GLM-4.6 Tool Calling & MCP Use: A Technical Analysis

By Cirra Published October 18, 2025 40 min read

Executive Summary

GLM-4.6 is Zhipu AI's latest flagship mixture-of-experts (MoE) language model, explicitly designed for **agentic tasks** and **tool usage**. It features a vast 200K-token context window, reasoning-capable "thinking mode," and native support for structured function/tool calls. According to Zhipu's own benchmarks and independent analyses, GLM-4.6 significantly outperforms its predecessor (GLM-4.5) on agent and coding tasks, demonstrating *"improved agent/tool use"* and achieving near-parity with leading models like Anthropic's Claude Sonnet 4 in multi-turn coding (48.6% win-rate) (Source: www.cometapi.com) (Source: glm46.org). Notably, GLM-4.6 is explicitly curated to *"support tool use during inference"*, autonomously deciding when to invoke external tools such as web search, calculators, or code execution in its "thinking mode" (Source: glm46.org) (Source: blog.oproai.com).

In terms of *tool calling*, GLM-4.6 builds on GLM-4.5's foundation of native function-calling (which already boasted a 90.6% success rate in benchmarks (Source: openlm.ai). The new model promises even greater reliability: its architecture and fine-tuning emphasize **chain-of-thought planning** and *"boring magic"* reliability in function calls (double-checking arguments and rejecting unknown tools) (Source: sider.ai) (Source: glm46.org). Third-party reports confirm that GLM-4.6 *"performs significantly better in agents that employ tools and search functions"* (Source: medium.com), and independent developers find its code-generation and multi-step workflows *"far exceeded expectations"* and closely matched top proprietary systems when ample contextual information is provided (Source: www.xmsumi.com).

Regarding **Model Context Protocol (MCP) use**, MCP is an open standard (introduced by Anthropic in late 2024 (Source: cloud.google.com) (Source: medium.com) that lets LLMs interact with external data/services in a consistent way, akin to a "universal interface" or "USB-C port" for AI (Source: medium.com) (Source: cloud.google.com). GLM-4.6 is not natively an MCP host (since MCP is a separate standard), but it can be integrated into MCP-based agent systems. For example, community-built MCP servers (e.g. CCGLM-MCP) already allow GLM-4.6 to serve as the AI engine behind Claude Code by routing queries through an MCP

tool to Zhipu's API (Source: lobehub.com) (Source: lobehub.com). In practice, any agent framework supporting MCP (such as NVIDIA's NeMo Agent or Claude's CLI) can be adapted to use GLM-4.6 as the LLM. GLM's JSON-formatted outputs and native function-calling make it well-suited for MCP-style workflows (Source: glm46.org) (Source: cloud.google.com).

In summary: GLM-4.6 has been architected and tuned explicitly for sophisticated tool usage and agentic applications. Empirical evaluations and expert testimonials indicate it handles tool calls and planning very effectively, often outperforming previous open models and closely rivaling the best proprietary systems in coding and agent tasks (Source: blog.oproai.com) (Source: www.cometapi.com). While MCP itself is an orthogonal protocol mainly championed by Anthropic, GLM-4.6 can plug into MCP ecosystems via existing tools. Overall, GLM-4.6 appears very capable at tool calling, and with appropriate interfaces it can leverage MCP-based context and services, making it a strong option for agentic AI development.

Introduction and Background

The advent of large language models (LLMs) has revolutionized artificial intelligence (AI), enabling **chatbots, coders, and AI assistants** to perform tasks once thought uniquely human. Early LLMs (e.g. GPT-2/3) could generate coherent text, but their "knowledge" was fixed at training time and they **could not interact with the outside world**. A major current trend is enabling LLMs to *augment* human workflows by **calling external tools and services** in a structured way. This addresses two key limitations: (1) LLMs' training data is static and may be outdated (Source: cloud.google.com), and (2) LLMs alone cannot execute code, fetch real-time data, or manipulate user environments. By giving the model a "universal connector" to external knowledge sources – akin to a USB-C port for AI (Source: medium.com) – we create *agentic* systems that can plan and act.

Two main frameworks have emerged for this purpose. One is **function/plugin calling** in platforms like GPT-4 and Claude (e.g. OpenAI's plugin system), where the model can output a JSON-formatted "function call" that a wrapper interprets to call an API (Source: cloud.google.com). The other is **Model Context Protocol (MCP)**, introduced by Anthropic in late 2024 (Source: cloud.google.com) (Source: medium.com). MCP is an open standard that specifies how an LLM can request structured information or actions from *any* external resource via a unified client-server protocol (Source: cloud.google.com). In essence, MCP and related approaches (e.g. Retrieval-Augmented Generation, AI agent frameworks) enable LLMs to request database queries, web searches, or function executions without manual prompt engineering. Instead of ad-hoc API calls, MCP formalizes the interaction with a standardized interface.

GLM (*General Language Model*) is a family of open, multilingual LLMs developed by Zhipu AI (Z.ai) in China. The GLM-4 series is explicitly targeted at reasoning, coding, and intelligent agent applications (Source: blog.oproai.com) (Source: stable-learn.com).GLM-4.5, released in mid-2025, was already notable for its "hybrid reasoning" design: it operated in either a *"thinking mode"* (for complex, multi-step reasoning and tool usage) or a *non-thinking mode* (for fast direct responses) (Source: blog.oproai.com). GLM-4.5 integrated reasoning, coding, and tool-calling capabilities, supporting structured `<tool_call>` outputs and chain-of-thought reasoning out of the box (Source: stable-learn.com), and achieved top-tier scores on benchmarks (third place overall on a 12-task suite) (Source: blog.oproai.com). Its open licensing (MIT) and published APIs made it readily accessible for developers, including compatibility with agent frameworks like Claude Code, Cline, and Roo Code (Source: stable-learn.com).

In September 2025, Zhipu released the next-generation GLM-4.6. According to Zhipu's **research announcement**, GLM-4.6 maintains ~355B parameters (32B active via its MoE architecture) but crucially expands the **context window** from 128K tokens (GLM-4.5) to 200K tokens (Source: blog.oproai.com) (Source: glm46.org). This allows the model to ingest far larger documents (e.g. whole codebases or documentation) at once. Zhipu also highlights *"superior coding performance"* and *"advanced reasoning"* in GLM-4.6 (Source: blog.oproai.com). Importantly, tool use is explicitly emphasized: GLM-4.6 *"supports tool use during inference"* and *"exhibits stronger performance in tool using and search-based agents"*, indicating focused improvements for execution of structured tasks (Source: blog.oproai.com) (Source: z.ai).

In parallel, the MCP ecosystem was evolving. With the MCP standard, tools come in two parts: **MCP servers** (lightweight wrappers around data sources or services) and an **MCP client/host** (the AI application/env). The LLM, through the MCP client, issues high-level data queries or action requests that the servers fulfill (Source: medium.com) (Source: cloud.google.com). MCP is architected to support any model, open or proprietary, as long as the model can produce standardized query requests. Thus, any LLM with appropriate tooling can plug into an MCP framework. The question is whether GLM-4.6 is particularly adept at this.

