

Deploying & Serving Your LLM App

Now that you've fine-tuned and evaluated your LLM, it's time to bring it into production. In this section, we'll walk through how to deploy your model as an accessible, scalable service—so it can start delivering real value to users.

Throughout the course we've been using LLMs through APIs provided by **Model Studio** (models like **qwen-plus**) and **DashScope** (models like **text-embedding-v2**, **text-embedding-v3**). It's a powerful shortcut and saves a lot of time in deploying and troubleshooting models on the cloud.

However, in some cases you may want to deploy your own model. Deploying a model means moving it from development into a production environment where it can serve real-time inference requests. But deployment isn't always required.

Let's explore your options.

```
graph TD
    subgraph Deployment_Options [Deployment Options]
        A[Where to deploy my LLM?] --> B{Deployment path}
        B --> C[Direct API Call]
        C --> D[Model Studio]
        D --> E[Managed service]
    end

    No_deployment[No deployment needed]
    Rate_limited[Rate-limited]
    Great_for_Tongyi[Great for Tongyi model family]

    B --> F[Self-Hosted]
    F --> G[Local vLLM]
    G --> H[Test only]

    Single_node[Single node]
    No_scaling[No scaling]

    F --> I[Cloud Deployment]
    I --> L[PAI-EAS]
    I --> M[ECS / EGS / ACK / ACS]

    L --> P[Elastic inference]

    Real_time[Real-time, high-concurrency]
    M --> Q[Full control]

    For_complex[For complex or custom setups]

    end
```

```
style C fill:#D6EAF8,stroke:#3498DB
style F fill:#D5F5E3,stroke:#2ECC71
style D fill:#EBDEF0,stroke:#8E44AD
style G fill:#FEF9E7,stroke:#F4D03F
style L fill:#ABEBC6,stroke:#27AE60
style M fill:#ABEBC6,stroke:#27AE60
```

The story so far...

You've come a long way since the early days of **TaskFriend**.

What started as a simple AI assistant that could chat about tasks has evolved into a **smart, agentic system** capable of retrieving private data, understanding complex queries like *"I'm stuck — what should I do next?"*, and even planning weekend trips by checking calendars and weather.

You've engineered prompts, optimized retrieval with reranking and HyDE, evaluated performance with Ragas, and built multi-agent workflows that act on behalf of the user.

Now, it's time to take **TaskFriend** from your development environment into the real world.

Because no matter how intelligent your AI is — if it's not deployed, it can't help anyone.

But deployment isn't just about "putting it online." It's about making smart trade-offs:

- Should you use a managed API for simplicity, or self-host for control?
- How do you serve models efficiently at scale?
- What does it mean to run an LLM app in production?

This chapter closes the loop — from idea, to prototype, to **a production-ready, deployable AI application**.

Goals

- Understand when to **deploy** your LLM app — and when **not to**.
- Compare **managed services** (e.g., Model Studio) vs. **self-hosted solutions** (e.g., vLLM, PAI-EAS).
- Learn how to **serve models efficiently** using high-performance inference engines like **vLLM**.
- Run a **local deployment** with OpenAI-compatible APIs for fast testing.
- Deploy to the cloud using **PAI-EAS** for elastic, high-concurrency serving.
- Evaluate **latency, throughput, and reliability** in real-world conditions.
- Make informed decisions about **cost, scalability, and control** in production.

Resource-light "Deployment"

Skip deployment completely: Just call the API

Let's start with a radical idea: you don't always need to deploy a model. If you're using a pre-trained, cloud-hosted model like those available in [Model Studio](#), you can skip deployment entirely.

This is what you've been doing throughout the entire **[LMP-C01] LLM Engineer (Professional)** course. You didn't deploy the models you used. The models were already live — hosted, scaled, and managed by Alibaba Cloud via Model Studio.

That's the power of managed LLM APIs.

You just send a request. You get a response. No servers. No scaling headaches.

Why this works well

Think of it like using Gmail instead of running your own email server.

Benefit	What It Means for You
Zero deployment effort	No DevOps, no containers, no GPU setup
Pay-per-use pricing	Only pay for tokens consumed — no idle GPU costs
Automatic scaling	Handles traffic spikes without configuration
Built-in reliability	High availability, SLAs, and monitoring included

This is perfect for:

- Prototypes and PoCs
- Low-to-medium traffic apps
- Teams that want to focus on product, not infrastructure

⚠ Limits:
APIs are subject to rate limiting (e.g., QPM: queries per minute, TPM: tokens per minute). Exceeding these limits results in failed requests.

And if your model is **custom**, **fine-tuned**, or **not supported by Model Studio**, you'll need to take matters into your own hands.

