Hanzo Chat: Multi-Model Conversational AI with Tool Integration and Memory

David Wei Marcus Chen Zach Kelling
Hanzo AI Research
research@hanzo.ai

February 2026

Abstract

We present **Hanzo Chat**, a multi-model conversational AI system that enables users to interact with over 100 language models through a unified interface with persistent memory, tool integration, and adaptive model selection. Unlike single-model chat applications, Hanzo Chat introduces three key innovations: (i) a *model routing layer* that dynamically selects optimal models based on query characteristics, cost constraints, and historical performance, achieving 23% higher user satisfaction than fixed-model baselines; (ii) a *hierarchical memory architecture* combining session, project, and global memory stores with retrieval-augmented context injection, supporting conversations spanning weeks without context window limitations; and (iii) a *Model Context Protocol (MCP)* integration layer providing access to 260+ tools including code execution, web search, file manipulation, and API interaction. We evaluate Hanzo Chat on a suite of conversational benchmarks including MT-Bench, ChatBot Arena, and a novel multi-turn tool-use benchmark (MTTU-100), demonstrating that multi-model routing with memory achieves state-of-the-art performance while reducing inference cost by 41% compared to always using the most capable model. We report results from 18 months of production deployment serving over 500,000 conversations, including analysis of model selection patterns, memory utilization, and tool usage statistics.

1 Introduction

The emergence of large language models (LLMs) has transformed human-computer interaction through natural language conversation [15, 1, 7]. However, current conversational AI systems face three fundamental limitations:

1. **Model lock-in**: Users are bound to a single model provider, unable to leverage the complementary strengths of different models (e.g., Claude for reasoning, GPT-4 for code generation, Gemini for multimodal understanding).

2. **Context amnesia**: Conversations are bounded by fixed context windows (4K–200K tokens), and information from previous sessions is lost entirely.

3. **Tool isolation**: Tool integration is model-specific and typically limited to a small set of built-in capabilities.

Hanzo Chat addresses all three limitations through a unified architecture that decouples the user interface from model selection, memory management, and tool execution.

1.1 Design Principles

Hanzo Chat is built on four design principles:

- **Model agnosticism**: Any LLM conforming to the OpenAI Chat Completions API can be integrated, with automatic format translation for non-conforming models.

- **Persistent context**: Memory persists across sessions, projects, and users, with retrieval-augmented injection into the active context window.
- **Tool composability**: Tools are defined via the Model Context Protocol (MCP) standard, enabling composition of arbitrary tool chains.
- **Cost awareness**: Model selection balances quality, latency, and cost according to user-configurable policies.

1.2 Contributions

1. A multi-model routing algorithm that achieves Pareto-optimal quality-cost trade-offs (§3).
2. A hierarchical memory architecture with semantic retrieval and automatic summarization (§4).
3. An MCP-based tool integration framework supporting 260+ tools (§5).
4. Comprehensive evaluation on standard and novel benchmarks (§6).
5. Production deployment analysis from 500K+ conversations (§7).

2 System Architecture

2.1 Overview

Hanzo Chat comprises five layers:

1. **Presentation Layer**: React-based UI with markdown rendering, code highlighting, file previews, and streaming response display.
2. **Routing Layer**: Model selection, load balancing, and failover across providers.
3. **Memory Layer**: Hierarchical storage and retrieval of conversational context.
4. **Tool Layer**: MCP server management, tool discovery, and execution sandboxing.
5. **Gateway Layer**: Unified API proxy (Hanzo LLM Gateway) translating between provider-specific formats.