This report investigates GLM-4.6's performance and suitability for (1) calling external tools and (2) being utilized in MCP-style agentic systems. We will review the model's architecture, official and community evaluations of its tool-use capabilities, and practical integrations. We will compare GLM-4.6 to other leading models (e.g. GPT-4, Claude) on agentic metrics,

analyze case studies from developers, and discuss the implications for future AI systems. Throughout, we cite peer-reviewed, industry, and expert sources to substantiate claims. Our findings indicate that GLM-4.6 is highly competent at tool calling, and with open interfaces can serve effectively in MCP-based applications, while also noting remaining challenges and trajectories for LLM agent development.

GLM-4.6 Model Overview

To assess GLM-4.6's tool-calling abilities, it is essential to understand its design and how it differs from its predecessors and peers. The GLM-4.x models use a Mixture-of-Experts (MoE) transformer architecture, meaning that for each token only a subset of "expert" sub-networks are activated. This allows GLM-4.x to scale to very large total parameter counts (about 355–357 billion) while keeping per-token computation tractable (32 billion active parameters per token) (Source: openlm.ai) (Source: glm46.org). MoE also provides inherent advantages for routing different tasks or queries to different experts.

Architecture and Training

GLM-4.6 inherits the core "agentic, reasoning, and coding" (ARC) foundation of GLM-4.5 (Source: blog.oproai.com) (Source: stable-learn.com). According to Zhipu's technical descriptions, key architectural highlights include **Grouped-Query Attention with partial RoPE**, a large number of attention heads (96) for deep context understanding, and custom optimizers and training techniques (like a Muon optimizer and multi-token prediction) to boost reasoning capabilities (Source: blog.oproai.com). Importantly, GLM-4.5/4.6 were pretrained on an enormous mixed corpus: standard text plus heavy code and reasoning data. As one analysis notes, GLM-4.5 was first pretrained on 15 trillion tokens of general text, then on ~7–8 trillion tokens of code and QA data, with further fine-tuning for tool use and multi-turn tasks (Source: stable-learn.com) (Source: stable-learn.com). This extensive training composition underpins GLM's strong performance on both language and coding benchmarks.

After pretraining, GLM-4.x undergoes reinforcement learning (RL) fine-tuning to encourage coherent multi-step reasoning and tool execution. Zhipu open-sourced an RL framework called *"slime"* that implements a hybrid training pipeline with curriculum learning on task difficulty (Source: blog.oproai.com) (Source: stable-learn.com). The effect of this is that GLM-4.x learns to plan out tasks and invoke tools in a human-like reasoning loop. Indeed, the technical report for GLM-4.5 explicitly states that the model is trained and fine-tuned to support "tool invocation" and multi-turn execution (outputting structured tool-call tokens, etc.) (Source: stable-learn.com) (Source: stable-learn.com). These design choices make GLM inherently suited for agentic tasks.

Parameter Counts and Context Window

GLM-4.6 is a **MoE scale-up** from GLM-4.5, but the total parameter count remains roughly similar (~355–357B total, with 32B active). The **biggest change is in context length**. GLM-4.5's 128K-token window was already one of the largest contexts among LLMs, but GLM-4.6 further expands this to 200K tokens (Source: blog.oproai.com) (Source: glm46.org). In practice, this means GLM-4.6 can consider nearly *twice* as much text/code in a single prompt. Such a context extension is crucial for complex workflows like analyzing entire manuals or multi-file code projects without losing context.

The parameter increase from 355B to 357B is modest, suggesting that Zhipu focused on architectural and inference changes (like new routing/gating) rather than just brute force scaling (Source: openlm.ai) (Source: stable-learn.com). The mixture-of-experts layout allows this scale with manageable inference cost. Indeed, third-party reports note that GLM-4.6 achieves similar task outcomes with about **15% fewer output tokens** and thus lower API cost than GLM-4.5 (Source: www.cometapi.com), implying more efficient reasoning. In summary, GLM-4.6 doubles down on GLM-4.5's approach: very large context, agent-oriented learning, and cost-effective MoE compute.

Agentic Reasoning ("Thinking Mode")

A signature feature of GLM-4.x models is their *dual-mode* operation. GLM-4.5 introduced a "thinking mode" that explicitly engages in chain-of-thought reasoning and tool use, contrasted with a normal mode for quick answers (Source: blog.oproai.com). GLM-4.6 inherits and enhances this. The official documentation for GLM-4.6 states that **when thinking mode is enabled**, the model can

autonomously decide to invoke tools such as web search, calculators, or code execution, effectively **“turning the model into an agentic problem solver”** (Source: glm46.org). In other words, GLM-4.6 can internally plan multi-step actions: it reasons about the user’s request, chooses appropriate tools, passes arguments to them, and then integrates the results back into its solution.

Practically, this works by providing GLM-4.6 a special flag or instruction in the JSON API (for example, `"thinking": {"type": "enabled"}`), which tells it to output structured function calls when needed. It will output tokens describing the tool invocation (often encoded in JSON or a specialized format), then stream results back. The model’s output pipeline is designed for “JSON-native responses” and has built-in parsing logic, so tools can be tightly coupled (Source: glm46.org). This is similar to OpenAI’s function-calling interface, but GLM-4.6’s “thinking mode” is distinguished by its autonomous planning: the model isn’t told *when* to call a tool; it figures that out as part of its token generation.

The **improvement in reasoning and tool orchestration** is a key advertised highlight. According to Zhipu, GLM-4.6 shows *“a clear improvement in reasoning performance and supports tool use during inference”*, with *“stronger performance in tool using and search-based agents”* (Source: blog.oproai.com). By contrast, GLM-4.5 already supported tool calls but in GLM-4.6 the emphasis is on consistency and reliability of those calls. Internal documentation emphasizes minimizing “hallucinations” during tool calls and ensuring output JSON structures are well-formed (Source: sider.ai) (Source: sider.ai). One analysis notes that GLM-4.6 has better “boring magic” – i.e. it excels at getting the mundane parts of function-calling exactly right, such as correct JSON keys and types (Source: sider.ai). This suggests GLM-4.6’s fine-tuning heavily penalized errors in tool outputs, making it more robust in practice.

On a high level, GLM-4.6 treats tools as part of its reasoning. It has been described as being able to play the role of an “AI agent brain,” deciding *when* and *how* to invoke external services (Source: www.cometapi.com). Developers can embed GLM-4.6 into multi-agent frameworks like LangChain or LlamaIndex; the model’s output (with streaming tokens and JSON) can be parsed directly into downstream automation without heavy custom wrapper code (Source: glm46.org) (Source: www.cometapi.com). This design makes GLM-4.6 well-prepared for complex workflows that require dynamic calls to search, databases, or user-defined functions.

Tool Calling Capabilities of GLM-4.6

Given the above architecture, we now examine GLM-4.6 specifically on “tool calling” capabilities. This includes its native ability to produce structured function calls, the reliability of those calls, and how it performs in benchmarks and real applications that emphasize tool usage.

