Qwen3: Think Deeper, Act Faster

April 29, 2025 · 10 min · 2036 words · Qwen Team | Translations: 简体中文

QWEN CHAT ↗ GITHUB ↗ HUGGING FACE ↗ MODELSCOPE ↗ KAGGLE ↗

DEMO ↗ DISCORD ↗

Introduction

Today, we are excited to announce the release of **Qwen3**, the latest addition to the Qwen family of large language models. Our flagship model, **Qwen3-235B-A22B**, achieves competitive results in benchmark evaluations of coding, math, general capabilities, etc., when compared to other top-tier models such as DeepSeek-R1, o1, o3-

Qwen2.5-72B-Instruct.

	Qwen3-235B-A22B MoE	Qwen3-32B Dense	OpenAI-o1 2024-12-17	Deepseek-R1	Grok 3 Beta Think	Gemini2.5-Pro	OpenAI-o3-mini Medium
ArenaHard	95.6	93.8	92.1	93.2	-	96.4	89.0
AIME'24	85.7	81.4	74.3	79.8	83.9	92.0	79.6
AIME'25	81.5	72.9	79.2	70.0	77.3	86.7	74.8
LiveCodeBench v5, 2024.10-2025.02	70.7	65.7	63.9	64.3	70.6	70.4	66.3
CodeForces Elo Rating	2056	1977	1891	2029	-	2001	2036
Aider Pass@2	61.8	50.2	61.7	56.9	53.3	72.9	53.8
LiveBench 2024-11-25	77.1	74.9	75.7	71.6	-	82.4	70.0
BFCL v3	70.8	70.3	67.8	56.9	-	62.9	64.6
MultiIF 8 Languages	71.9	73.0	48.8	67.7	-	77.8	48.4

1. AIME 24/25: We sample 64 times for each query and report the average of the accuracy. AIME'25 consists of Part I and Part II, with a total of 30 questions.
2. Aider: We didn't activate the think mode of Qwen3 to balance efficiency and effectiveness.
3. BFCL: The Qwen3 models are evaluated using the FC format, while the baseline models are assessed using the highest scores obtained from either the FC or prompt formats.

	Qwen3-30B-A3B MoE	QwQ-32B	Qwen3-4B Dense	Qwen2.5-72B-Instruct	Gemma3-27B-IT	DeepSeek-V3	GPT-4o 2024-11-20
ArenaHard	91.0	89.5	76.6	81.2	86.8	85.5	85.3
AIME'24	80.4	79.5	73.8	18.9	32.6	39.2	11.1
AIME'25	70.9	69.5	65.6	15.0	24.0	28.8	7.6
LiveCodeBench v5, 2024.10-2025.02	62.6	62.7	54.2	30.7	26.9	33.1	32.7
CodeForces Elo Rating	1974	1982	1671	859	1063	1134	864
GPQA	65.8	65.6	55.9	49.0	42.4	59.1	46.0
LiveBench 2024-11-25	74.3	72.0	63.6	51.4	49.2	60.5	52.2
BFCL v3	69.1	66.4	65.9	63.4	59.1	57.6	72.5
MultiIF 8 Languages	72.2	68.3	66.3	65.3	69.8	55.6	65.6

1. AIME 24/25: We sample 64 times for each query and report the average of the accuracy. AIME'25 consists of Part I and Part II, with a total of 30 questions.
2. Aider: We didn't activate the think mode of Qwen3 to balance efficiency and effectiveness.
3. BFCL: The Qwen3 models are evaluated using the FC format, while the baseline models are assessed using the highest scores obtained from either the FC or prompt formats.

We are open-weighting two MoE models: **Qwen3-235B-A22B**, a large model with 235 billion total parameters and 22 billion activated parameters, and **Qwen3-30B-A3B**, a smaller MoE model with 30 billion total parameters and 3 billion activated parameters. Additionally, six dense models are also open-weighted, including **Qwen3-32B**, **Qwen3-14B**, **Qwen3-8B**, **Qwen3-4B**, **Qwen3-1.7B**, and **Qwen3-0.6B**, under Apache 2.0 license.