2.2 Request Flow

A user message m_t at turn t triggers the following pipeline:

Algorithm 1 Hanzo Chat Request Pipeline

Require: User message m_t, conversation history $H_{<t}$, user profile U

$\boldsymbol{e}_t \leftarrow \text{Embed}(m_t)$ ▷ Semantic embedding
$\mathcal{R} \leftarrow \text{RetrieveMemory}(\boldsymbol{e}_t, U)$ ▷ Memory retrieval
$C_t \leftarrow \text{BuildContext}(H_{<t}, \mathcal{R}, m_t)$ ▷ Context assembly
$\hat{M} \leftarrow \text{RouteModel}(C_t, U.\text{policy})$ ▷ Model selection
$\mathcal{T} \leftarrow \text{DiscoverTools}(C_t, \hat{M})$ ▷ Tool discovery
$r_t \leftarrow \text{Generate}(\hat{M}, C_t, \mathcal{T})$ ▷ LLM generation
while r_t contains tool calls **do**
    $\text{results} \leftarrow \text{ExecuteTools}(r_t.\text{tool\_calls})$
    $r_t \leftarrow \text{Generate}(\hat{M}, C_t \cup \text{results}, \mathcal{T})$
end while
$\text{UpdateMemory}(m_t, r_t, \boldsymbol{e}_t)$
return r_t

2.3 Gateway Integration

Hanzo Chat connects to LLM providers through the Hanzo LLM Gateway [8], which provides:

- Unified OpenAI-compatible API for 100+ models across providers (OpenAI, Anthropic, Google, Together, Ollama, etc.).
- Automatic retry with exponential backoff and provider failover.
- Request/response logging for analytics and debugging.
- Token counting and cost tracking per request.
- Rate limiting and quota management per user/organization.

3 Multi-Model Routing

3.1 Problem Formulation

Given a query context C_t and a set of available models $\mathcal{M} = \{M_1, \ldots, M_K\}$, select model $\hat{M}$ that maximizes expected quality subject to cost and latency constraints:

$$\hat{M} = \arg\max_{M \in \mathcal{M}} \mathbb{E}[Q(M, C_t)] \quad (1)$$
$$\text{s.t.} \quad \text{Cost}(M, |C_t|) \leq B_t,$$
$$\text{Latency}(M, |C_t|) \leq L_{\max}.$$

3.2 Query Classification

We classify queries into categories that predict model performance using a lightweight classifier (distilled BERT, 67M parameters) trained on 50K labeled examples:

Category	Best Model Family	Accuracy
Code generation	Claude, GPT-4	94.2%
Mathematical reasoning	Claude, o1	91.7%
Creative writing	GPT-4, Claude	89.3%
Factual Q&A	Gemini, GPT-4	92.8%
Multimodal	Gemini, GPT-4V	88.6%
Summarization	Claude, Gemini	93.1%
Translation	GPT-4, Gemini	90.4%
Tool use	Claude, GPT-4	91.9%

Table 1: Query category classification accuracy and best-performing model families.

3.3 Multi-Armed Bandit Formulation

We model the routing problem as a contextual multi-armed bandit [10] where:

- **Arms**: Available models $\mathcal{M}$.
- **Context**: Query embedding $\boldsymbol{e}_t$, category c_t, conversation length $|H_{<t}|$, user preferences.
- **Reward**: User satisfaction signal (explicit rating, implicit signals such as follow-up corrections, regeneration requests, copy actions).

We use Thompson Sampling with a neural reward model:

Algorithm 2 Neural Thompson Sampling for Model Routing

Require: Context $\boldsymbol{x}_t = (\boldsymbol{e}_t, c_t, |H_{<t}|, U)$, models $\mathcal{M}$
for each model $M_k \in \mathcal{M}$ **do**
  Sample $\hat{\theta}_k \sim \mathcal{N}(\mu_k(\boldsymbol{x}_t), \sigma_k^2(\boldsymbol{x}_t))$
  $\hat{Q}_k \leftarrow f_\theta(\boldsymbol{x}_t, k)$ ▷ Neural reward estimate
  $\hat{R}_k \leftarrow \hat{Q}_k / \text{Cost}(M_k, |\boldsymbol{x}_t|)$ ▷ Quality per dollar
end for
$\hat{M} \leftarrow \arg\max_k \hat{R}_k$ subject to constraints
return $\hat{M}$

The neural reward model f_θ is a 3-layer MLP with ReLU activations, trained online with replay buffer of size 100K. The posterior variance σ_k^2 is estimated via MC Dropout [6].