Native Function Calling and JSON Output

GLM-4.6 natively supports structured function/tool calls. When the API is used in an agentic mode, the model can output a JSON payload (or XML-like structure) indicating which tool to call and with what arguments. For example, it might output something like:

```
{"tool": "web_search", "query": "Latest Tesla stock price", "count": 1}
```

or a code execution request. The GLM-4.x crafting pipeline is aware of these tool “[tool_call]” tokens, and the output parser is trained to respect JSON schemas instead of free text.

This mechanism was present in GLM-4.5, where the model could wrap function calls in `<tool_call> ... </tool_call>` tags (Source: stable-learn.com). In GLM-4.6, the interface has been refined: the documentation explicitly mentions “function calling” as a feature, with output being “JSON-native” and easily integrable into workflows (Source: glm46.org). This parallels how OpenAI GPT-4 exposes a `function_call` API parameter, but in GLM-4.6 the model internally learns when to use these calls.

A key metric is **tool-calling success rate**. Zhipu reports that GLM-4.5 achieved a 90.6% success rate on a web-browsing benchmark (BrowseComp), surpassing many competitors (Source: openlm.ai). Although analogous numbers for GLM-4.6 are not explicitly published, the improvements in training suggest it would be at least as reliable. Third-party analyses observe that GLM-4.6 has “improved agent/tool use” over GLM-4.5 (Source: www.cometapi.com), which implies higher success or more appropriate use of tools. Practitioners also note that GLM-4.6’s function calls are more correct: one analysis says it “will refuse unknown tools and minimize invented arguments,” aiming for zero hallucination in tool calls (Source: sider.ai). In practical terms, GLM-4.6 tightly adheres to the provided tool schema, reducing the need for the developer to clean up malformed output.

Benchmark Performance and Comparisons

To quantify GLM-4.6's abilities, we look at benchmark results that involve agentic/tool tasks and coding workloads:

- **Agentic Benchmarks**: In Zhipu's internal evaluations, GLM-4.6 was tested across eight public benchmarks covering agentic reasoning and coding (Source: blog.oproai.com). While raw numbers aren't given in the announcement, GLM-4.6 "shows clear gains over GLM-4.5" and "holds competitive advantages over leading models" (Source: blog.oproai.com). OpenLM's summary of GLM-4.5 indicates it placed 3rd overall on these combined benchmarks, so an improvement with GLM-4.6 could mean it is a top-tier performer.
- **Tool/Browser Tasks**: On web browsing and search integration tasks (which simulate tool use), GLM-4.5 already matched Claude Sonnet 4 and nearly reached GPT-4's performance (Source: openlm.ai). Given GLM-4.6's architecture, one expects similar or better ratings. Unfortunately, public head-to-head data is limited, but CometAI's summary confirms that GLM-4.6 achieves *"near parity"* with Claude Sonnet 4. (Source: www.cometapi.com).
- **Coding Benchmarks (CC-Bench)**: GLM-4.6 was evaluated in the "CC-Bench" multi-turn coding competition. The official report notes that GLM-4.6 improved over GLM-4.5 and reached a **48.6% win-rate** against Claude Sonnet 4 (on par across 74 tasks) (Source: z.ai). This is explicitly attributed to better tool usage (e.g. integrated reference to specs and testing) and indicates that GLM-4.6 often produced higher-quality code solutions. In fact, the win-rate table shows GLM-4.6 vs Sonnet 4 at near-even odds (Source: glm46.org).
- **Efficiency**: GLM-4.6 not only maintains performance but does so more efficiently. Across CC-Bench tasks, it completes tasks with *15% fewer tokens* than GLM-4.5 (Source: z.ai). This token efficiency likely comes from smarter planning (making fewer rounds of back-and-forth) and better integration of tools. From a developer perspective, this means faster results and lower API costs.

We summarize these comparisons in the table below. It highlights GLM-4.6's specs and how they relate to earlier models and competitors on agentic tasks:

MODEL	TOTAL PARAMS (ACTIVE)	CONTEXT WINDOW	TOOL USAGE SUPPORT	AGENTIC FEATURES	PERFORMANCE (SELECTED)
GLM-4.5	~355B (32B active)	128K tokens	Native function-calling (XML/JSON), 90.6% success on web search tasks (Source: openlm.ai)	Hybrid modes: 'thinking' (chain-of-thought + tool use) & direct (Source: blog.oproai.com)	3rd on 12-benchmark suite (Source: blog.oproai.com)
GLM-4.6	~355B (32B active)	200K tokens	Enhanced native calls: JSON output, automated tool invocation (Source: glm46.org)	Advanced chain-of-thought; autonomous tool orchestration (Source: glm46.org) (Source: blog.oproai.com)	48.6% coding win vs Claude Sonnet 4 (CC-Bench) (Source: z.ai)
Claude Sonnet 4	(proprietary)	~100K tokens (*est.*)	Supports tools via MCP/plugins (authorized by Anthropic)	Emphasizes safe multi-step reasoning	~49% CC-Bench win vs GLM-4.6 (Source: www.cometapi.com)
GPT-4 (OpenAI)	~ (unpublished)	8K (or 32K/128K ext.)	Plugin ecosystem (OpenAI functions, Web browsing)	Chain-of-thought guided by prompt; no built-in "thinking mode"	Benchmarks strong on broad NLP (not directly measured here)

Table 1: Comparison of GLM-4.6 and peer models on key architecture and agentic capabilities (sources cited). GPT-4 and Claude entries are approximate and for context; GLM data from Zhipu/independent benchmarks (Source: glm46.org) (Source: www.cometapi.com).

Above, GLM-4.6 stands out for its **expanded context**, **explicit tool orchestration**, and **open-source availability** (unlike Claude or GPT). Its benchmarks suggest it is at least competitive with top proprietary systems on agentic tasks. The improvements from GLM-4.5 to GLM-4.6 are captured in the following summary table:

FEATURE	GLM-4.5	GLM-4.6	NOTES/CITATIONS
Context Window	128K tokens	200K tokens (1.56× larger) (Source: blog.oproai.com)	Allows processing of longer dialogues/docs
Parameters	~355B total (32B active)	~355B total (32B active) (Mixture-of-Experts)	Similar scale; focus on efficiency/enhancements
Coding Performance	Strong (e.g. CC-Bench tie)	Superior coding: 48.6% win vs. Claude Sonnet 4 (Source: z.ai)	Improved with new benchmarks (front-end code synthesis)
Reasoning & Planning	Good multi-step reasoning	Improved multi-step reasoning; advanced chain-of-thought	Emphasis on “reinforced chain planning” in fine-tuning
Tool Invocation	Supported via `tool_call` outputs (Source: stable-learn.com)	Enhanced “thinking mode” that autonomously triggers tools (Source: glm46.org)	More robust JSON function calls, less hallucination (Source: sider.ai)
Agent Integration	Supported (compatible APIs) (Source: stable-learn.com)	More effective integration; official docs cite embedding in agent workflows (Source: glm46.org)	Smoother with frameworks like LangChain, vLLM, SGLang
Efficiency (Tokens)	Baseline	~15% fewer tokens to complete tasks (Source: z.ai)	Faster inference / lower cost per task
Empirical Ranking	Ranked 3rd on industry benchmarks (Source: blog.oproai.com)	Improved over 4.5; “clear gains” on 8 benchmarks (Source: blog.oproai.com)	Nearly tied with top closed models

Table 2: Key differences between GLM-4.5 and GLM-4.6 (sources from Zhipu and analyses).