Models	Layers	Heads (Q / KV)	Tie Embedding	Context Length
Qwen3-0.6B	28	16 / 8	Yes	32K
Qwen3-1.7B	28	16 / 8	Yes	32K

Qwen3-4B	36	32 / 8	Yes	32K
Qwen3-8B	36	32 / 8	No	128K
Qwen3-14B	40	40 / 8	No	128K
Qwen3-32B	64	64 / 8	No	128K

Models	Layers	Heads (Q / KV)	# Experts (Total / Activated)	Context Length
Qwen3-30B-A3B	48	32 / 4	128 / 8	128K
Qwen3-235B-A22B	94	64 / 4	128 / 8	128K

The post-trained models, such as **Qwen3-30B-A3B**, along with their pre-trained counterparts (e.g., **Qwen3-30B-A3B-Base**), are now available on platforms like **Hugging Face**, **ModelScope**, and **Kaggle**. For deployment, we recommend using frameworks like **SGLang** and **vLLM**. For local usage, tools such as **Ollama**, **LMStudio**, **MLX**, **llama.cpp**, and **KTransformers** are highly recommended. These options ensure that users can easily integrate Qwen3 into their workflows, whether in research, development, or production environments.

We believe that the release and open-sourcing of Qwen3 will significantly advance the research and development of large foundation models. Our goal is to empower researchers, developers, and organizations around the world to build innovative solutions using these cutting-edge models.

Feel free to try Qwen3 out in Qwen Chat Web (chat.qwen.ai) and mobile APP!