3.4 Cascading Strategy

For complex queries, we employ a cascading strategy that starts with a cheaper model and escalates if quality is insufficient:

Algorithm 3 Model Cascade

Require: Context C_t, model tiers $[M_{\text{fast}}, M_{\text{mid}}, M_{\text{best}}]$
$r_t \leftarrow \text{Generate}(M_{\text{fast}}, C_t)$
$q \leftarrow \text{QualityEstimate}(r_t, C_t)$
if $q < \tau_1$ **then**
  $r_t \leftarrow \text{Generate}(M_{\text{mid}}, C_t)$
  $q \leftarrow \text{QualityEstimate}(r_t, C_t)$
  if $q < \tau_2$ **then**
    $r_t \leftarrow \text{Generate}(M_{\text{best}}, C_t)$
  end if
end if
return r_t

The quality estimator is a small classifier trained to predict human preference rankings from response features (length, perplexity, tool call correctness, factual consistency).

3.5 Cost Optimization

With the cascade and routing combined, Hanzo Chat achieves significant cost savings:

Strategy	Quality	Cost/query	Latency
Always best model	8.92/10	\$0.042	3.2s
Always cheapest	6.14/10	\$0.003	0.8s
Random selection	7.31/10	\$0.019	1.9s
Hanzo routing	8.73/10	\$0.025	2.1s
Hanzo cascade	**8.81/10**	**\$0.018**	2.4s

Table 2: Quality-cost trade-offs across routing strategies. Hanzo cascade achieves 98.8% of best-model quality at 43% of the cost.

4 Hierarchical Memory Architecture

4.1 Memory Hierarchy

Hanzo Chat implements a three-tier memory hierarchy:

Definition 1 (Memory Tiers)**.**

- ***Session Memory*** *($\mathcal{M}_S$): Full conversation history within the current session. Stored in-memory, evicted on session close. Capacity: full context window.*
- ***Project Memory*** *($\mathcal{M}_P$): Facts, decisions, and summaries relevant to a specific project. Persisted in vector database. Retention: indefinite.*
- ***Global Memory*** *($\mathcal{M}_G$): User preferences, frequently referenced information, and cross-project knowledge. Persisted globally. Retention: indefinite.*

4.2 Memory Storage

Each memory entry is a tuple $(k, v, \boldsymbol{e}, t, s, \rho)$ where k is a unique identifier, v is the content string, $\boldsymbol{e} \in \mathbb{R}^d$ is the embedding vector, t is the creation timestamp, $s \in \{S, P, G\}$ is the scope, and $\rho \in [0, 1]$ is the relevance score.

Embeddings are computed using a fine-tuned E5-large model [24] ($d = 1024$) and stored in a pgvector [18] index for efficient similarity search.

4.3 Memory Retrieval

Given a query embedding $\boldsymbol{e}_t$, we retrieve the top-k memories from each tier:

$$\mathcal{R}_s = \text{Top-}k_s\left(\{m \in \mathcal{M}_s : \cos(\boldsymbol{e}_t, m.\boldsymbol{e}) > \tau_s\}\right), \tag{2}$$

where $k_S = 20$, $k_P = 10$, $k_G = 5$ and $\tau_S = 0.3$, $\tau_P = 0.5$, $\tau_G = 0.6$ are tier-specific thresholds (higher thresholds for broader scopes to maintain relevance).

The combined retrieval set $\mathcal{R} = \mathcal{R}_S \cup \mathcal{R}_P \cup \mathcal{R}_G$ is re-ranked by a cross-encoder [13] and truncated to fit within the context budget.

4.4 Automatic Summarization

When session memory exceeds a threshold (default: 75% of context window), older conversation turns are automatically summarized:

Algorithm 4 Adaptive Memory Summarization

Require: Session history $H = [h_1, \ldots, h_T]$, context budget B

```
tokens ← CountTokens(H)
while tokens > 0.75 · B do
    oldest ← H[1 : ⌊T/3⌋]                ▷ Oldest third
    summary ← LLM.Summarize(oldest)
    Extract facts F ← LLM.ExtractFacts(oldest)
    Store F in project memory M_P
    H ← [summary] ∪ H[⌊T/3⌋ + 1 : T]
    tokens ← CountTokens(H)
end while
return H
```

This process is transparent to the user: the full conversation appears continuous even though the underlying representation has been compressed.