These tables and metrics show that GLM-4.6 is indeed engineered for **robust tool usage**. The next sections examine *how* well it uses tools in practice, as opposed to its raw specifications.

Reliability and Hallucination Behavior

An important aspect of tool-calling is **reliability**: does the model call the tool correctly, or does it “hallucinate” missing arguments or produce invalid output? Zhipu and independent analysts have focused on this “boring magic” of error-free calls.

Zhipu materials note improvements aimed at reducing chain-of-thought failures in multi-tool scenarios (Source: sider.ai). For instance, one blog notes that GLM-4.6 **“expects fewer unnecessary errors when handling multiple tool calls and large contexts”** (Source: sider.ai). This means the model is trained to minimize mis-parsing or runaway loops when juggling many sub-queries.

Moreover, explicit features target hallucinations: Unknown tool rejection and exact arguments. One source highlights that GLM-4.6 will **halt fabrication of extra parameters**: e.g. if asked to call `get_flights(dest, date)`, GLM-4.6 will not invent an extra param like `cabin_class` unless explicitly requested (Source: sider.ai). In contrast, less constrained models might hallucinate extra fields. This behavior is a direct result of GLM's structured training: it actually learns to format outputs to the given schema reliably, and RL fine-tuning likely penalized deviations heavily.

Another example of increased robustness: the model "fails fast" on bad arguments. If a tool rejects an input, GLM-4.6 outputs an explicit error rather than silently correcting it (Source: sider.ai). This transparency means downstream code can catch and handle errors properly, instead of trusting an unsound correction.

An illustrative quote from the analysis is revealing: *"The boring magic is making sure a tool call is correct every time. GLM-4.6 is better at the boring stuff."* (Source: sider.ai). In other words, GLM-4.6 may not do more (it's not more "creative"), but it does the mundane job of calling tools exactly right far more consistently. This is crucial in production: what matters is reliable function execution rather than occasional cleverness.

In sum, GLM-4.6's design and emerging reports suggest that tool calls are not only *possible* but *trustworthy*. It has been explicitly tuned to minimize hallucinations in the context of tool invocation (Source: sider.ai) (Source: sider.ai). Developers working with GLM-4.6 consistently note that it "obeys more reliably" and that its function calls "are more structured and accurate" than previous models (Source: sider.ai) (Source: sider.ai). This reliability makes it well-suited to applications where a failed tool call could derail a workflow.

GLM-4.6 in Agentic and MCP Frameworks

Moving from model design to systems integration, we now examine how GLM-4.6 can be embedded in larger agent frameworks and how it can work with the Model Context Protocol (MCP).

Embedding in Agent Frameworks (e.g. LangChain, LlamaIndex)

Because GLM-4.6 produces streaming, structured outputs and supports function calling, it can slot into existing agentic frameworks nearly as easily as GPT-4 or Claude. Indeed, Zhipu documentation explicitly mentions that teams can *"embed GLM 4.6 into production workflows without custom scaffolding"* (Source: glm46.org). In practice, this means any orchestration layer (like LangChain or LlamaIndex) that supports a JSON function interface can use GLM-4.6 as the "llm" backend.

For example, one could configure a LangChain agent with GLM-4.6 behind it: the chain would pass prompts to GLM-4.6's API, receive completions (including JSON tool calls), and then invoke the identified tools programmatically. GLM's high tool-call success rate and structured output reduce the need for custom parsing. Similarly, LlamaIndex (for retrieval-augmented agents) can feed GLM-4.6 a retrieval result and ask it to answer with tool use embedded. The open reference implementation *SGLang* (Structured-GLM) already provides parsers tailored to GLM, further simplifying this process (Source: blog.oproai.com).

One concrete integration example: Novita AI provides a platform to use GLM-4.6 inside Anthropic's Claude Code IDE. By signing up for a GLM-4.6 API key through Novita, developers can seamlessly run GLM-4.6 as the model behind Claude Code (Source: blogs.novita.ai) (Source: blogs.novita.ai). This means GLM-4.6 can participate in Claude Code workflows (which include features like IDE integration, plan mode, etc.) without modifications. Novita's guide highlights that GLM-4.6 *"integrates more effectively within agent frameworks"*, specifically calling out external tool integration as a strength (Source: blogs.novita.ai).

Another example is the open-source **SGLang** project, which implements a reasoning and tool-call parser for GLM models. It provides the code to parse GLM-4.6's output and handle JSON schemas, making it easy to chain GLM with data processing libraries (Source: blog.oproai.com). The existence of these libraries underscores that GLM-4.6 is supported in the open ecosystem, akin to Hugging Face and vLLM backends mentioned by Zhipu (Source: blog.oproai.com).

Model Context Protocol (MCP) Integration

Anthropic's Model Context Protocol is rapidly being adopted as a standard way for LLMs to leverage external data. While MCP is model-agnostic, its specifications must be implemented by both client (agent) and server (tools). GLM-4.6 itself does not ship with a built-in MCP client, but it can act as the LLM *host* in an MCP architecture.

In practice, this means an MCP client layer (such as Claude's CLI or NVIDIA NeMo's agent toolkit) can feed GLM-4.6's API input instructions and handle its JSON outputs. For example, as one developer report notes, within the Claude Code CLI it is possible to "add" MCP tool suites (like `exa` or `context7`) and then call them via `@tool` syntax (Source: www.xmsumi.com). Since GLM-4.6's output can contain JSON instructions, the MCP client can interpret them as tool invocations to MCP servers.

A particularly instructive case is the *CCGLM MCP Server* (community project). This project provides an MCP-compliant web service that bridges Claude Code tools to GLM-4.6. It registers a tool (named `glm_route`) which, when invoked, takes the user prompt and proxies it to GLM-4.6 via Zhipu's API (Source: lobehub.com). In effect, the Claude CLI (MCP host) sends a tool call to the CCGLM server, which then calls GLM-4.6. The chain is: **Claude Code → CCGLM MCP server tool → GLM-4.6 API → output** (Source: lobehub.com) (Source: lobehub.com). This demonstrates that GLM-4.6 can be slotted into an MCP ecosystem: the user interacts through the Claude/MCP interface, but the underlying LLM is GLM-4.6. The server handles authentication and response conversion, effectively treating GLM as just another "expert" LLM at the end of the chain.

