Neural Magic | vLLM

vLLM Office Hours #13 - November 7, 2024

Guest Topic: SOTA Tool-Calling Implementation in vLLM

Michael Goin

Engineering Lead at Neural Magic

vLLM Committer

Kyle Mistele

Founder of **Constellate AI**

A Few Housekeeping Items Before We Start

Let's make this an interactive session

- Turn your camera on 🙏
- Have something to say? Just start talking!

Ask questions using Zoom Chat

- Let's test it out – Open the chat and tell us where you are in the world.

This session is recorded

- Find it on our "vLLM Office Hours" page, YouTube, and X.
- Ask follow-up questions in vLLM Discord and Developer Slack.

What's New [in the Past Two Weeks]

vLLM Project Update

- vLLM v0.6.3.post1
- Model support: Ministral, VLM2Vec
- Optimizations for ngram spec decode

Today's Guest Topic

- ***POSTPONED*** SOTA Tool-Calling Implementation in vLLM (Kyle Mistele, Head of Product and Engineering at Zelus Labs)

Upcoming Office Office Hours Sessions

- **[NOV 14]** **The Impact of Disaggregated Prefill and KV Cache Storage in vLLM** *(Kuntai Du, vLLM Committer and Ph.D. Student at the University of Chicago)*
- **[DEC 5]** **Deep Dive into Machete, a Mixed-Input GEMM Kernel Optimized for NVIDIA Hopper GPUs** *(Lucas Wilkinson, Principal Engineer HPC at Neural Magic)*
- **[DEC 19]** **vLLM Project Update: 2024 Retrospective and 2025 Roadmap** *(Michael Goin, vLLM Committer and Engineering Lead at Neural Magic)*

View Past Recordings and Register for Future vLLM Office Hours Here.

About vLLM

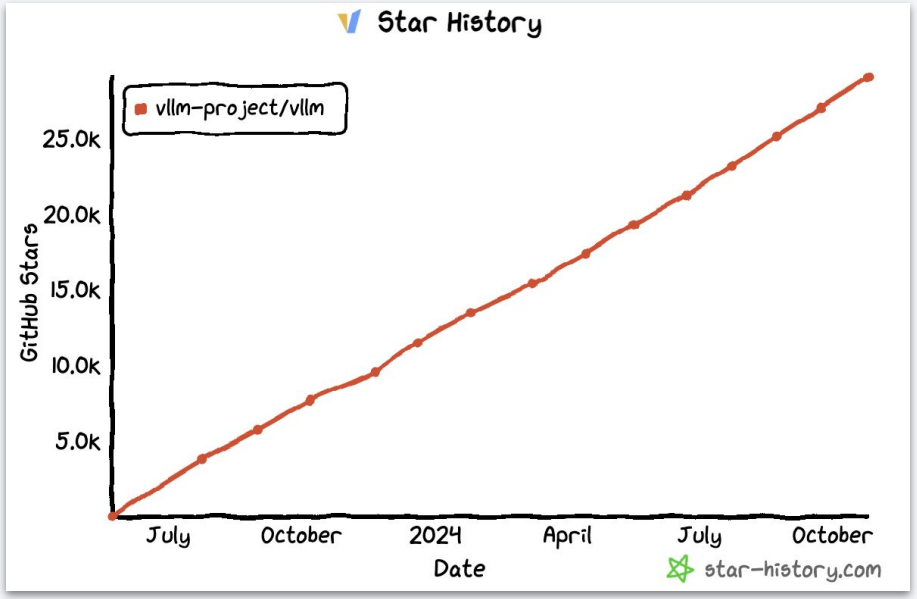
The de-facto standard in open source model serving

vLLM

- Fast and easy to use open source inference server

```
$ vllm serve neuralmagic/Meta-Llama-3.1-405B-Instruct-FP8 --tensor-parallel-size 8
```

- Support for all key model families, SOTA inference acceleration research, and diverse hardware backends like NVIDIA GPUs, AMD GPUs, Google TPUs, AWS Neuron, Intel Gaudi, and x86 CPUs.
- Full coverage of inference optimizations:
  - Quantization: GPTQ, AWQ, INT8, FP8, KV Cache
  - Chunked Prefill, Automatic Prefix Caching, Multi LoRA
  - Tensor Parallelism, Pipeline Parallelism



Llama Granite Gemma Qwen DeepSeek Mistral

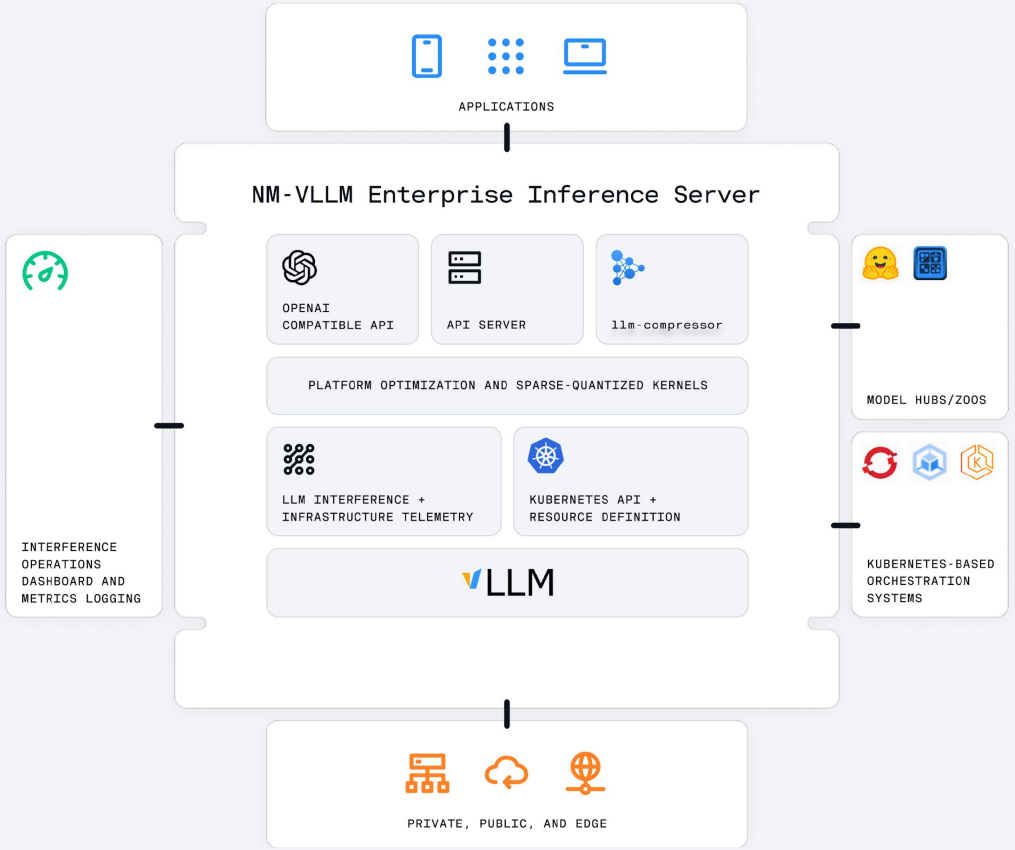
NVIDIA GPU AMD Instinct intel Gaudi Google TPU aws Neuron

About Neural Magic

Neural Magic's is a top open-source contributor and produces enterprise-ready distributions of vLLM

nm-vllm

- **Supported, stable distribution of vLLM**
- **Model optimization flows and best practices**
- **Registry of pre-optimized models**
- **Production telemetry and reference architectures**



We bring the software, people, and processes to make your vLLM deployment successful.