Deploy Your LLM Application on Alibaba Cloud

As technology evolves, more and more organizations and individuals are moving away from purchasing their own high-performance servers and adopting cloud services as the primary means of LLM deployment. This approach is provides an overall improvement on multiple fronts:

- **High Resource Cost:** Requires a significant upfront investment to purchase a large number of high-performance servers.
- **High Operational Cost:** Daily maintenance of the servers—including monitoring, upgrades, and troubleshooting—requires a high level of technical expertise.
- **Reliability Concerns:** Service stability and reliability depend heavily on the skill of the maintenance personnel and the project's budget. It is difficult to establish a highly available and stable model service under limited costs.
- **Low Flexibility:** Being constrained by fixed hardware resources, it is impossible to dynamically adjust resources based on actual needs, leading to either inadequate model service performance or

resource wastage.

Alibaba Cloud provides a wealth of services such as **Model Studio**, **PAI-EAS**, **GPU-accelerated ECS instances**, **Container Service for Kubernetes (ACK)**, or **Container Compute Service (ACS)** that you can choose from to use or deploy your LLM service or application. As cloud services, they are highly flexible and scalable - a prime choice for production-grade LLM services, while giving you the flexibility to quickly adapt to changing business needs.

Deploying to a Linux environment (with vLLM)

It's always a good idea to test out deployment on a small model. The popular choice would be to use a **Linux-based** environment, as that is what you'll be using on the cloud. There are many frameworks to deploy models - but the most popular would be **vLLM** - a blazing-fast, open-source inference engine designed for LLMs.

Think of vLLM as your model's "test track." It lets you:

- Serve models with minimal code
- Benchmark performance under load
- Simulate real API behavior with OpenAI-compatible endpoints

We'll use the **Qwen2.5-1.5B-Instruct** model — small enough to run on a single GPU, powerful enough to impress.

Setting up the environment

We'll continue to use PAI-DSW, but before that, we need to switch to a different instance - one that's **equipped with a GPU**.

If you're already using a GPU-accelerated PAI-DSW instance, continue using it.

If not, see: [00 Setting Up the Environment](#)

Once that's done, let's download our model from the [HuggingFace model library](#):

```
from huggingface_hub import snapshot_download

# Specify the repository ID of the model on Hugging Face Hub
repo_id = "Qwen/Qwen2.5-1.5B-Instruct"

# Specify the local directory where you want to save the model
local_dir = "./model/qwen2_5-1_5b-instruct"

# Download the model
snapshot_download(repo_id=repo_id, local_dir=local_dir)
```

To start using the model, we'll have to install vLLM.

```
pip install vllm
```

Then, serve your model on vLLM.

```
vllm serve "./model/qwen2_5-1_5b-instruct" --load-format "safetensors" --port 8000
```

Testing out your vLLM deployment

vLLM lets you run a local server that is compatible with the OpenAI API. This means you can interact with your own locally hosted large language models (like [Qwen2.5-1.5B-Instruct](#)) using the same API format as OpenAI. You can test whether the model service is working correctly by sending an HTTP request using [curl](#).

```
%bash
curl -s -X POST http://localhost:8000/v1/chat/completions \
  -H "Content-Type: application/json" \
  -d '{
    "model": "./model/qwen2_5-1_5b-instruct",
    "messages": [
      {"role": "system", "content": "You are TaskFriend, a helpful
AI assistant that helps users manage daily tasks, prioritize work, and
optimize time."},
      {"role": "user", "content": "I think I am burning out, can you
help me?"}
    ]
  }'
```

If you get a successful response, it means that your local vLLM server is up and running.

In addition, the service is compatible with the [/v1/models](#) endpoint, which allows you to view the list of deployed models. For more information, see vLLM's documentation on [OpenAI-Compatible Server](#).

```
%bash
curl -X GET http://localhost:8000/v1/models
```

Evaluating service performance

Now, to evaluate the performance of the deployed model service, we'll use a simple HTTP performance testing tool called [wrk](#) to simulate load testing requests and generate performance reports. The following example demonstrates how to perform load testing on the [POST /v1/chat/completions](#) endpoint to measure key performance metrics.