In a broader sense, MCP integration simply requires an MCP client that knows how to call GLM-4.6. Any LLM-agnostic MCP host (like NeMo Agent Toolkit) can register GLM-4.6 as an "LLM provider" by using its API endpoints. For example, NVIDIA's documentation shows that new LLM providers can be added via plugins (Source: docs.nvidia.com) (Source: docs.nvidia.com), and GLM-4.6 could be integrated this way. Once integrated, the MCP host treats GLM-4.6 just like any other LLM: it issues structured queries, and GLM-4.6 returns structured answers.

Crucially, GLM-4.6's native features align well with MCP's design. MCP expects the LLM to produce structured JSON queries to ask for data, and GLM-4.6 by default outputs JSON for tools (Source: glm46.org). Thus, no additional translation layer is needed for GLM-4.6's output; an MCP client can feed GLM-4.6 a prompt and parse the JSON it returns. In fact, Zhipu explicitly mentions the ability to produce **JSON-native responses and function calls** (Source: glm46.org), which essentially means it is ready to speak the language of APIs.

In summary, GLM-4.6 can readily operate within an MCP framework. While GLM-4.6 itself is separate from the MCP spec, it is engineered to be a "plug-in LLM" that supports double-ended integration. Existing MCP server implementations (like `exa`, `context7`) can be connected to GLM-4.6 by using GLM-4.6 as the underlying model for reasoning. Indeed, practical user reports already describe using GLM-4.6 within MCP tools: one user explains how they installed `exa` and `context7` via Claude's MCP CLI when using GLM-4.6, leading to markedly improved code generation when ample documentation context was given (Source: www.xmsumi.com). This indicates that GLM-4.6 not only can use MCP tools but benefits greatly from them.

Data Analysis and Evidence

We now present data, case studies, and expert analyses that illuminate GLM-4.6's tool/agent performance.

Benchmark and Performance Data

As discussed, quantitative benchmarks are limited but telling:

- **CC-Bench (Multi-turn coding)**: Zhipu's evaluation found that GLM-4.6 had a **48.6% win-rate** against Claude Sonnet 4 on 74 coding tasks (Source: glm46.org). (Since a 50% win would indicate parity, this is essentially even.) It also required ~15% fewer tokens than GLM-4.5. The same review notes GLM-4.6 "lifts multi-turn coding win rates to 48.6% vs Claude Sonnet 4 and trims token usage by ~15% vs GLM-4.5" (Source: glm46.org). These figures are based on human-evaluated code output in realistic scenarios.
- **BrowseComp (Web search tasks)**: OpenLM reported that GLM-4.5 scored 26.4% on BrowseComp, surpassing Claude 4 Opus (18.8%) and approaching GPT-4 mini-high (28.3%) (Source: openlm.ai). While GLM-4.6's exact BrowseComp score isn't given, we expect it improved further given the 200K window and tuning.
- **Aggregate benchmarks**: In GLM-4.5, across 12 benchmarks (agentic, reasoning, coding), it scored 63.2 (ranked 3rd) (Source: blog.oproai.com). Though GLM-4.6's aggregated score is not published, the qualitative claim of *"clear gains"* suggests it now outperforms many rivals.

- **Token Efficiency**: In the same CC-Bench suite, GLM-4.6 uses 15% fewer tokens per task compared to 4.5 (Source: z.ai), indicating not just faster completion but likely more concise intermediate reasoning. This is a proxy for higher efficiency in workflows.

While none of these metrics directly isolate "tool calling," they implicitly reflect it. Good scores on multi-turn agentic tasks mean the model is successfully incorporating external steps. For example, in coding tasks, GLM-4.6 likely used test-running or searching to fix bugs via chained calls. Its BrowseComp success suggests reliable web-browsing requests.

Additionally, anecdotal user data provides insight:

- **OpenAI Compatibility Benchmarks**: Some community-run comparisons (via SourceForge and similar sites) peer GLM-4.6 against other models. One source notes GLM-4.6's enhanced performance vs GLM-4.5 ("step change in context, fewer tokens, improved agent/tool use") and near-equal strength against Claude on coding (Source: www.cometapi.com).
- **ModelScope/Hugging Face Releases**: Official releases of GLM-4.6 (and GLM-4.6-Air) have been made on open model zoos. Users report that GLM-4.6 tokens execute faster per token than GLM-4.5 (due to fewer layers activated?). Though not formally measured, this hints at inference efficiency.

In summary, the data indicate that GLM-4.6's raw performance on agent tasks is very strong. The model performs comparably or better than top proprietary rivals on tasks where tool integration matters. The tables above encapsulate the available figures: GLM-4.6's key strengths are its context size, improved reasoning, and coding wins, which translate into high success on tool-assisted tasks.

Expert Analyses and Benchmarks

Multiple expert analyses of GLM-4.6 have emerged in late 2025, providing additional context:

- **OproAI Engineering Blog** (Source: blog.oproai.com): A research-oriented blog presented GLM-4.6's improvements and benchmark scores. It explicitly states that GLM-4.6 *"exhibits stronger performance in tool using and search-based agents, and integrates more effectively within agent frameworks."* It also notes GLM-4.5's 90.6% tool-calling success. This endorsement from a third-party AI engineering group confirms Zhipu's claims on reasoning and tool use.
- **AI Monks (Medium)** (Source: medium.com): A detailed write-up emphasizes GLM-4.6's *"advanced agentic capabilities"*, claiming *"it performs significantly better in agents that employ tools and search functions"*. This aligns with the model's design for inference-time tool use.
- **Sider.ai (Thai blog)** (Source: sider.ai): This analysis, although informal, highlights that GLM-4.6 focuses on "multi-turn reasoning, tool use, and expanded context." It literally says: *"Focus on multi-turn reasoning, tool use, and wider context: GLM-4.6 is the improved new version that fine-tunes the parts you only notice when building it."* (Source: sider.ai). The blog goes in-depth, concluding that GLM-4.6 is especially suitable for tasks requiring structured tool invocation and multi-step planning.
- **Stable-Learn (Tech Explorer) on GLM-4.5** (Source: stable-learn.com): While about GLM-4.5, this source shows how Zhipu intentionally built GLM-4.x as "agent-native" with structured tool calls. It provides background: GLM-4.5 was pretrained and fine-tuned to understand tasks, plan substeps, invoke tools, and process results (Source: stable-learn.com). Thus, GLM-4.6 inherits a strong base.

Key takeaways from these analyses:

1. **Consensus on Strength**: All sources agree GLM-4.6 is a significant leap for agentic tasks. It is repeatedly described as "flagship" merits, "cost-efficient alternative", and overtaking previous open models (Source: www.cometapi.com).
2. **Tool/Agent Focus**: The uniqueness of GLM-4.6 lies in agent/task synergy. Experts repeatedly mention "tool use", "search-based agents", "structured tool calls" as focus points (Source: glm46.org) (Source: stable-learn.com) (Source: www.cometapi.com). It's clear GLM-4.6's developers and reviewers consider its tool-calling an area of strength, rather than an afterthought.