What's New in vLLM v0.6.3.post1

New Models

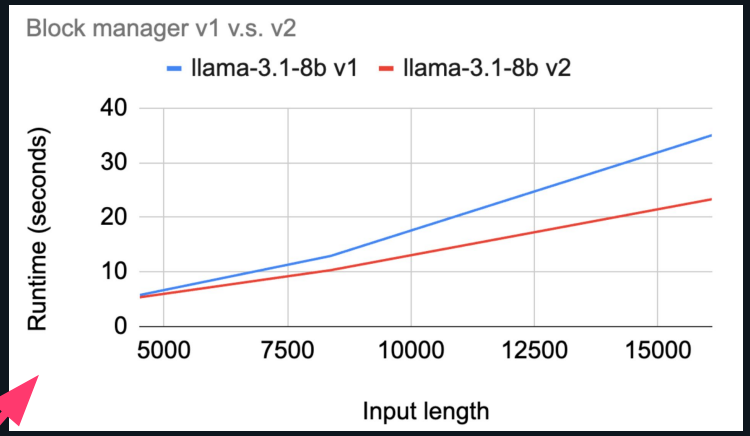
- Support Ministral 3B and Ministral 8B via interleaved attention (#9414)
- Support multiple and interleaved images for Llama3.2 (#9095)
- Support VLM2Vec, the first multimodal embedding model in vLLM (#9303)

Important bug fix

- Fix chat API continuous usage stats (#9357)
- Fix vLLM UsageInfo and logprobs None AssertionError with empty token_ids (#9034)
- Fix Molmo text-only input bug (#9397)
- Fix CUDA 11.8 Build (#9386)
- Fix `_version.py` not found issue (#9375)

Other Enhancements

- Remove block manager v1 and make block manager v2 default (#8704)
- Spec Decode Optimize ngram lookup performance (#9333)



Some performance numbers on a single H100:
input_len: 550, output_len: 150
I changed the prompt to try different system efficiency (which might include the number of CPU <-> GPU sync).

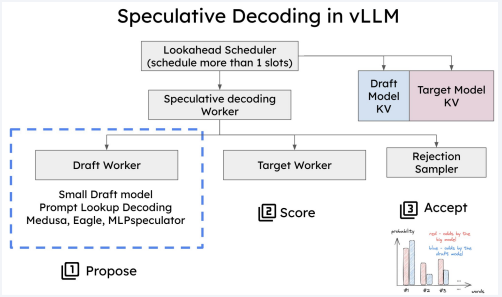
System efficiency	propose time before this PR	propose time after this PR	end2end latency before this PR	end2end latency after this PR
0.31	4.4ms	2.2ms	6.4s	5.6s
0.63	3.3ms	1.5ms	3.8s	3.2s
0.80	2.6 ms	1.5ms	3.0s	2.6s

~2x improvement!

Blogs Worth Checking Out

[Introducing Machete, a Mixed-Input GEMM Kernel Optimized for NVIDIA Hopper GPUs]

[We Ran Over Half a Million Evaluations on Quantized LLMs: Here's What We Found]



[How Speculative Decoding Boosts vLLM Performance by up to 2.8x]

Neural Magic | vLLM

vLLM Office Hours - November 7, 2024

SOTA Tool-Calling Implementation in vLLM

Kyle Mistele

Founder of **Constellate AI**

SOTA Tool-Calling in vLLM: A Brief Outline

Today we'll cover:

- What are tools & functions?
- Why tools matter for open-source AI
- How do models implement tools?
- How I implemented OpenAI-style tools in vLLM + challenges faced
  - Compatibility
  - Standardization
  - Streaming
- Putting it all together with vLLM

OpenAI-Style Tools & Functions

Functions

- Function API Introduced by OpenAI
- Allows developers to provide functions the AI model can “call”
- AI generates a JSON object indicating the function it wants to call + arguments
- Provide the result to the model; it can create a chat response or call more functions
- **Important because it allows AI to interact with external systems in a developer-defined manner!**

Tools

- A later abstraction introduced by OpenAI
- Supplements user-provided functions, with additional OpenAI provided “tools”
- Intended for abstraction → extensibility
- e.g. **Code interpreter, web search**
- Most models ONLY adopt functions (e.g. Mistral, Hermes, Qwen)
- Other models allow user-defined functions AND support built-in tools (Llama 3.1)

Why do tools matter?

- **Tools let you give LLMs new capabilities!**
  - Web search
  - API integrations (get_current_weather)
  - save & load “memories” for very long conversations or other use-cases
  - Vector, full-text, or SQL search (RAG)
  - Code interpreter: write & execute code!
  - Robotics applications…

Agents!

Why do tools matter?

Why do we want OpenAI-compatible tools in vLLM?

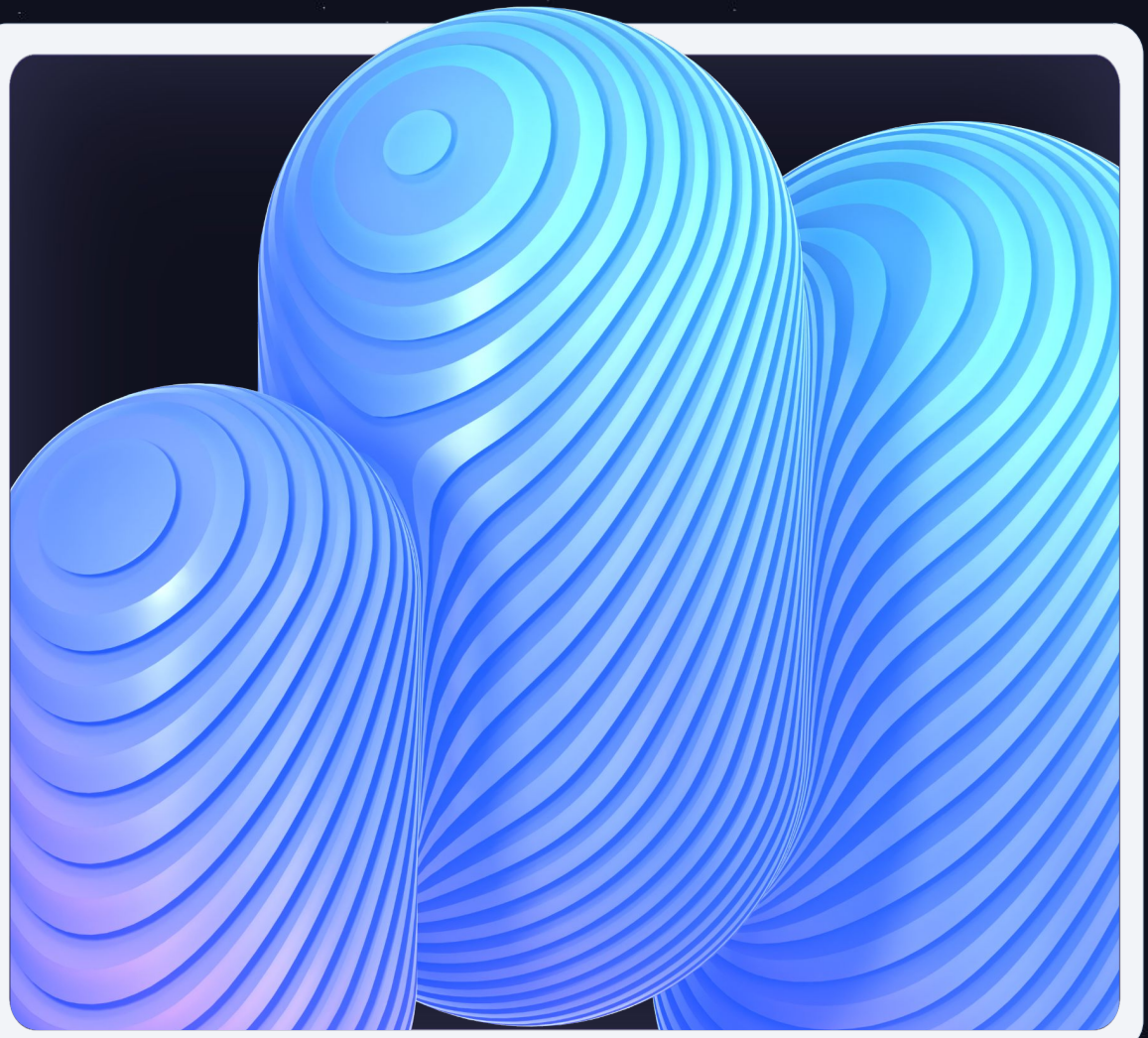
- Build agentic applications with the toolchain you're already using! (vLLM + OpenAI-compatible SDKs etc.)
- Enable tool streaming
  - user-facing applications
  - better UI/UX
- Increase OpenAI compatibility
- Become THE framework for building agents & applications on open-source LLMs