Key Features

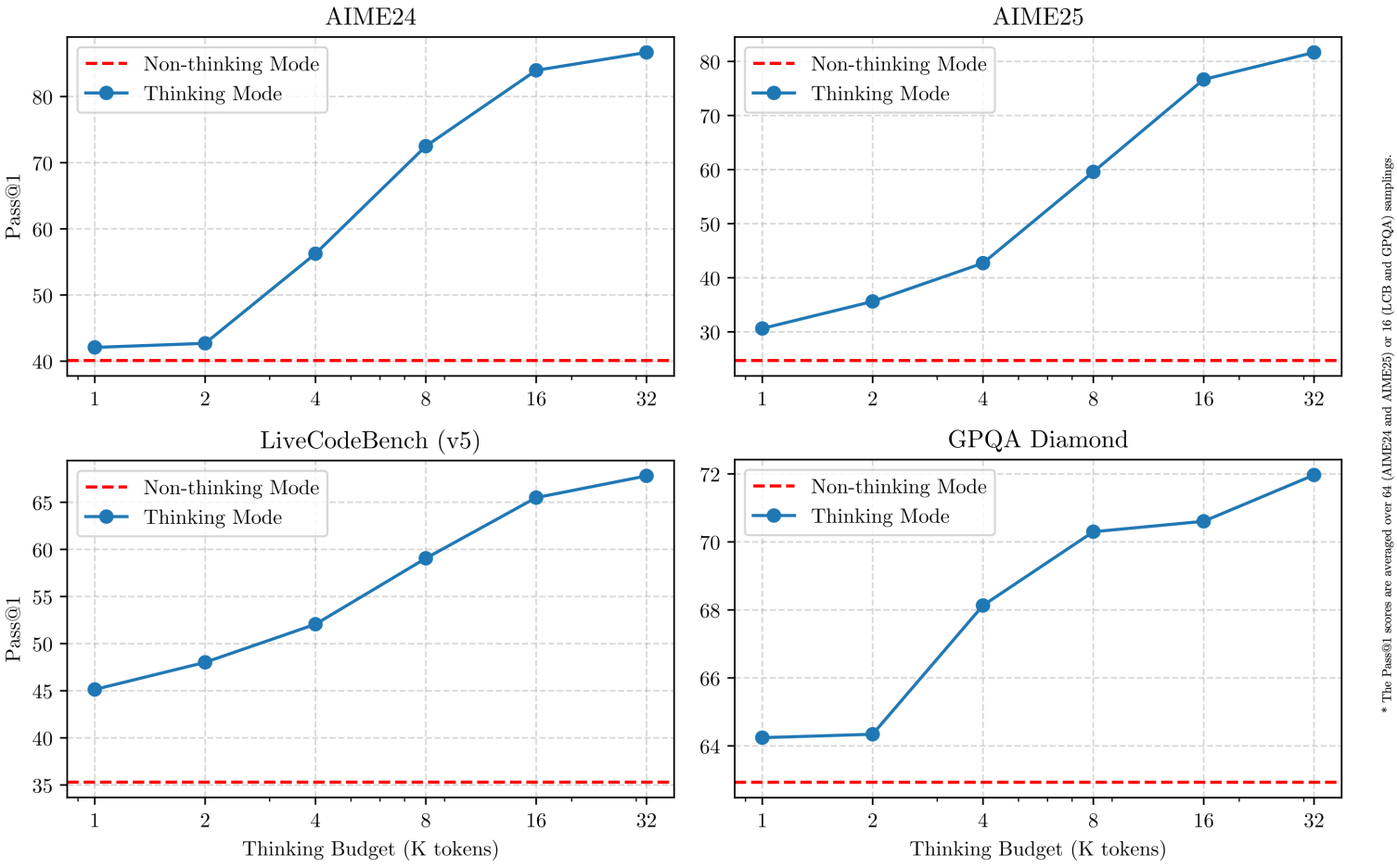
- **Hybrid Thinking Modes**

Qwen3 models introduce a hybrid approach to problem-solving. They support two modes:

1. Thinking Mode: In this mode, the model takes time to reason step by step before delivering the final answer. This is ideal for complex problems that require deeper thought.
2. Non-Thinking Mode: Here, the model provides quick, near-instant responses, suitable for simpler questions where speed is more important than depth.

This flexibility allows users to control how much “thinking” the model performs based on the task at hand. For example, harder problems can be tackled with extended reasoning, while easier ones can be answered directly

performance improvements that are directly correlated with the computational reasoning budget allocated. This design enables users to configure task-specific budgets with greater ease, achieving a more optimal balance between cost efficiency and inference quality.



- **Multilingual Support**

Qwen3 models are supporting **119 languages and dialects**. This extensive multilingual capability opens up new possibilities for international applications, enabling users worldwide to benefit from the power of these models.

Language Family	Languages & Dialects
Indo-European	English, French, Portuguese, German, Romanian, Swedish, Danish, Bulgarian, Russian, Czech, Greek, Ukrainian, Spanish, Dutch, Slovak, Croatian, Polish, Lithuanian, Norwegian Bokmål, Norwegian Nynorsk, Persian, Slovenian, Gujarati, Latvian, Italian, Occitan, Nepali, Marathi, Belarusian, Serbian, Luxembourgish, Venetian, Assamese, Welsh, Silesian, Asturian, Chhattisgarhi, Awadhi, Maithili, Bhojpuri, Sindhi, Irish, Faroese, Hindi, Punjabi, Bengali, Oriya, Tajik, Eastern Yiddish, Lombard, Ligurian, Sicilian, Friulian, Sardinian, Galician, Catalan, Icelandic, Tosk Albanian, Limburgish, Dari, Afrikaans, Macedonian, Sinhala, Urdu, Magahi, Bosnian, Armenian

Sino-Tibetan	Chinese (Simplified Chinese, Traditional Chinese, Cantonese), Burmese
Afro-Asiatic	Arabic (Standard, Najdi, Levantine, Egyptian, Moroccan, Mesopotamian, Ta'izzi-Adeni, Tunisian), Hebrew, Maltese
Austronesian	Indonesian, Malay, Tagalog, Cebuano, Javanese, Sundanese, Minangkabau, Balinese, Banjar, Pangasinan, Iloko, Waray (Philippines)
Dravidian	Tamil, Telugu, Kannada, Malayalam
Turkic	Turkish, North Azerbaijani, Northern Uzbek, Kazakh, Bashkir, Tatar
Tai-Kadai	Thai, Lao
Uralic	Finnish, Estonian, Hungarian
Austroasiatic	Vietnamese, Khmer
Other	Japanese, Korean, Georgian, Basque, Haitian, Papiamento, Kabuverdianu, Tok Pisin, Swahili

- **Improved Agentic Capabilities**

We have optimized the Qwen3 models for coding and agentic capabilities, and also we have strengthened the support of MCP as well. Below we provide examples to show how Qwen3 thinks and interacts with the environment.

