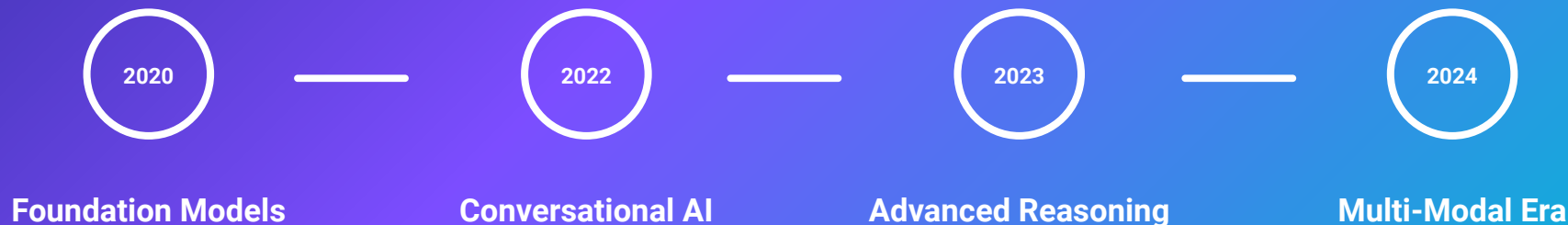


- Natural language
- Context-aware
- Adaptive interaction

State of LLM 2024

Current LLM capabilities:

- Multi-modal understanding (text, images, audio)
- Advanced reasoning and problem solving
- Tool use and API integration



Prompting fundamentals

Basic Prompting

Input:

Translate to Spanish:
Hello, how are you?

Output:

Hola, ¿cómo estás?

Chain of Thought

Input:

Solve: If John has 5
apples and gives 2 to
Mary, how many left?

Output:

Let's solve step by step:
1. Initial: 5 apples
2. Given away: 2
3. Result: $5 - 2 = 3$

Few-Shot Learning

Input:

Example 1:
Input: Happy → Joyful

Example 2:
Input: Sad → Unhappy

Now do:
Input: Angry →

Output: Furious

Advanced Prompting Techniques

Function Calling

```
{  
  "name": "get_weather",  
  "description": "Get weather information for a  
location",  
  "parameters": { "type": "object", "properties": {  
    "location": { "type": "string", "description": "City" }  
  },  
  "required": ["location"]  
}
```

System Prompts

```
{  
  "role": "system",  
  "content": "You are a SQL expert. Always validate  
queries for security and performance. Check for: -  
SQL injection vulnerabilities - Proper indexing -  
Query optimization - Table permissions"  
}
```

Context Window Management

System Context

Role and Rules

Prior Messages

Conversation History

Current Query

User's Request

Memory

Key Information

Building Blocks of LLM Applications



Orchestrators Deep Dive

Key Features

Prompt Management

Chain Composition

Memory Management

Error Handling

Tool Integration

Caching & Optimization

Implementation

```
from langchain import LLMChain
from llama_index import VectorStoreIndex

# define chain components
chain = LLMChain(
    llm=ChatOpenAI(),
    prompt=PromptTemplate(),
    memory=ConversationMemory()
)

# error handling
try:
    response = chain.run(
        input="query"
    )
```

Common patterns

Sequential Chain

Chain of operations where output of one feeds into next

Router Chain

Directs queries to specific chains based on content

Agent Chain

Dynamic tool selection and execution based on task

Building Blocks of LLM Applications



RAG Deep Dive

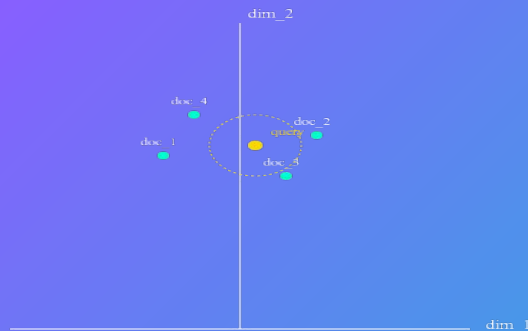
Text to Embeddings

"The quick brown fox jumps"



[0.12, 0.45, -0.23, 0.89, ...]

Vector Database



RAG Pipeline

Query



Embedding



Vector Search



Context



LLM

Building Blocks of LLM Applications



ETL Systems for LLM Applications

Extract

- PDFs, Docs
- Websites, HTML
- Databases
- APIs

Transform

- Text Cleaning
- Chunking
- Metadata Extraction
- Embeddings

Load

- Vector Stores
- Document Stores
- Metadata DB
- Cache Layer

Processing Pipeline

RAW Document



Extract



Clean



Chunk



Store

Building Blocks of LLM Applications



LLM Monitoring Systems



Continuous Improvement Loop: Monitor → Analyze → Collect Feedback → Implement → Validate