How do we implement tool calling?

1. **Tool-Compatible chat templates**

2. **Identify ideal configuration** (temperatures, tokenizers, etc.)

3. **OpenAI-compatible API implementation:**
    a. non-streaming tool extraction (easy)
    b. streaming tool extraction (very, very hard)



Challenges

- **Lack of standardization.** Different models have their own standard for how they indicate tools calls in their output; different chat templates
- **Varying quality**. Some models are just *better* than others with respect to tool calling
- **“Gotchas”**. model-specific quirks; custom prompting, etc.
- **Streaming**. How do we translate the model’s generated tokens into OpenAI’s streaming format *on the fly?*

vLLM Open Office Hours - November 7, 2024

Open Model Tool Call Formats (Easy)

Nous Research's Hermes Format (& Qwen 2.5) JSON

```
<tool_call>
{"name": "get_weather", "arguments": { "location": "San Francisco" }}
</tool_call>
```

Mistral AI Function Calling Format JSON

```
[TOOL_CALLS]{"name": "get_weather", "arguments": { "location": "San Francisco"
}}]
```

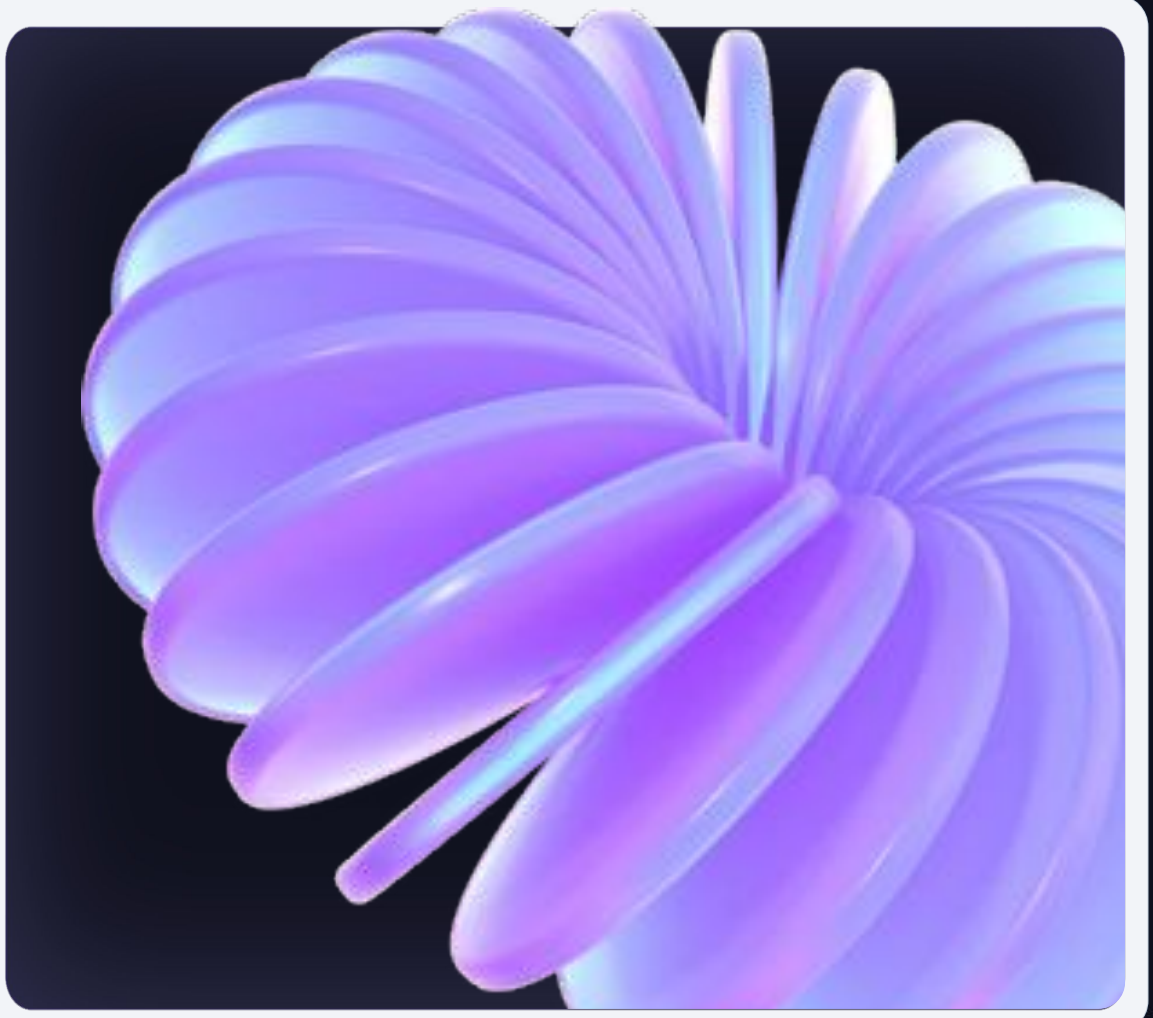
Tool Extraction & Parsing: Non-Streaming

Architectural Solution for Tool Extraction:

1. Create a **tool parser** module for each supported model, with separate methods to handle streaming extraction, and to handle non-streaming extraction

Non-Streaming Tool Extraction

1. Defines a regex to check for & extract tool call portion of response, if present
2. Parse extracted call to find name, arguments
3. Convert into OpenAI-compatible tool call
4. Return an OpenAI-compatible chat completion to the client



vLLM Open Office Hours - November 7, 2024

Tool Extraction & Parsing:

Non-Streaming Example

Nous Research's Hermes Format (& Qwen 2.5) JSON

```
<tool_call>
{"name": "get_weather", "arguments": { "location": "San Francisco" }}
</tool_call>
```