0:00 / 0:56

Pre-training

In terms of pretraining, the dataset for Qwen3 has been significantly expanded compared to Qwen2.5. While Qwen2.5 was pre-trained on 18 trillion tokens, Qwen3 uses nearly twice that amount, with approximately 36 trillion tokens covering 119 languages and dialects. To build this large dataset, we collected data not only from the web but also from PDF-like documents. We used Qwen2.5-VL to extract text from these documents and Qwen2.5 to improve the quality of the extracted content. To increase the amount of math and code data, we used Qwen2.5-Math and Qwen2.5-Coder to generate synthetic data. This includes textbooks, question-answer pairs, and code snippets.

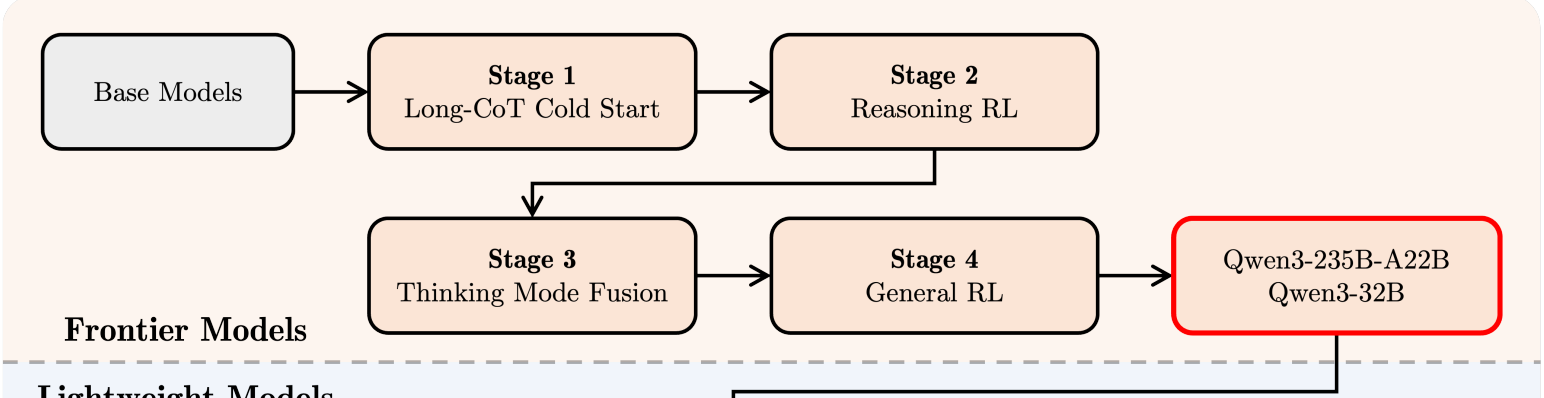
The pre-training process consists of three stages. In the first stage (S1), the model was pretrained on over 30 trillion tokens with a context length of 4K tokens. This stage provided the model with basic language skills and general knowledge. In the second stage (S2), we improved the dataset by increasing the proportion of knowledge-intensive data, such as STEM, coding, and reasoning tasks. The model was then pretrained on an additional 5 trillion tokens. In the final stage, we used high-quality long-context data to extend the context length to 32K tokens. This ensures the model can handle longer inputs effectively.

	Qwen2.5-72B Base	Qwen2.5-Plus Base	LLaMA-4-Maverick Base	DeepSeek-V3 Base	Qwen3-235B-A22B Base

# Activated Params	72B	37B	17B	37B	22B
			General Tasks		
MMLU	86.06	85.02	85.16	87.19	**87.81**
MMLU-Redux	83.91	82.69	84.05	86.14	**87.40**
MMLU-Pro	58.07	63.52	63.91	59.84	**68.18**
SuperGPQA	36.20	37.18	40.85	41.53	**44.06**
BBH	86.30	85.60	83.62	86.22	**88.87**
			Mathematics & Science Tasks		
GPQA	45.88	41.92	43.94	41.92	**47.47**
GSM8K	91.50	91.89	87.72	87.57	**94.39**
MATH	62.12	62.78	63.32	62.62	**71.84**
			Multilingual tasks		
MGSM	82.40	82.21	79.69	82.68	**83.53**
MMMLU	84.40	83.49	83.09	85.88	**86.70**
INCLUDE	69.05	66.97	73.47	**75.17**	73.46
			Code tasks		
EvalPlus	65.93	61.43	68.38	63.75	**77.60**
MultiPL-E	58.70	62.16	57.28	62.26	**65.94**
MBPP	76.00	74.60	75.40	74.20	**81.40**
CRUX-O	66.20	68.50	77.00	76.60	**79.00**