3. **Quantitative Benchmarks**: Evaluations like CC-Bench and BrowseComp are cited showing GLM-4.6 on par with best models (Source: z.ai) (Source: www.cometapi.com). These serve as quantitative evidence that tool-related performance is top-tier.

4. **Developer Experiences**: Users experimenting with GLM-4.6 in coding environments report highly positive results. A Chinese developer wrote that GLM-4.6's *"code quality"* is indistinguishable from Claude 4 on tasks like generating Rust or web code (Source: www.xmsumi.com). Another emphasizes that with the right context via MCP tools, *"the effect is very good"* for code generation (Source: www.xmsumi.com). These real-world accounts suggest the theoretical tool abilities translate into practical coding success.

Case Study: Stock Price Agent (Frank Morales, 2025)

Frank Morales Aguilera's Medium post (Oct 2025) provides a concrete example of GLM-4.6 in an agentic scenario (Source: medium.com) (Source: medium.com). Although informal, it is illustrative:

- **Setup**: GLM-4.6 is used via an OpenAI-compatible client with a system prompt instructing it to act as an "novelist" for creative tasks — demonstrating general generation. More pertinently, for the agentic part, GLM-4.6 is given a description of a Python function `get_current_stock_price(ticker)` that returns stock prices in JSON.

- **Workflow**: The author prompts GLM-4.6: "What are the stock prices for Tesla (TSLA) and Apple (AAPL)?" GLM-4.6 autonomously:

  1. **Plans** that it needs to call the tool for each stock symbol.
  2. **Executes**: calls `get_current_stock_price("TSLA")` and `get_current_stock_price("AAPL")` (simulated in the notebook).
  3. **Reflects**: synthesizes the JSON outputs into a final answer format.

  The log prints: *"Agent: Thinking/Planning... Agent: Calling tool(s) as planned... Agent: Synthesizing final answer..."* followed by a final formatted report with the prices (Source: medium.com) (Source: medium.com).

- **Analysis**: This showcases GLM-4.6 performing a multi-step tool-enabled task end-to-end. It correctly decides to call the function, passes arguments to it, and formats the results. The author notes that this parallels human problem-solving (plan, act, reflect). Crucially, the "brain" of the agent was GLM-4.6 itself, deciding when and how to use the function. This example mirrors how GLM's "thinking mode" is described: the model "makes a plan and calls tools as needed" (Source: medium.com) (Source: medium.com).

While the stock tool was a mocked function, the pattern is real: in a production system, the function could be an actual API call (like a finance database). GLM-4.6's ability to integrate it and handle the JSON output demonstrates most of the MCP interaction flow (structured tool request and structured response). The example culminated in a final answer with the fetched data. One limitation noted is that the tool was simulated; the author recognized that GLM-4.6's success depends on external tool quality. He concludes that designing reliable tools is as important as the AI agent itself.

Overall, the case study provides hands-on evidence that GLM-4.6 effortlessly manages an agentic query involving a custom tool. It exemplifies the features described in documentation: dynamic planning, tool invocation, and result integration (Source: medium.com) (Source: medium.com).

Discussion: Analysis and Broader Implications

Combining the above evidence, we can draw several conclusions about GLM-4.6's capabilities and implications for agentic AI.

Strengths in Tool Calling and Agentic Use

GLM-4.6's formal design, benchmark performance, and user reports all point to it being **very competent at tool calling**. Its explicit support for function outputs, large context for handling intricate tool outputs or long documents, and fine-tuned reasoning align to effective tool use. Compared to GLM-4.5, it is described as having *"clear improvements"* in tool use and reasoning (Source: blog.oproai.com), and it indeed achieves strong empirical results in coding/simulated tool tasks (Source: z.ai) (Source: glm46.org). In practice, users echo this; the developer who tried GLM-4.6 with Claude Code said it *"far exceeded expectations"* on coding tasks

and worked best when given context via MCP tools (Source: www.xmsumi.com). Experts also note GLM-4.6's *"highly expressive for complex, action-oriented tasks"* and that it helps *"interact with external systems with much more accuracy"* (Source: medium.com).

Crucially, GLM-4.6 is openly available (weights can be downloaded, and its API is accessible via providers like Novita), unlike ChatGPT/GPT-4. This openness enables developers to customize its tool-chain deeply. Software stacks (like Hugging Face Transformers, vLLM, or Copper built tools) can integrate GLM-4.6 directly. For example, the GLM-4.6 SGLang parser is open-source (Source: blog.oproai.com), meaning one can run local inference and hook tools without third-party gatekeeping. This stands in contrast to closed APIs which allow less flexibility with internals. This open nature means the full potential of GLM-4.6's tool interface can be exploited by researchers and companies.

Integration with MCP and Agentic Architectures

On the *MCP* front, GLM-4.6 is ready and able to interface with such systems. While it does not inherently come with an "Anthropic-endorsed" MCP client, it is entirely capable of being the LLM behind one. The CCGLM MCP server example shows one path: GLM-4.6 can be wrapped behind an MCP tool and used via Claude Code. Similar integration could be done with any MCP host (e.g. LlamaIndex v0.7 supports MCP, NeMo Agents do, etc.). The only real effort is coding the glue – which many open-source projects (LobeHub, mopserver, etc.) are already solving. The GLM/Anthropic interplay does raise some interesting dynamics: since MCP is primarily an Anthropic spec, GLM's use in that context often goes through Anthropic's front-ends (Claude CLI). But as the LLM is abstract in MCP, it's agnostic whether you call Claude's default model or Zhipu's GLM on the back end.

There is one philosophical point: MCP strives to standardize LLM integration, but as long as the LLM outputs conform to the MCP schema language, any model can drive that architecture. GLM-4.6's emphasis on JSON outputs and API control means it fits nicely. The threshold will be developer adoption: organizations already committed to MCP (e.g. certain banks or tech firms using Claude-based agents) could swap in GLM-4.6 if it meets accuracy and compliance needs (and it's cheaper and open). The technical report from Zhipu suggests GLM-4.6 was open-sourced on Hugging Face (Source: blog.oproai.com), which means on-premise integration with MCP is straightforward for enterprises concerned about data governance.

Limitations and Considerations

No model is perfect, and GLM-4.6 has some caveats. First, being new and complex, it is larger and still more resource-intensive than many models (especially the step from 128K to 200K context). In practice, serving 200K contexts requires substantial GPU memory or clever chunking. Also, MoE models have sometimes shown unpredictable latency (since different experts fire). The real-world speed and scaling for GLM-4.6 in production agentic loops will need evaluation.

Second, like all LLMs, GLM-4.6 can still hallucinate factual information and can reflect biases from training. Its improvements in tool use reduce *tool hallucinations*, but it can still misunderstand prompts outside the strict context. Ensuring the prompt engineering is tight (and possibly adding user auditing) remains important.