Chat Completion Response JSON

```
{
      "role": "assistant",
      "content": null,
      "tool_calls": [
            {
                  "id": "call_abc123def456" ,
                  "type": "function",
                  "function": {
                        "name": "get_weather",
                        "arguments": "{\"location\":\"San Francisco\"}"
                  }
            }
      ]
}
```

vLLM Open Office Hours - November 7, 2024

Tool Extraction & Parsing:

Streaming Example

Nous Research's Hermes Format (& Qwen 2.5) JSON

```
<tool_call>
{"name": "get_weather", "arguments": { "location": "San Francisco" }}
</tool_call>
```

Chat Completion Response JSON


```
data: {"id":"chatcmpl-AMPzSm67wEVeCINXnR1qAI9TpZulv","object":"chat.completion.chunk","created":1729906486,"model":"gpt-0613","system_fingerprint":null,"choices":[{"index":0,"delta":{"role":"assistant","content":null,"tool_calls":[{"index":0,"id":"call_5RxV1Ni27OBGTA20ZUHdZusK","type":"function","function":{"name":"get_current_weather","arguments":""}}],"refusal":null},"logprobs":null,"finish_reason":nu

data: {"id":"chatcmpl-AMPzSm67wEVeCINXnR1qAI9TpZulv","object":"chat.completion.chunk","created":1729906486,"model":"gpt-0613","system_fingerprint":null,"choices":[{"index":0,"delta":{"tool_calls":[{"index":0,"function"{"arguments":"{\n"}}]},"logprobs":null,"finish_reason":null}]}

data: {"id":"chatcmpl-AMPzSm67wEVeCINXnR1qAI9TpZulv","object":"chat.completion.chunk","created":1729906486,"model":"gpt-0613","system_fingerprint":null,"choices":[{"index":0,"delta":{"tool_calls":[{"index":0,"function"{"arguments":" "}}]},"logprobs":null,"finish_reason":null}]}

data: {"id":"chatcmpl-AMPzSm67wEVeCINXnR1qAI9TpZulv","object":"chat.completion.chunk","created":1729906486,"model":"gpt-0613","system_fingerprint":null,"choices":[{"index":0,"delta":{"tool_calls":[{"index":0,"function"{"arguments":" \""}}]},"logprobs":null,"finish_reason":null}]}

data: {"id":"chatcmpl-AMPzSm67wEVeCINXnR1qAI9TpZulv","object":"chat.completion.chunk","created":1729906486,"model":"gpt-0613","system_fingerprint":null,"choices":[{"index":0,"delta":{"tool_calls":[{"index":0,"function"{"arguments":"city"}}]},"logprobs":null,"finish_reason":null}]}

data: [DONE]
```

vLLM Open Office Hours - November 7, 2024

Tool Extraction & Parsing:

(Simplified)

Nous Research's Hermes Format (& Qwen 2.5) JSON

```
<tool_call>
{"name": "get_weather", "arguments": { "location": "San Francisco" }}
</tool_call>
```

Simplified Tool Call SSE JSON

```
{"role":"assistant","content":null, "tool_calls": [{
"id":"call_emnz7Hxbc2Sov0MRg65HfDTF","type":"function",
 "index":0,
 "function":{"name":"get_current_weather","arguments":""}}]}

{"tool_calls":[{"index":0,"function":{"arguments":"{\n"}}]}
{"tool_calls":[{"index":0,"function":{"arguments":" "}}]}
{"tool_calls":[{"index":0,"function":{"arguments":" \""}}]}
{"tool_calls":[{"index":0,"function":{"arguments":"city"}}]}
{"tool_calls":[{"index":0,"function":{"arguments":"\":"}}]}
{"tool_calls":[{"index":0,"function":{"arguments":" \""}}]}
{"tool_calls":[{"index":0,"function":{"arguments":"San"}}]}
{"tool_calls":[{"index":0,"function":{"arguments":" Fr"}}]}
{"tool_calls":[{"index":0,"function":{"arguments":"anc"}}]}
{"tool_calls":[{"index":0,"function":{"arguments":"isco"}}]}
{"tool_calls":[{"index":0,"function":{"arguments":"\"\n"}}]}
{"tool_calls":[{"index":0,"function":{"arguments":"}"}}]}
```

Tool Extraction & Parsing: Streaming

How do we support tools + streaming?
On the fly, we have to:

1. Detect when a tool call is being generated by the model

2. Use partial JSON parsing to extract the tool name as soon as the name is finished, and stream that to the client

3. As the arguments are being generated, use partial JSON parsing to extract diffs and stream them to client.

It's **stateful** and **recursive**

vLLM Open Office Hours - November 7, 2024

Partial JSON Parsing Examples

1 of 2

Partial JSON Parsing Examples — JSON

```
// with "incomplete fields" disabled
{"nam                          -> {}
{"name": "Ky                   -> {}
{"name": "Kyle", "s"           -> {"name": "Kyle"}
{"name": "Kyle", "s": tr       -> {"name": "Kyle"}
{"name": "Kyle", "s": true     -> {"name": "Kyle", "s": true}


// with "incomplete fields" enabled
{"nam                                -> {}
{"name": "Ky                         -> {"name": "Ky"}
{"name": "Kyle"                      -> {"name": "Kyle"}
{"name": "Kyle", "s"                 -> {"name": "Kyle"}
{"name": "Kyle", "s": tr             -> {"name": "Kyle", "s": true}
{"name": "Kyle", "s": true, "a": 1   -> {"name": "Kyle", "s": true, "a": 1}
{"name": "Kyle", "s": true, "a": 100} -> {"name": "Kyle", "s": true, "a": 100}
```

vLLM Open Office Hours - November 7, 2024

Partial JSON Parsing Examples

2 of 2

Partial JSON Parsing Examples — JSON

```
<tool_call>
<tool_call>{
<tool_call>{"nam
<tool_call>{"name":
<tool_call>{"name": "get_
<tool_call>{"name": "get_weather"
<tool_call>{"name": "get_weather", "argu
<tool_call>{"name": "get_weather", "arguments
<tool_call>{"name": "get_weather", "arguments": {
<tool_call>{"name": "get_weather", "arguments": { "loca
<tool_call>{"name": "get_weather", "arguments": { "location
<tool_call>{"name": "get_weather", "arguments": { "location": "
<tool_call>{"name": "get_weather": "arguments": { "location": "San
<tool_call>{"name": "get_weather": "arguments": { "location": "San Fran
<tool_call>{"name": "get_weather": "arguments": { "location": "San Francisco
<tool_call>{"name": "get_weather": "arguments": { "location": "San Franscisco" }
<tool_call>{"name": "get_weather": "arguments": { "location": "San Franscisco" }}
<tool_call>{"name": "get_weather": "arguments": { "location": "San Franscisco" }}</tool_call>
```