Due to advancements in model architecture, increase in training data, and more effective training methods, the overall performance of Qwen3 dense base models matches that of Qwen2.5 base models with more parameters. For instance, Qwen3-1.7B/4B/8B/14B/32B-Base performs as well as Qwen2.5-3B/7B/14B/32B/72B-Base, respectively. Notably, in areas like STEM, coding, and reasoning, Qwen3 dense base models even outperform larger Qwen2.5 models. For Qwen3-MoE base models, they achieve similar performance to Qwen2.5 dense base models while using only 10% of the active parameters. This results in significant savings in both training and inference costs.

Post-training



To develop the hybrid model capable of both step-by-step reasoning and rapid responses, we implemented a four-stage training pipeline. This pipeline includes: (1) long chain-of-thought (CoT) cold start, (2) reasoning-based reinforcement learning (RL), (3) thinking mode fusion, and (4) general RL.

In the first stage, we fine-tuned the models using diverse long CoT data, covering various tasks and domains such as mathematics, coding, logical reasoning, and STEM problems. This process aimed to equip the model with fundamental reasoning abilities. The second stage focused on scaling up computational resources for RL, utilizing rule-based rewards to enhance the model's exploration and exploitation capabilities.

In the third stage, we integrated non-thinking capabilities into the thinking model by fine-tuning it on a combination of long CoT data and commonly used instruction-tuning data. This data was generated by the enhanced thinking model from the second stage, ensuring a seamless blend of reasoning and quick response capabilities. Finally, in the fourth stage, we applied RL across more than 20 general-domain tasks to further strengthen the model's general capabilities and correct undesired behaviors. These tasks included instruction following, format following, and agent capabilities, etc.

Develop with Qwen3

Below is a simple guide for you to use Qwen3 on different frameworks. First of all, we provide an standard example of using Qwen3-30B-A3B in Hugging Face transformers:

```
from modelscope import AutoModelForCausalLM, AutoTokenizer

model_name = "Qwen/Qwen3-30B-A3B"

# load the tokenizer and the model
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype="auto",
    device_map="auto"
)

# prepare the model input
prompt = "Give me a short introduction to large language model."
messages = [
    {"role": "user", "content": prompt}
]
text = tokenizer.apply_chat_template(
```

```
    enable_thinking=True # Switch between thinking and non-thinking modes. Default is True.
)
model_inputs = tokenizer([text], return_tensors="pt").to(model.device)

# conduct text completion
generated_ids = model.generate(
    **model_inputs,
    max_new_tokens=32768
)
output_ids = generated_ids[0][len(model_inputs.input_ids[0]):].tolist()

# parsing thinking content
try:
    # rindex finding 151668 (</think>)
    index = len(output_ids) - output_ids[::-1].index(151668)
except ValueError:
    index = 0

thinking_content = tokenizer.decode(output_ids[:index], skip_special_tokens=True).strip("\n")
content = tokenizer.decode(output_ids[index:], skip_special_tokens=True).strip("\n")

print("thinking content:", thinking_content)
print("content:", content)
```

To disable thinking, you just need to make changes to the argument `enable_thinking` like the following:

```
text = tokenizer.apply_chat_template(
    messages,
    tokenize=False,
    add_generation_prompt=True,
    enable_thinking=False  # True is the default value for enable_thinking.
)
```

For deployment, you can use `sglang>=0.4.6.post1` or `vllm>=0.8.4` to create an OpenAI-compatible API endpoint:

- SGLang:

  ```
  python -m sglang.launch_server --model-path Qwen/Qwen3-30B-A3B --reasoning-parser qwen3
  ```

- vLLM:

  ```
  vllm serve Qwen/Qwen3-30B-A3B --enable-reasoning --reasoning-parser deepseek_r1
  ```

If you use it for local development, you can use ollama by running a simple command `ollama run qwen3:30b-a3b` to play with the model, or you can use LMStudio or llama.cpp and ktransformers to build locally.