Building Blocks of LLM Applications



LLM Testing Strategies

Testing Types

Semantic Similarity

Compare response meaning

Factual Accuracy

Verify information

Format Validation

Check output structure

Response Time

Performance metrics

Tests Examples

```
test_input = "What is the capital of France?"
test_context = "Paris is the capital and largest city of France."
expected_output = "Paris is the capital of France."

# 1. Semantic Similarity - Compare response meaning
semantic_test = SemanticSimilarityMetric(threshold=0.8).measure(
    actual=llm_chain.run(test_input),
    expected=expected_output
).score > 0.8

# 2. Factual Accuracy - Check for hallucinations
factual_test = HallucinationMetric(threshold=0.3).measure(
    context=test_context,
    response=llm_chain.run(test_input)
).score < 0.3

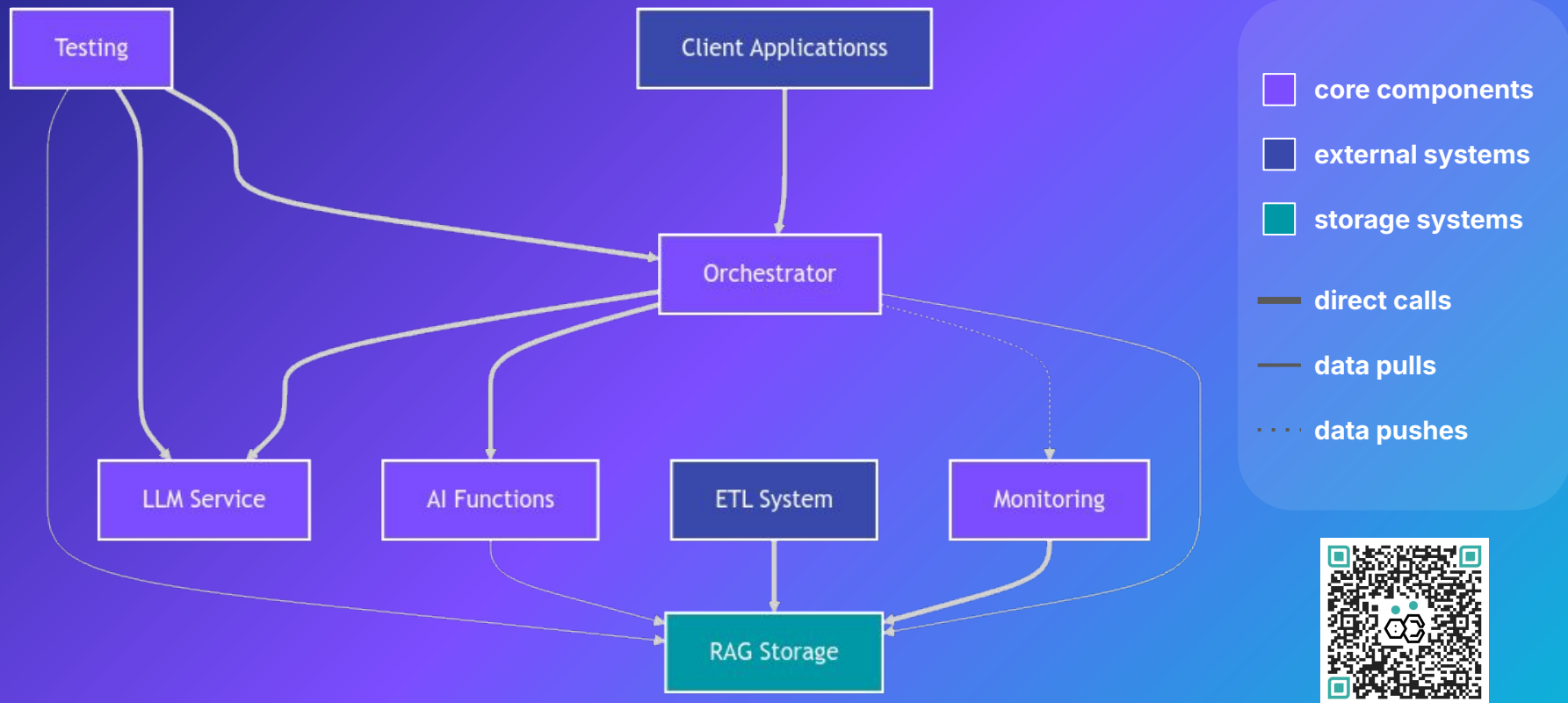
# 3. Format Validation - Verify JSON structure
format_test = ResponseFormatMetric(
    json_schema={"type": "object", "required": ["answer", "confidence"]}
).validate(response=json.dumps({"answer": "Paris", "confidence": 0.95}))

# 4. Response Time - Check performance
latency_test = LatencyMetric(threshold_seconds=2.0).measure(
    lambda: llm_chain.run(test_input)
).score < 2.0
```

Building Blocks of LLM Applications



Modern LLM Application Architecture



Thank you!