Third, while GLM-4.6 is open, some of the most optimized usage (e.g. inference at 100 tokens/sec) may rely on proprietary runtimes or quantization packs (e.g. VLLM, x86 optimizations). Community support for GLM-4.6 is growing but not yet as mature as for GPT or LLaMA. That said, several third-party tooling projects (like LM Studio, Openrouter, and Hugging Face) already support it, and major Chinese cloud providers are adding APIs.

Finally, the **MCP** ecosystem is still young and evolving. Although GLM-4.6 can join, the broader adoption of MCP is uncertain; competing approaches (like Retrieval-Augmented Generation with closed retrieval APIs) are also viable. If MCP gains widespread traction, GLM-4.6 with an MCP wrapper will be well-positioned. If not, GLM-4.6 will still be useful via bilateral tool integrations. In either scenario, its agentic strengths remain valuable.

Case Studies & Real-World Examples

To ground this discussion, we summarize some concrete examples of GLM-4.6 being used effectively with tools and agents:

- **Programming Assistant**: Several Chinese developer accounts highlight GLM-4.6 as an effective code "copilot". In [57], a developer test-switching from Cursor (with Claude) to GLM-4.6 in Claude Code reports that GLM-4.6 was nearly as good as Sonnet 4. With proper context, *"the effect is very good"* for generating correct code functions (Source: www.xmsumi.com). They emphasize that GLM-4.6 excelled when given relevant documentation via MCP tools (e.g. exa, context7). This shows how tool use (document retrieval) combined with GLM's reasoning yields strong performance.
- **Engaged Writing**: GLM-4.6 is bilingual and Turing-complete in generation. In writing-driven tasks (roleplay, style alignment), Zhipu claims it "aligns better with human preferences" in style (Source: blog.oproai.com). Researchers or content teams can use GLM-4.6 to draft narratives or summaries with the aid of knowledge retrieval (potentially via MCP). For instance, a user with GLM-4.6 could retrieve a historical text via MCP and have the model generate an article summarizing it. The Thai blog (Source: sider.ai) describes using GLM-4.6 with structured prompts to enforce plan->tools->execute cycles even in text tasks.
- **Creative Workflows**: The "fairy tale" segment in [33] also shows GLM-4.6 in a creative context. It produced an imaginative narrative when asked, demonstrating its generative ability as well as integration: it didn't call the stock tool in that section, but it shows GLM-4.6 can maintain character over multi-turn conversation. The blog frames GLM-4.6 as *"a smart and creative novelist"*, then uses the same model for agentic tool use in combination. This dual-use case suggests GLM-4.6 is flexible across programming/creative domains.
- **Enterprise Data Retrieval**: In principle, GLM-4.6 could be used in customer support or finance. For example, an enterprise could set up GLM-4.6 with an MCP server connected to their CRM and email system (like the Google example: "find latest report and email manager") (Source: cloud.google.com). While no public case study of GLM doing exactly that exists yet, its architecture implies it could: MCP standard dictating how to ask for a report from a DB, GLM planning the steps, calling the "database_query" and "email_sender" tools via output JSON, and confirming. Performance on such tasks would hinge on the tool correctness (MCP server implementation) more than the model; GLM-4.6's role would be understanding the high-level goal and integrating results.
- **Multimodal Agents**: While outside the scope of "tool calling", note that GLM-4.x also has multimodal versions (GLM-4.5-V). Future GLM-4.6 derivatives may include vision. In that case, "tool use" might extend to image classifiers etc., via MCP. For instance, a GLM-4.6 variant could get a context image or PDF and call a "vision" tool on it. Since this report is focused on tool calling, we mention it as future potential: a unified architecture could have GLM-4.6 controlling both text and vision tools.

Future Directions and Implications

Given GLM-4.6's capabilities, what does this imply for the field and for future models?

Standardization of Agentic Interfaces

GLM-4.6's integration with chain-of-thought and function calls exemplifies a broader trend: language models are evolving from static chatbots into **orchestrators of processes**. The strong support for tool APIs and MCP-standard interfaces suggests a future where many LLMs (open and closed) will converge on similar patterns: outputting JSON commands, streaming results, and abiding by schema checks. This is similar to how models rapidly adopted the OpenAI-style function-calling interface. We expect this trend to continue: GLM-4.6 already uses JSON natively; likely GLM-5 (if released) will push context even further or add more refined control (e.g. better uncertainly estimation, or sub-agent delegation).

Likewise, MCP or analogous protocols may become commonplace. If Anthropic's MCP garners widespread support, we may see more open models explicitly tested on MCP benchmarks. GLM-4.6 is well-suited for MCP, so if MCP faces a "killer app" moment, GLM-4.6 and its successors will be prime candidates to leverage it. We already see industry momentum: NVIDIA, Google, etc., talking about MCP in 2025 (Source: cloud.google.com) (Source: cloud.google.com). GLM-4.6 being open gives it an edge for organizations wanting to experiment with MCP outside Anthropic or OpenAI ecosystems.

GLM's Place in the Ecosystem

GLM-4.6's performance positions it as a leading open model. In many Chinese developer communities it is already considered on par with or better than international models for certain tasks (Source: www.xmsumi.com). The fact that it is open-source (MIT license) allows customization (fine-tuning, local hosting) that closed APIs cannot. We foresee more usage in China and other regions where data locality & cost are concerns. Zhipu's release also signals an intensifying global competition: it's one of the "Chinese Big Four" open models mentioned in tech press (Source: eu.36kr.com).

Future iterations (or rivals like Pangu-Ω, INTERNLM) will likely continue to focus on agentic capabilities. "Mixture-of-Experts" training, as used by GLM-4.x, may become more common if it yields domain-specific sub-networks (e.g. an "expert" for tool understanding). Similarly, we anticipate more hybrid training (text+code+planning) as the norm.

Limitations and Areas for Improvement

While GLM-4.6's tool use is strong, some weaknesses remain open for research. Error handling and safety in agentic modes still need caution. For instance, an LLM calling tools could potentially cause harmful actions if not properly sandboxed. Future work might involve formal verification of generated API calls, or multi-model oversight (e.g. using another model to audit GLM's tool calls).

Another area is **alignment**: as an "agentic" model, GLM-4.6 could be very persuasive. Ensuring it doesn't, say, invoke unauthorized services, requires careful meta-controls. Zhipu mentions "alignment fine-tuning" for persona consistency (Source: glm46.org), but alignment of actions (not just outputs) in tools could be a next frontier.

Finally, the economics: GLM-4.6 uses less tokens than GLM-4.5, but as model sizes grow, inference cost can still be high. Development of smaller distilled versions or early-exit mechanisms (stop thinking after a plan is made) could reduce latency. Also, as context windows extend, efficient memory handling (perhaps mixing retrieval and streaming context) will be key when the context length dwarfs memory.

Conclusion

In conclusion, the evidence strongly indicates that **GLM-4.6 excels at tool calling and works effectively within MCP-like agent frameworks**. Its architecture, training, and fine-tuning were explicitly guided toward agentic proficiency: the model inherently reasons about tasks in multi-step fashion and outputs structured tool invocations. Benchmark results and user experiences corroborate this, showing high success rates on web-search and code problems, often matching the best closed-source models (Source: z.ai) (Source: medium.com). Compared to GLM-4.5, the 4.6 version offers clear gains: a much larger context window (200K vs 128K (Source: blog.oproai.com), better reasoning and coding performance (Source: z.ai), and more refined tool calling.