Streaming Function Call Extraction Example

A	B	C	D	E
Generated tokens	**parsed JSON object**	**generated token(s)**	**streamed token(s)**	**stream field**
<tool_call>				
<tool_call>{	{}	{		
<tool_call>{"nam	{}	"nam		
<tool_call>{"name":	{}	e":		
<tool_call>{"name": "get_	{}	"get_		
<tool_call>{"name": "get_weather"	{"name": "get_weather"}	weather"	"get_weather"	name
<tool_call>{"name": "get_weather", "argu	{"name": "get_weather"}	, "argu		arguments
<tool_call>{"name": "get_weather", "arguments	{"name": "get_weather"}	ments		arguments
<tool_call>{"name": "get_weather", "arguments": {	{"name": "get_weather", "arguments": {}}	": {	"{"	arguments
<tool_call>{"name": "get_weather", "arguments": { "loca	{"name": "get_weather", "arguments": {}}	"loca		arguments
<tool_call>{"name": "get_weather", "arguments": { "location	{"name": "get_weather", "arguments": {}}	tion		arguments
<tool_call>{"name": "get_weather", "arguments": { "location": "	{"name": "get_weather", "arguments": {"location": ""}}	":"	"\"location\":	arguments
<tool_call>{"name": "get_weather": "arguments": { "location": "San	{"name": "get_weather", "arguments": {"location": "San"}}	San	"\"San "	arguments
<tool_call>{"name": "get_weather": "arguments": { "location": "San Fran	{"name": "get_weather", "arguments": {"location": "San Fran"}}	Fran	" Fran"	arguments
<tool_call>{"name": "get_weather": "arguments": { "location": "San Francisco	{"name": "get_weather", "arguments": {"location": "San Francisco"}}	cisco	"cisco"	arguments
<tool_call>{"name": "get_weather": "arguments": { "location": "San Franscisco" }	{"name": "get_weather", "arguments": {"location": "San Francisco"}}	" }	"\"}"	arguments
<tool_call>{"name": "get_weather": "arguments": { "location": "San Franscisco" }}	{"name": "get_weather", "arguments": {"location": "San Francisco"}}	}	"}"	arguments
<tool_call>{"name": "get_weather": "arguments": { "location": "San Franscisco" }}</tool_call>	{"name": "get_weather", "arguments": {"location": "San Francisco"}}	</tool_call>		arguments

Diffing From the Start of the string

model generation	partially parsed JSON		
<tool_call>{"name": "get_weather": "arguments": { "location": "San	"{\"location\": \"San\"}}"		
<tool_call>{"name": "get_weather": "arguments": { "location": "San Fran	"{\"location\": \"San Fran\"}}"	starting from the front: difference is cisco\"}}	
<tool_call>{"name": "get_weather": "arguments": { "location": "San Francisco	"{\"location\": \"San Francisco\"}}"		

So what do we do about the ‘\“}}’ ?

Diffing From the End of the string

model generation	partially parsed JSON		
<tool_call>{"name": "get_weather": "arguments": { "location": "San	"{\"location\": \"San\"}}"		
<tool_call>{"name": "get_weather": "arguments": { "location": "San Fran	"{\"location\": \"San Fran\"}}"	starting from the end: difference starts after \"}}	
<tool_call>{"name": "get_weather": "arguments": { "location": "San Francisco	"{\"location\": \"San Francisco\"}}"		
		conclusion: send the diff "cisco"	

It gets worse.

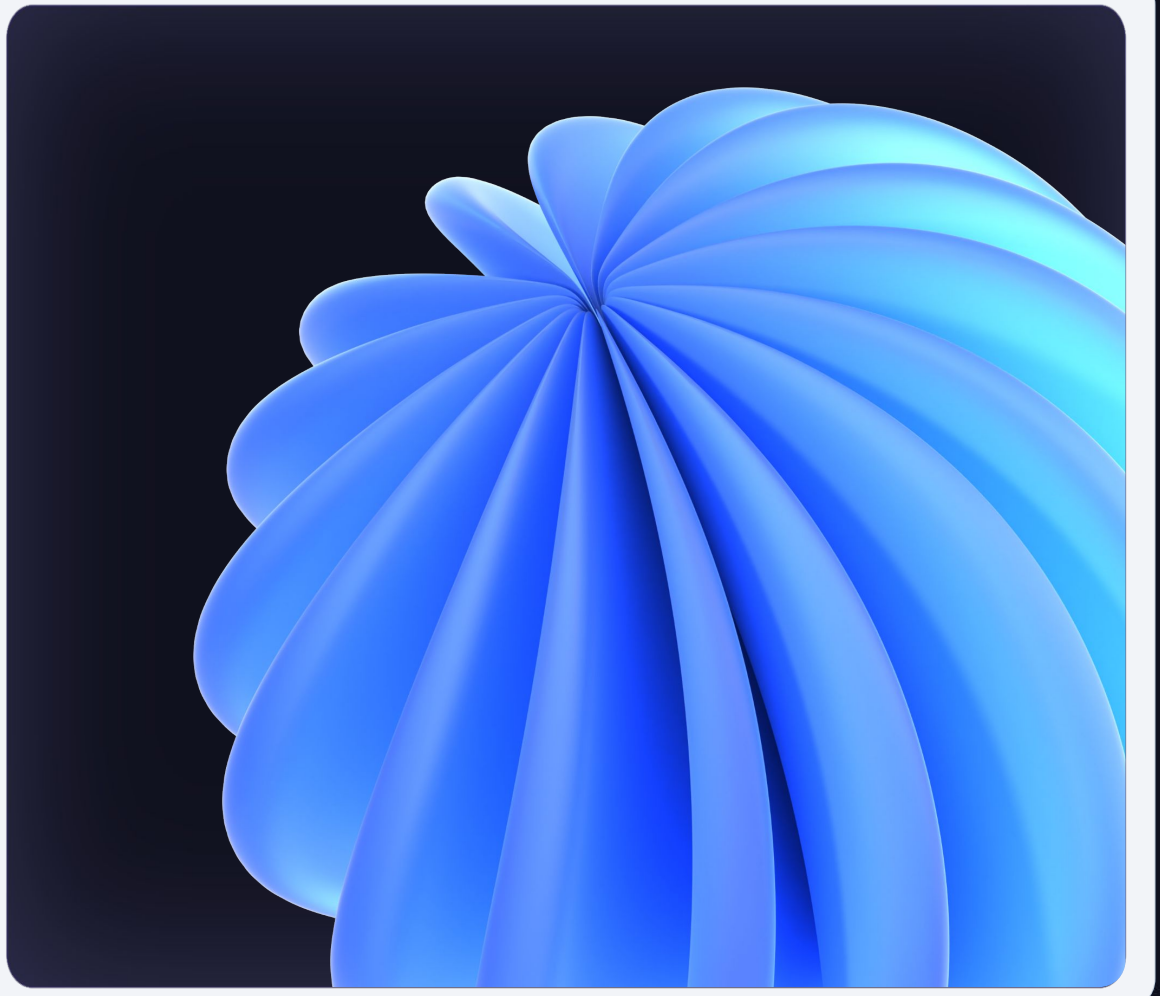
model generation	partially parsed JSON			
<tool_call>{"name": "get_weather": "arguments": { "location": "San	"{\"location\": \"San\"}}"			
<tool_call>{"name": "get_weather": "arguments": { "location": "San Fran	"{\"location\": \"San Fran\"}}"	starting from the front: diff is " Fran\"}}"		
<tool_call>{"name": "get_weather": "arguments": { "location": "San	"{\"location\": \"San\"}}"			
<tool_call>{"name": "get_weather": "arguments": { "location": "San Fran	"{\"location\": \"San Fran\"}}"			
		starting from the end: difference starts after "an\"}}"		
		diff of the diffs: " Fr" when the model actually generated " Fran"		

partially parsed JSON				
"{\"location\": \"San\"}}"				
"{\"location\": \"San Fran\"}}"	starting from the front: diff is " Fran\"}}"			
-> remove "{\"location\": \"San				
then come at it from backwards:				
\"}}"				
Fran\"}}"	diff from the end, having removed the matching part from the forward pass:			
	we get the correct diff, "Fran"			

Additional Tool Streaming Problems

But wait, there's more!