Advanced Usages

We provide a soft switch mechanism that allows users to dynamically control the model's behavior when enable_thinking=True. Specifically, you can add /think and /no_think to user prompts or system messages to

Here is an example of a multi-turn conversation:

```python
from transformers import AutoModelForCausalLM, AutoTokenizer

class QwenChatbot:
    def __init__(self, model_name="Qwen/Qwen3-30B-A3B"):
        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
        self.model = AutoModelForCausalLM.from_pretrained(model_name)
        self.history = []

    def generate_response(self, user_input):
        messages = self.history + [{"role": "user", "content": user_input}]

        text = self.tokenizer.apply_chat_template(
            messages,
            tokenize=False,
            add_generation_prompt=True
        )

        inputs = self.tokenizer(text, return_tensors="pt")
        response_ids = self.model.generate(**inputs, max_new_tokens=32768)[0][len(inputs.input_ids[0]):].t
        response = self.tokenizer.decode(response_ids, skip_special_tokens=True)

        # Update history
        self.history.append({"role": "user", "content": user_input})
        self.history.append({"role": "assistant", "content": response})

        return response

# Example Usage
if __name__ == "__main__":
    chatbot = QwenChatbot()

    # First input (without /think or /no_think tags, thinking mode is enabled by default)
    user_input_1 = "How many r's in strawberries?"
    print(f"User: {user_input_1}")
    response_1 = chatbot.generate_response(user_input_1)
    print(f"Bot: {response_1}")
    print("----------------------")

    # Second input with /no_think
    user_input_2 = "Then, how many r's in blueberries? /no_think"
    print(f"User: {user_input_2}")
    response_2 = chatbot.generate_response(user_input_2)
    print(f"Bot: {response_2}")
    print("----------------------")

    # Third input with /think
    user_input_3 = "Really? /think"
    print(f"User: {user_input_3}")
    response_3 = chatbot.generate_response(user_input_3)
    print(f"Bot: {response_3}")
```

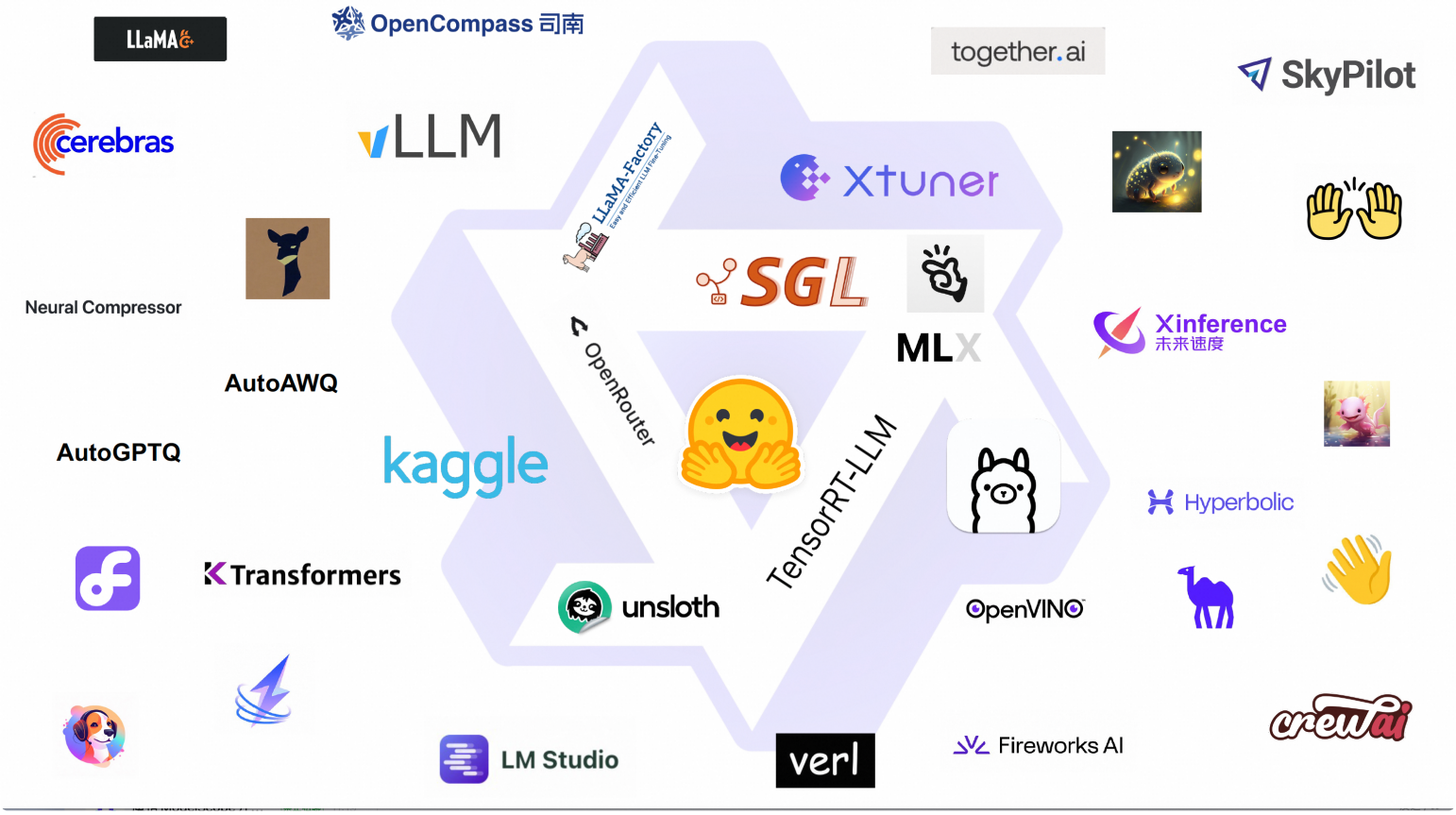
Qwen3 excels in tool calling capabilities. We recommend using Qwen-Agent to make the best use of agentic ability of Qwen3. Qwen-Agent encapsulates tool-calling templates and tool-calling parsers internally, greatly reducing coding complexity.