Step 1: Install [wrk](#)

Go back to your terminal and install [wrk](#)

```
sudo apt update
sudo apt install wrk
```

Step 2: Prepare the POST request body

Prepare the body data for the POST request. We've already prepared this for you in `./resources/post.lua`, with the following content:

```
wrk.method = "POST"
wrk.headers["Content-Type"] = "application/json"
wrk.body = [[
    {
        "model": "./model/qwen2_5-1_5b-instruct",
        "messages": [
            {"role": "system", "content": "You are a helpful AI
assistant."},
            {"role": "user", "content": "Say hi!"}
        ]
    }
]]
```

Step 3: Run the stress test with `wrk`

Next, execute the `wrk` stress testing command in the terminal. Two tests are performed with different concurrency levels (`-c`), both lasting for 20 seconds (`-d 10s`). The goal is to observe how the service performs under different loads.

```
%bash

echo "Test 1: concurrency = 1"
echo "-----"
wrk -t1 -c1 -d10s -s ./resources/post.lua
http://localhost:8000/v1/chat/completions
echo -e "\n\n"

echo "Test 2: concurrency = 20"
echo "-----"
wrk -t1 -c20 -d10s -s ./resources/post.lua
http://localhost:8000/v1/chat/completions
```

Step 4: Evaluate stress testing results

You should get a load testing result similar to this:



Stress test results for local vLLM server

From the test results we got, we see that as the concurrency level increases from **1** to **20**:

- **Average latency** increases by close to 50% (from 539.36ms to 794.34ms)
- **QPS (Queries Per Second)** increases by approximately 12 times (from 1.80 to 24.07)

At even higher load, we can expect to see some timeouts - but be careful: timeouts may also mean that the **wrk** tool did not receive a complete response (which can happen if the LLM's response is too long, i.e. cannot be fully returned within the timeframe.)

Deploying models with PAI-EAS

Elastic Algorithm Service (PAI-EAS), is part of **Platform for AI (PAI)**, which provides a robust platform for deploying both open-source and custom-trained models as production-grade online services. PAI-EAS supports advanced features such as:

- Auto-scaling
- Blue-green deployments
- Version management
- Resource monitoring
- Model pre-warming (to eliminate cold starts)

These capabilities make PAI-EAS particularly well-suited for **real-time synchronous inference** scenarios where low latency and high availability are critical.

For cost-sensitive applications, you can choose to use PAI-EAS Spot. This offering leverages the use of preemptible instances, allowing you to run inference on lower-cost, interruptible instances—ideal for non-critical or fault-tolerant workloads while keeping costs low. For more information see: [Best practices for PAI-EAS Spot](#).

Deployment guide:

Get started quickly with [Quickly deploy LLMs in EAS](#).

Deploying models with ECS or Container Services

Elastic Compute Service (ECS) and container-based platforms (like ACK or ACS) are some of the most popular approaches when deploying LLMs. It provides you with complete control over the deployment environment. This approach is ideal for models that require:

- Specific OS configurations
- Custom software dependencies
- Fine-grained performance tuning
- Long-running, high-throughput inference services

ECS provides persistent, round-the-clock services, and can be integrated with:

- **Auto Scaling** for auto-scaling

- **Server Load Balancer (SLB)** for high availability
- **Security Groups and VPCs** for enhanced security

Alibaba Cloud also provides container-based platforms like **Container Service for Kubernetes (ACK)** or **Container Compute Service (ACS)** for users who are familiar with Kubernetes and container-based environments. Users can choose to go for semi-managed or fully managed services based on their preferences.

While these services offer maximum flexibility, it also demands higher technical expertise and ongoing operational maintenance, so you'll need to evaluate the suitability of these instances against your use case or scenario:

Suitability	Scenarios
High	<ul style="list-style-type: none">• Large models requiring stable, long-term performance• Enterprises needing full control over resources and costs• Complex inference pipelines with custom optimizations
Low	<ul style="list-style-type: none">• Rapid prototyping or low-resource teams• Projects requiring quick deployment and minimal O&M

Deployment guides:

- Elastic Compute Service
 - [Build an LLM inference environment that supports security measurement on a heterogeneous confidential computing instance](#)
- Container-based platforms
 - Container Service for Kubernetes:
 - [Quickly deploy a model in ACK](#)
 - [Use vLLM to deploy a Qwen model as an inference service in ACK](#)
 - [Best practice for deploying the DeepSeek full version across multiple nodes in ACK](#)
 - Container Compute Service:
 - [Deploy Qwen3-32B on GPU-accelerated ACS clusters](#)
 - [Build AI applications in ACS with Dify](#)

Which services should I use?

When using and deploying models on Alibaba Cloud, choosing the right service requires a comprehensive consideration of **business needs**, **model characteristics**, **technical capabilities**, **operational complexity**, and **cost**.

The following table provides a simple reference on choosing Alibaba Cloud services.