- How do you handle sequential tool calls?
- State management?
- How do you handle fault tolerance?
  - Open-source models, especially small ones, are error-prone!
- Error handling and recovery?

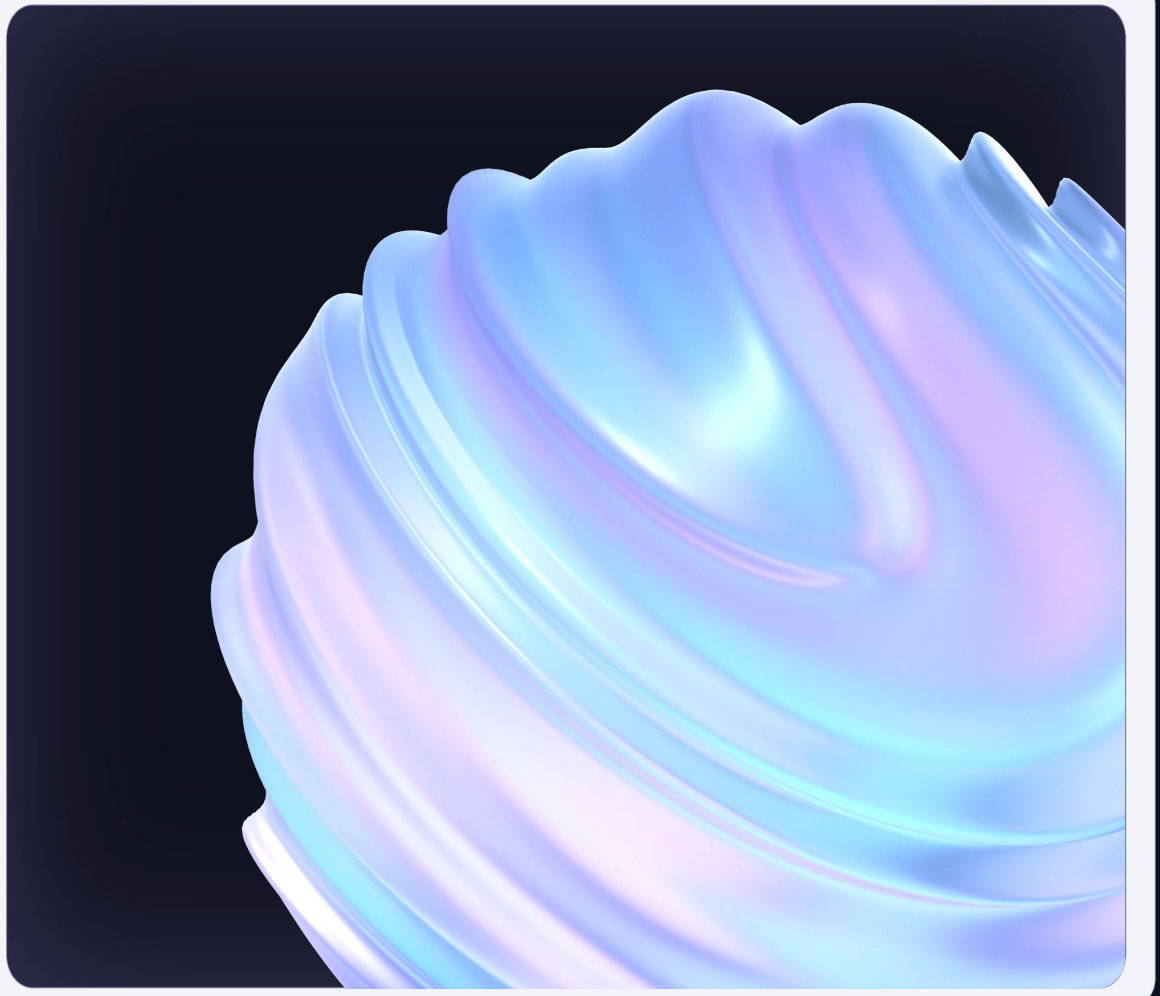


Best Practices for Tool Calling in vLLM

1. Create a custom chat template - models can often be instructed to handle parallel tool calls better; to avoid hallucinating tools; etc.

2. Careful with quantization! You lose precision. FP8 is ok, INT8 maybe (depends on quality/type - pref. AWQ). INT4 quants may not call tools reliably depending on size

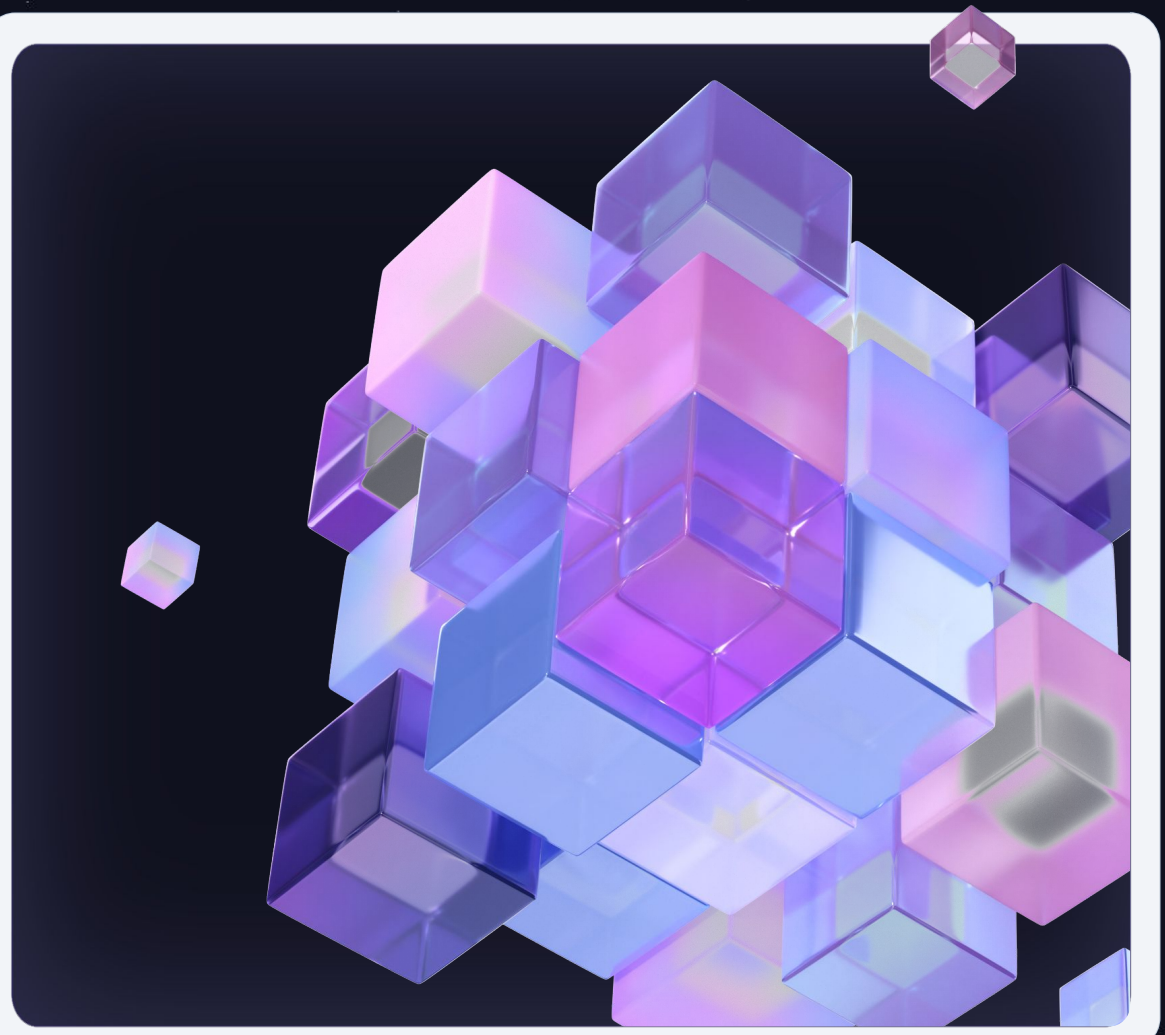
3. Double, triple, quadruple check the chat template if you have issues, even if you're using the official one! Even official ones often have bugs!!!