On the MCP front, GLM-4.6 can be viewed as a fully capable participant. While MCP is a separate protocol championed by Anthropic, any LLM can use it if integrated properly. GLM-4.6's native JSON and function-call interface means it speaks a compatible language. Early integrations (like CCGLM-MCP) demonstrate feasibility (Source: lobehub.com). In practice, one could plug GLM-4.6 into any MCP-enabled system: for instance, making GLM-4.6 the engine behind an AI that answers enterprise queries by querying in-house databases via MCP servers. The model's strong comprehension and tool use ensure it could orchestrate such workflows competently.

As for limitations, GLM-4.6, like any LLM, is not flawless: it requires high-quality prompts and tool implementations, and large contexts can strain resources. It does mitigate many typical pitfalls by fully supporting the "thinking + acting" paradigm. The research and early adopters' experiences both suggest that when deployed thoughtfully, GLM-4.6 is an extraordinarily effective agentic model.

Given all this, the answer is clear: **Yes, GLM-4.6 is very good at tool calling.** Its native design for chain-of-thought and function outputs gives it a high level of competence in invoking external tools during inference. Additionally, GLM-4.6 is well-suited to MCP-based applications; while MCP itself is not part of GLM-4.6's code, GLM-4.6 can be integrated into any MCP architecture via standard

APIs. As such, GLM-4.6 can leverage the full power of MCP (connecting to databases, software, devices, etc.) as an agent. All claims here are supported by documented model specs and evaluations (Source: glm46.org) (Source: z.ai), and by independent analyses and case studies (Source: medium.com) (Source: www.xmsumi.com).

Future Implications: The success of GLM-4.6 in these regards signals a broader shift: open models are catching up to proprietary ones in agentic abilities. We expect to see even more integrated systems where models like GLM-4.6 drive complex workflows. Industry will likely embrace MCP and similar protocols, and GLM-4.6 users will be among the first to build versatile, cost-effective AI agents combining reasoning with tools.

Overall, for developers and organizations prioritizing powerful tool-calling and agent features in an open large model, GLM-4.6 is a top contender, validated by robust evidence and expert consensus.

References: (All claims above are supported by sources linked inline throughout the text, e.g. Zhipu AI's technical blog (Source: blog.oproai.com) (Source: z.ai), official model documentation (Source: glm46.org), model evaluations (Source: openlm.ai) (Source: www.cometapi.com), expert analyses (Source: sider.ai) (Source: medium.com), and developer case studies (Source: medium.com) (Source: www.xmsumi.com).)

Tags: glm-4.6, tool calling, model context protocol, mcp, agentic ai, function calling, zhipu ai, large language models, mixture of experts

---

About Cirra

About Cirra AI

Cirra AI is a specialist software company dedicated to reinventing Salesforce administration and delivery through autonomous, domain-specific AI agents. From its headquarters in the heart of Silicon Valley, the team has built the **Cirra Change Agent** platform—an intelligent copilot that plans, executes, and documents multi-step Salesforce configuration tasks from a single plain-language prompt. The product combines a large-language-model reasoning core with deep Salesforce-metadata intelligence, giving revenue-operations and consulting teams the ability to implement high-impact changes in minutes instead of days while maintaining full governance and audit trails.

Cirra AI's mission is to **"let humans focus on design and strategy while software handles the clicks."** To achieve that, the company develops a family of agentic services that slot into every phase of the change-management lifecycle:

- **Requirements capture & solution design** – a conversational assistant that translates business requirements into technically valid design blueprints.
- **Automated configuration & deployment** – the Change Agent executes the blueprint across sandboxes and production, generating test data and rollback plans along the way.
- **Continuous compliance & optimisation** – built-in scanners surface unused fields, mis-configured sharing models, and technical-debt hot-spots, with one-click remediation suggestions.
- **Partner enablement programme** – a lightweight SDK and revenue-share model that lets Salesforce SIs embed Cirra agents inside their own delivery toolchains.

This agent-driven approach addresses three chronic pain points in the Salesforce ecosystem: (1) the high cost of manual administration, (2) the backlog created by scarce expert capacity, and (3) the operational risk of unscripted, undocumented changes. Early adopter studies show time-on-task reductions of 70-90 percent for routine configuration work and a measurable drop in post-deployment defects.

Leadership

Cirra AI was co-founded in 2024 by **Jelle van Geuns**, a Dutch-born engineer, serial entrepreneur, and 10-year Salesforce-ecosystem veteran. Before Cirra, Jelle bootstrapped **Decisions on Demand**, an AppExchange ISV whose rules-based lead-routing engine is used by multiple Fortune 500 companies. Under his stewardship the firm reached seven-figure ARR without external funding, demonstrating a knack for pairing deep technical innovation with pragmatic go-to-market execution.

Jelle began his career at ILOG (later IBM), where he managed global solution-delivery teams and honed his expertise in enterprise optimisation and AI-driven decisioning. He holds an M.Sc. in Computer Science from Delft University of Technology and has lectured widely on low-code automation, AI safety, and DevOps for SaaS platforms. A frequent podcast guest and conference speaker, he is recognised for advocating “human-in-the-loop autonomy”—the principle that AI should accelerate experts, not replace them.

Why Cirra AI matters

- **Deep vertical focus** – Unlike horizontal GPT plug-ins, Cirra’s models are fine-tuned on billions of anonymised metadata relationships and declarative patterns unique to Salesforce. The result is context-aware guidance that respects org-specific constraints, naming conventions, and compliance rules out-of-the-box.
- **Enterprise-grade architecture** – The platform is built on a zero-trust design, with isolated execution sandboxes, encrypted transient memory, and SOC 2-compliant audit logging—a critical requirement for regulated industries adopting generative AI.
- **Partner-centric ecosystem** – Consulting firms leverage Cirra to scale senior architect expertise across junior delivery teams, unlocking new fixed-fee service lines without increasing headcount.
- **Road-map acceleration** – By eliminating up to 80 percent of clickwork, customers can redirect scarce admin capacity toward strategic initiatives such as Revenue Cloud migrations, CPQ refactors, or data-model rationalisation.

Future outlook

Cirra AI continues to expand its agent portfolio with domain packs for Industries Cloud, Flow Orchestration, and MuleSoft automation, while an open API (beta) will let ISVs invoke the same reasoning engine inside custom UX extensions. Strategic partnerships with leading SIs, tooling vendors, and academic AI-safety labs position the company to become the de-facto orchestration layer for safe, large-scale change management across the Salesforce universe. By combining rigorous engineering, relentlessly customer-centric design, and a clear ethical stance on AI governance, Cirra AI is charting a pragmatic path toward an autonomous yet accountable future for enterprise SaaS operations.