Requirement	Model Studio	PAI-EAS	ECS	ACK/ACS
-------------	--------------	---------	-----	---------

Requirement	Model Studio	PAI-EAS	ECS	ACK/ACS
Best for	Directly calling APIs of Qwen & Wan models	Standard NLP/vision model deployment (balanced performance & ease)	Custom environments, complex dependencies, full control	Advanced DevOps, microservices, scalable AI platforms
Model compatibility	<ul style="list-style-type: none">QwenWan	All	All	All
Operational complexity	★ No O&M (fully managed, visual interface)	★★ Low O&M (managed inference, simple config)	★★★ High O&M (self-managed OS, drivers, runtime)	★★★★ High O&M (cluster management, CI/CD, networking)
Team skill level	Non-technical users, product teams	Algorithm engineers, ML practitioners	DevOps, system admins, ML engineers	Experienced DevOps, platform engineers
Cost model	Pay-per-call	Pay-as-you-go	Pay-as-you-go or subscription	Cluster + node + management overhead
Use cases	Public-facing AI apps, quick prototyping	Production-grade APIs, real-time inference	Model development, specialized hardware needs	Multi-model platforms, CI/CD pipelines, hybrid AI systems

Disclaimer: This table is provided for reference only. It may be subject to change as our services evolve or your requirements change.

Alternatively, you can use this decision tree to help choose Alibaba Cloud services for you LLM application:

```
graph TD
    A["Start: Deploying a LLM"] --> B{"What is your core requirement?"}
    B --> C["Rapid app prototyping (e.g., chatbot, assistant)"]
    B --> E["Standard model deployment (image, text, NLP)"]
    B --> F["Custom environment or complex dependencies"]
    C --> G["Model Studio"]
```

```

✓ Best for Qwen models
✓ Best for Wan models
✓ No O&M
✓ Fast launch"]
    E --> I["Use PAI-EAS

✓ Balanced performance & ease
✓ Managed inference
✓ Supports common frameworks"]

    F --> J{"Team is
highly skilled?"}
    J -->|Yes| K["ECS, ACK, or ACS

✓ Full control
✓ Custom images & scaling
✓ For advanced teams"]
    J -->|No| L["Consider PAI-EAS
or Model Studio
for easier management"]

style G fill:#D6EAF8,stroke:#1F618D,color:#154360
style I fill:#FEF9E7,stroke:#F4D03F,color:#9A7D0A
style K fill:#E8DAEF,stroke:#884EA0,color:#4A235A

```

Disclaimer: This decision tree is provided for reference only. It may be subject to change as our services evolve or your requirements change.

What's Next?

Quiz yourself!

► 1. When should you choose self-hosting (on the cloud) over a managed API like Model Studio?

- A) When you want the fastest possible deployment with no DevOps
- B) When you need full control over data, model, and scaling
- C) When you're only testing locally and don't need high availability
- D) When you're using a model not supported by vLLM

View answer →

✓ **Correct answer:** B) When you need full control over data, model, and scaling

🔍 **Explanation:**

- Self-hosting gives you full control over infrastructure, data privacy, and performance tuning.
- Use it when you can't rely on rate-limited or black-box managed services.

- Managed APIs (e.g., Model Studio) are better for rapid prototyping.

Takeaways

- **You don't always need to deploy**
 - **If you're using a managed service (e.g., Model Studio), no deployment is needed** — the model is already hosted and scaled.
 - **This is ideal for prototyping, learning, or low-to-medium traffic apps.**
 - **Benefits include zero DevOps, pay-per-use pricing, and automatic scaling.**
 - **Trade-off:** Rate limits (QPM/TPM) and less control over infrastructure.
- **When to self-host (on the cloud)**
 - **Choose self-hosting when you need full control, data privacy, or custom optimizations.**
 - **Use cases:**
 - Fine-tuned or custom models
 - High-throughput production systems
 - Compliance-sensitive environments
 - **You take on responsibility for scaling, monitoring, and reliability.**
 - **Best for teams with DevOps or MLOps capabilities.**
- **vLLM**
 - **vLLM is an open-source engine for fast, efficient LLM serving.**
 - **Key features:**
 - **PagedAttention:** Reduces memory waste and increases throughput
 - **Continuous batching:** Handles multiple requests efficiently
 - **OpenAI-compatible API:** Easy integration with existing tools
 - **Ideal for local testing and production deployment.**
- **Deployment options**
 - **Model Studio (Managed):**
 - No deployment needed
 - Simple API access
 - Great for Alibaba Cloud's Tongyi models
 - **Local vLLM:**
 - Run on your machine
 - Fast setup for testing
 - Not scalable
 - **Cloud Deployment (PAI-EAS, ECS, ACK):**
 - Elastic, high-concurrency inference
 - Full control over images, scaling, and networking
 - Suitable for enterprise-grade applications

- **Choosing the Right Path**

- **Start with managed APIs** to validate your app and avoid infrastructure complexity.
- **Move to self-hosting** when you need performance, control, or privacy.
- **Use vLLM** for high-throughput, low-latency serving with minimal overhead.
- **Test on smaller models first** (e.g., 1.5B parameters) before scaling up.
- **Monitor GPU utilization, latency, and request errors in production.**