The Future of Open-Source Function Calling

What can we look forward to with function-calling in the future?

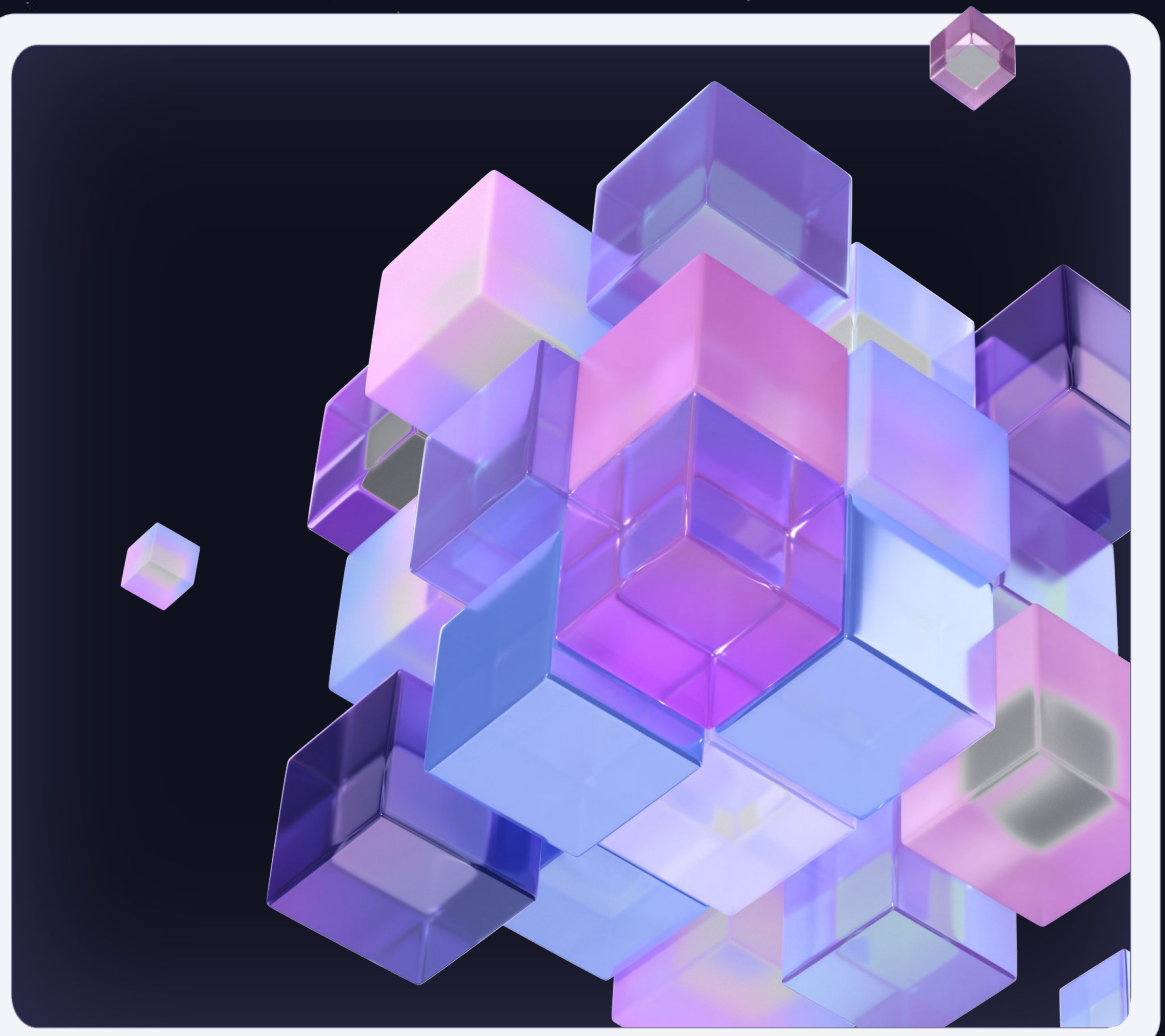
- Easier to fine-tune function-calling models as datasets proliferate (Axolotl!)
- Function call formats - right now it's the wild west (chat templates, anyone?)
- Expect to see function calling formats converge into a couple of standards (with exceptions!)
- Better function-calling on open-source SLMs
- Better toolchain support



vLLM's Role in the Future of Function Calling

How does vLLM fit in?

- First open-source inference/serving framework to support tool streaming!
- Tool architecture allows for extensibility; like chat templates it's easy to bring your own tool parser
- Day-0 support for new LLMs
- Officially recommended by multiple AI labs; many are starting to provide recommended vLLM configurations (Qwen, Mistral, etc)
- **vLLM is becoming the standard inference platform for building open-source agentic applications**



Models with vLLM Tool Parsers

Aa Model / Family	Status	Official Chat Template	Parallel Tool Calls	Non-Greedy Decoding
Nous Research's Hermes 2 & 3 Models	Done	Yes	Yes	Yes
Qwen 2.5 Models	Done	Yes	Yes	Yes
Mistral 7B v0.3, Nemo, Mixtral, Small, Large	Done	No	Mostly	No
Meta's Llama 3.1 (JSON format only)	Done	Yes	No	Yes
AI21 Labs's Jamba 1.5 (Mini & Large)	Done	Yes	Yes	Yes
InternLM 2.5 7B	Done	Yes	No	Yes
IBM's Granite 20B Functioncalling	In progress	Yes	Yes	Yes
IBM's Granite 3.0 Models	In progress	Yes	Yes	Yes
OpenBMB's MiniCPM 3-4B	In progress	Yes	Yes	Yes