To define the available tools, you can use the MCP configuration file, use the integrated tool of Qwen-Agent, or integrate other tools by yourself.

```python
from qwen_agent.agents import Assistant

# Define LLM
llm_cfg = {
    'model': 'Qwen3-30B-A3B',

    # Use the endpoint provided by Alibaba Model Studio:
    # 'model_type': 'qwen_dashscope',
    # 'api_key': os.getenv('DASHSCOPE_API_KEY'),

    # Use a custom endpoint compatible with OpenAI API:
    'model_server': 'http://localhost:8000/v1',  # api_base
    'api_key': 'EMPTY',

    # Other parameters:
    # 'generate_cfg': {
    #         # Add: When the response content is `<think>this is the thought</think>this is the answer;
    #         # Do not add: When the response has been separated by reasoning_content and content.
    #         'thought_in_content': True,
    #     },
}

# Define Tools
tools = [
    {'mcpServers': {  # You can specify the MCP configuration file
            'time': {
                'command': 'uvx',
                'args': ['mcp-server-time', '--local-timezone=Asia/Shanghai']
            },
            "fetch": {
                "command": "uvx",
                "args": ["mcp-server-fetch"]
            }
        }
    },
  'code_interpreter',  # Built-in tools
]

# Define Agent
bot = Assistant(llm=llm_cfg, function_list=tools)

# Streaming generation
messages = [{'role': 'user', 'content': 'https://qwenlm.github.io/blog/ Introduce the latest developments
for responses in bot.run(messages=messages):
    pass
print(responses)
```

Friends of Qwen

Thanks to the support of so many friends. Qwen is nothing without its friends! We welcome more people or organizations to join our community and help us become better!



Future Work

Qwen3 represents a significant milestone in our journey toward Artificial General Intelligence (AGI) and Artificial Superintelligence (ASI). By scaling up both pretraining and reinforcement learning (RL), we have achieved higher levels of intelligence. We have seamlessly integrated thinking and non-thinking modes, offering users the flexibility to control the thinking budget. Additionally, we have expanded support for a wide range of languages, enhancing global accessibility.

Looking ahead, we aim to enhance our models across multiple dimensions. This includes refining model architectures and training methodologies to achieve several key objectives: scaling data, increasing model size, extending context length, broadening modalities, and advancing RL with environmental feedback for long-horizon