4.5 Memory Consolidation

We periodically consolidate memories across tiers using a process inspired by hippocampal replay in neuroscience [12]:

1. **Deduplication**: Merge memories with cosine similarity > 0.95.
2. **Promotion**: Session memories accessed > 3 times are promoted to project memory.
3. **Generalization**: Cluster project memories and extract general rules for global memory.

4. **Decay**: Memories not accessed for 90 days have their relevance score decayed by 10%.

4.6 Context Window Management

The context window is partitioned as follows:

Section	Budget (%)	Example (128K)
System prompt	5%	6,400 tokens
Global memory	5%	6,400 tokens
Project memory	10%	12,800 tokens
Retrieved context	15%	19,200 tokens
Conversation history	50%	64,000 tokens
Tool definitions	5%	6,400 tokens
Current query	5%	6,400 tokens
Generation headroom	5%	6,400 tokens

Table 3: Context window budget allocation for a 128K token model.

5 Tool Integration via MCP

5.1 Model Context Protocol

The Model Context Protocol (MCP) [2] provides a standardized interface for LLMs to discover and invoke external tools. Hanzo Chat implements an MCP client that connects to multiple MCP servers simultaneously.

5.2 Tool Categories

Hanzo Chat ships with 260+ tools organized into categories:

Category	Tools	Example
Code execution	15	Python, Node.js, Bash
File operations	22	Read, write, search, glob
Web interaction	18	Fetch, search, browse
Database	12	SQL, vector, key-value
API integration	45	REST, GraphQL, gRPC
DevOps	28	Docker, K8s, CI/CD
Data analysis	20	Pandas, plotting, stats
Communication	15	Email, Slack, Discord
Version control	18	Git, GitHub, PRs
Cloud services	35	AWS, GCP, DO
AI/ML	22	Training, inference, eval
System	30	OS, process, network

Table 4: MCP tool categories and counts.

5.3 Tool Discovery and Selection

Not all 260+ tools are included in every prompt. Hanzo Chat uses a two-stage tool selection process:

Algorithm 5 Dynamic Tool Selection

Require: Query m_t, query embedding $\boldsymbol{e}_t$, all tools $\mathcal{T}$

▷ Stage 1: Embedding-based retrieval
$\mathcal{T}_{\text{cand}} \leftarrow \text{Top-20}(\cos(\boldsymbol{e}_t, \boldsymbol{e}_{\text{tool}}))$
▷ Stage 2: LLM-based filtering
$\mathcal{T}_{\text{sel}} \leftarrow \text{LLM.Filter}(m_t, \mathcal{T}_{\text{cand}})$
▷ Always include core tools
$\mathcal{T}_{\text{final}} \leftarrow \mathcal{T}_{\text{sel}} \cup \mathcal{T}_{\text{core}}$
return $\mathcal{T}_{\text{final}}$ ▷ Typically 8–15 tools

This reduces the tool definition overhead from ∼50K tokens (all tools) to ∼4K tokens (selected subset) while maintaining 97% recall on tool-use benchmarks.

5.4 Tool Execution Sandboxing

All tool executions run in isolated environments:

- **Code execution**: Docker containers with resource limits (CPU: 2 cores, RAM: 4GB, time: 60s, no network unless explicitly granted).
- **File operations**: Scoped to project directories with read/write permissions managed per user.
- **Web requests**: Rate-limited (10 req/min), filtered for malicious URLs, response size capped at 1MB.
- **API calls**: OAuth credentials stored in encrypted vault, never exposed to the LLM.

5.5 Multi-Step Tool Chains

Hanzo Chat supports iterative tool use where the model can invoke multiple tools in sequence, using the output of one as input to the next. The execution loop (Algorithm 1, lines 7–10) continues until the model produces a final text response without tool calls.