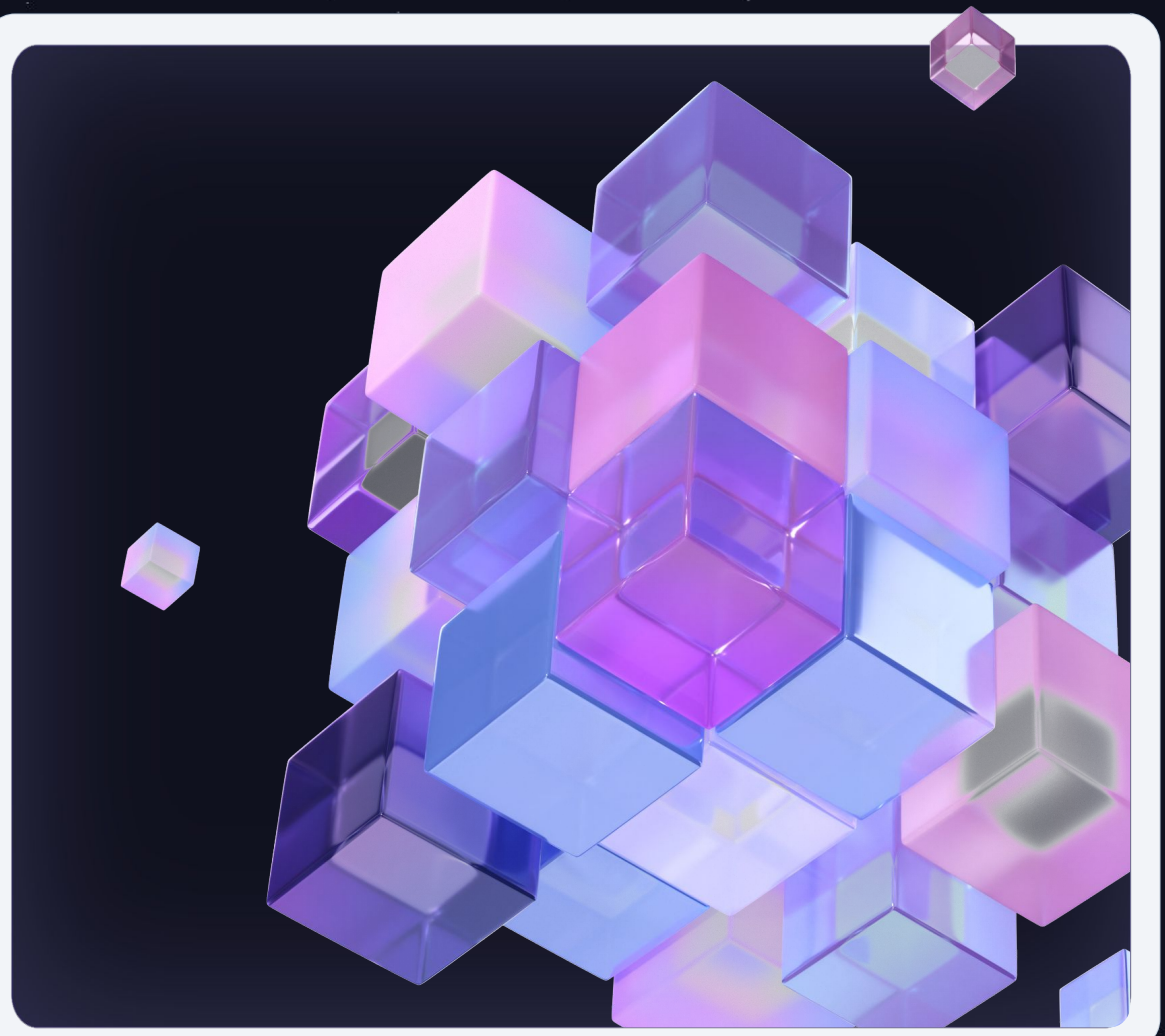
Tool Calling FAQs

What if my model isn't supported?

- Create a PR and add a tool parser 😁
- Open an issue and ask for help!

We launched with 2 parsers - now there are 5 parsers and 6 supported model families, with more on the way!

- You can bring-your-own parser without creating a PR.
- If you don't need streaming, it's just a regex and some parsing - 10 LOC tops :)
- Documented at https://docs.vllm.ai - search "tool parser"



Let’s Have a Discussion!

- Questions…
- Feedback…
- Feature requests…
- How are you using vLLM?
- How can we make office hours better?

Get Involved With the Community

Contribute to key vLLM features

- Comment and review PRs that are interesting to you
- Join the discussion on RFCs
- Check out "good first issue" tags
- Build examples and demos with other tools

Give Us Feedback

We'll email you today's recording as soon as it's ready.
Respond and tell us what we are doing right and what we can do better with vLLM office hours.
Or comment on this slide!

Meet Us at Upcoming Events

Last in-person vLLM meetup of 2024!

- Nov. 13 at the Snowflake HQ

Let's grab a coffee! We'll be at:

- **NeurIPS**, Vancouver (Dec 9 - Dec 15)

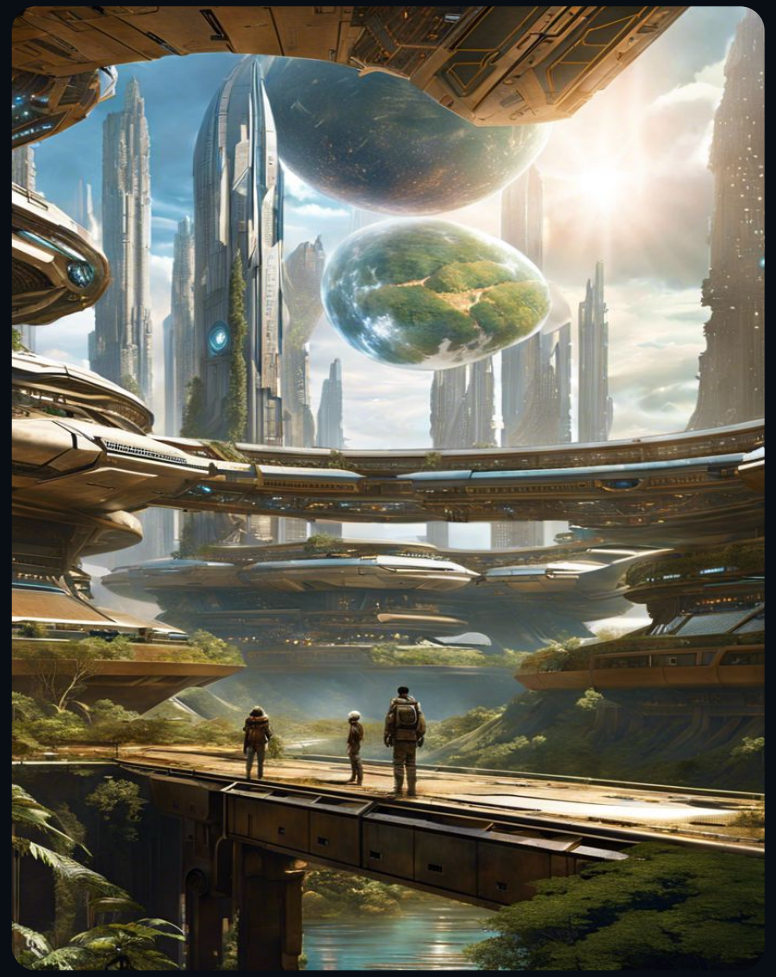
Which events are you attending?

Join Neural Magic's Mission

Neural Magic wants to bring open-source LLMs and vLLM to every enterprise on the planet.

We are looking for **vLLM Engineers** to help us accomplish our mission.

https://neuralmagic.com/careers



Neural Magic | vLLM

THANK YOU!

Michael Goin
Engineering Lead at Neural Magic
vLLM Committer

Kyle Mistele
Founder of **Constellate AI**