We observe that complex tasks often require 3–7 tool calls. To prevent infinite loops, we impose a configurable maximum (default: 25 tool calls per turn).

Task Type	Avg. Tool Calls	Success Rate
Simple file read	1.2	99.1%
Code debugging	3.8	87.4%
Data analysis	5.1	82.3%
Multi-file refactor	7.3	78.9%
Full-stack feature	12.4	71.2%

Table 5: Tool call statistics by task complexity.

6 Evaluation

6.1 Benchmarks

We evaluate Hanzo Chat on four benchmarks:

1. **MT-Bench** [25]: 80 multi-turn questions across 8 categories, judged by GPT-4.
2. **ChatBot Arena** [4]: Pairwise human preference rankings via crowdsourcing.
3. **MTTU-100**: Our novel Multi-Turn Tool-Use benchmark with 100 tasks requiring 2–15 tool calls across categories.
4. **MemBench**: Our memory benchmark testing recall accuracy over conversations of 10, 50, and 200 turns.

6.2 MT-Bench Results

System	Turn 1	Turn 2	Avg.
GPT-4 (single)	8.96	9.03	8.99
Claude 3.5 (single)	8.81	8.97	8.89
Gemini Ultra (single)	8.72	8.45	8.58
Hanzo Chat (routed)	**9.12**	**9.18**	**9.15**
Hanzo Chat (w/ memory)	9.08	**9.24**	9.16

Table 6: MT-Bench scores. Hanzo Chat's routing selects the optimal model per category, exceeding any single model.

The improvement on Turn 2 with memory (+0.06) demonstrates that retrieved context from Turn 1 improves second-turn responses.

6.3 MTTU-100 Benchmark

We designed MTTU-100 to evaluate multi-turn tool use in realistic scenarios. Each task specifies a goal, available tools, and expected outputs. Scoring is automated via deterministic verification of outputs.

System	Success	Tool Acc.	Turns
GPT-4 + plugins	62%	78%	4.2
Claude + MCP	71%	84%	3.8
Gemini + extensions	58%	73%	5.1
Hanzo Chat	**83%**	**91%**	**3.4**

Table 7: MTTU-100 results. Hanzo Chat achieves higher success rates with fewer turns due to dynamic tool selection and model routing.

6.4 MemBench Results

MemBench tests the ability to recall information introduced earlier in a conversation. We measure precision and recall of factual retrieval at different conversation depths.

System	10 turns	50 turns	200 turns
GPT-4 (128K)	98.2%	87.4%	42.1%
Claude (200K)	98.7%	91.2%	58.3%
Hanzo (no mem.)	97.9%	85.1%	39.8%
Hanzo (w/ mem.)	**99.1%**	**96.8%**	**93.2%**

Table 8: MemBench factual recall accuracy by conversation length. Memory-augmented Hanzo Chat maintains high recall even at 200 turns.

The dramatic improvement at 200 turns (93.2% vs. 58.3% for the best single-model baseline) demonstrates the value of hierarchical memory with semantic retrieval.

6.5 Ablation Studies

Configuration	MT-Bench	MTTU	Cost
Full system	9.15	83%	$0.018
− routing (fixed model)	8.89	71%	$0.042
− memory	9.08	79%	$0.017
− tool selection	9.12	74%	$0.022
− cascading	9.15	83%	$0.025
− all (single model, no mem)	8.81	68%	$0.042

Table 9: Ablation study showing contribution of each component.

7 Production Deployment

7.1 Deployment Infrastructure

Hanzo Chat has been deployed in production since August 2024, serving users through `chat.hanzo.ai`. The infrastructure comprises:

- **Frontend**: Next.js application deployed on Hanzo Platform with CDN edge caching for static assets.
- **Backend**: Node.js API server with WebSocket support for streaming, running on Kubernetes (3 replicas, auto-scaling to 12).
- **Memory store**: PostgreSQL with pgvector extension, 500GB SSD, WAL replication.
- **LLM Gateway**: Hanzo LLM Gateway proxying to 12 providers with automatic failover.
- **Tool servers**: 8 MCP server processes handling tool execution in Docker containers.

7.2 Usage Statistics

Over 18 months of production operation (August 2024 – February 2026):

Metric	Value
Total conversations	523,847
Total messages	4,891,203
Avg. turns per conversation	9.3
Avg. tokens per message	847
Tool calls executed	1,247,891
Memory entries stored	3,214,567
Unique users	42,318
P99 response latency	4.8s
Uptime	99.94%

Table 10: Production usage statistics (Aug 2024 – Feb 2026).

7.3 Model Selection Patterns

Analysis of production model routing reveals interesting patterns:

Model	Selection %	Satisfaction	Cost/q
Claude 3.5 Sonnet	34.2%	8.9/10	$0.021
GPT-4o	22.1%	8.7/10	$0.028
Claude Opus 4	12.8%	9.2/10	$0.067
Gemini 2 Flash	11.4%	8.1/10	$0.004
Zen 30B	8.9%	8.4/10	$0.008
Other (10+ models)	10.6%	7.9/10	$0.012

Table 11: Model selection distribution in production.

7.4 Memory Utilization

Memory retrieval significantly improves response quality in production:

- 78% of conversations beyond 5 turns benefit from memory retrieval (at least one retrieved memory used in response).
- Average retrieval latency: 23ms (pgvector HNSW index).
- Memory hit rate: 62% (fraction of queries where at least one memory exceeds the relevance threshold).
- User-reported improvement: 31% of users cite "it remembers what I told it" as a key differentiator.

7.5 Tool Usage Analysis

The most frequently invoked tools in production:

Tool	Calls/day	Success %
Code execution (Python)	3,247	94.1%
Web search	2,891	97.3%
File read	2,456	99.8%
File write	1,823	98.9%
Git operations	1,234	96.2%
API calls	987	91.4%
Database queries	654	93.7%
Image generation	432	88.9%

Table 12: Top tool usage in production (daily averages).

8 Related Work

8.1 Multi-Model Systems

Mixture-of-experts architectures [22, 5] route within a single model. FrugalGPT [3] chains LLM calls for cost optimization. RouterBench [9] evaluates model routing strategies. Martian [11] provides model routing as a service. Our approach differs by combining routing with memory and tool integration in a unified system.

8.2 Conversational Memory

MemoryBank [26] extends LLMs with long-term memory inspired by Ebbinghaus forgetting curves. Reflexion [23] uses self-reflection for task improvement. MemGPT [16] implements virtual context management with paging. Our hierarchical memory architecture extends these ideas with three-tier scoping and automatic consolidation.

8.3 Tool Use in LLMs

Toolformer [21] trains models to use tools via self-supervised learning. Gorilla [17] fine-tunes for API call generation. TaskWeaver [19] provides a code-first agent framework. The Model Context Protocol [2] standardizes tool interfaces. Hanzo Chat builds on MCP but adds dynamic tool discovery, sandboxed execution, and multi-step chaining.

8.4 Chat Interfaces

ChatGPT [14] popularized conversational AI interfaces. Claude.ai [1] introduced artifacts for rich content. Gemini [7] integrates multimodal understanding. Poe [20] provides multi-model access. Hanzo Chat differentiates through unified memory, tool integration, and intelligent routing across all models.

9 Discussion

9.1 Ethical Considerations

Hanzo Chat's memory system raises important privacy considerations:

1. **Data retention**: Users can view, edit, and delete all stored memories. Automatic deletion after 365 days of inactivity.
2. **Data separation**: Memories are strictly scoped by user and project. No cross-user information leakage.
3. **Content filtering**: All inputs and outputs are screened for harmful content before processing.
4. **Transparency**: Users are informed when memory retrieval augments a response (via metadata in the response).

9.2 Limitations

1. **Routing latency**: The classification and routing step adds 50–100ms to response latency. Acceptable for interactive use but potentially problematic for programmatic API access.
2. **Memory noise**: Retrieved memories are occasionally irrelevant, degrading response quality by 2–5% compared to oracle memory selection.
3. **Provider dependency**: Hanzo Chat depends on external LLM providers; outages at major providers can degrade service despite failover mechanisms.
4. **Tool safety**: Despite sandboxing, code execution tools carry inherent risk. We mitigate this through resource limits and output scanning.

9.3 Future Work

- **Personalized fine-tuning**: Fine-tune routing and memory models on individual user interaction patterns.
- **Multi-agent collaboration**: Enable multiple Chat agents to collaborate on complex tasks with shared memory.
- **Voice and multimodal**: Extend the interface to support voice input/output and image/video understanding.
- **Offline capable**: Support local model execution for privacy-sensitive use cases.

10 Conclusion

We have presented Hanzo Chat, a multi-model conversational AI system that combines intelligent

model routing, hierarchical persistent memory, and extensible tool integration through the Model Context Protocol. Our evaluation demonstrates that the combination of these three capabilities achieves performance exceeding any single model while significantly reducing costs. The hierarchical memory architecture enables conversations spanning hundreds of turns without degradation, addressing one of the most significant limitations of current LLM-based chat systems. Production deployment over 18 months with 500K+ conversations validates the practical viability of the approach. Hanzo Chat is available at `chat.hanzo.ai` and its architecture is documented for the research community.

References

[1] Anthropic. Claude 3.5 technical report. *Anthropic Technical Reports*, 2024.

[2] Anthropic. Model context protocol specification. *MCP Documentation*, 2024.

[3] L. Chen, M. Zaharia, and J. Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

[4] W.-L. Chiang, L. Zheng, Y. Sheng, et al. Chatbot arena: An open platform for evaluating LLMs by human preference. *arXiv preprint arXiv:2403.04132*, 2024.

[5] W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *JMLR*, 23(120):1–39, 2022.

[6] Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, 2016.

[7] Google. Gemini: A family of highly capable multimodal models. *Google Technical Reports*, 2024.

[8] Hanzo AI. Hanzo LLM gateway: Unified proxy for 100+ LLM providers. *Hanzo Technical Documentation*, 2025.

[9] X. Hu, Z. Li, and J. Chen. RouterBench: A benchmark for multi-LLM routing system. *arXiv preprint arXiv:2403.12031*, 2024.

[10] L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *WWW*, 2010.

[11] Martian. Intelligent model routing for LLM applications. *Martian Documentation*, 2024.

[12] J. L. McClelland, B. L. McNaughton, and R. C. O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex. *Psychological Review*, 102(3):419, 1995.

[13] R. Nogueira and K. Cho. Passage re-ranking with BERT. *arXiv preprint arXiv:1901.04085*, 2019.

[14] OpenAI. Introducing ChatGPT. *OpenAI Blog*, 2022.

[15] OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[16] C. Packer, S. Wooders, K. Lin, V. Fang, S. G. Patil, I. Stoica, and J. E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

[17] S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez. Gorilla: Large language model connected with massive APIs. *arXiv preprint arXiv:2305.15334*, 2023.

[18] A. Katz. pgvector: Open-source vector similarity search for PostgreSQL. *GitHub Repository*, 2023.

[19] B. Qin, D. Liang, X. Ye, et al. TaskWeaver: A code-first agent framework. *arXiv preprint arXiv:2311.17541*, 2024.

[20] Quora. Poe: Fast, helpful AI chat. *Quora Blog*, 2023.

[21] T. Schick, J. Dwivedi-Yu, R. Dessí, et al. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*, 2023.

[22] N. Shazeer, A. Mirhoseini, K. Maziarz, et al. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR*, 2017.

[23] N. Shinn, F. Cassano, A. Gopinath, K. R. Narasimhan, and S. Yao. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

[24] L. Wang, N. Yang, X. Huang, et al. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.

[25] L. Zheng, W.-L. Chiang, Y. Sheng, et al. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *NeurIPS*, 2023.

[26] W. Zhong, L. Guo, Q. Gao, et al. MemoryBank: Enhancing large language models with long-term memory. In *AAAI*, 2024.