ChinaRxiv

English-access research papers from ChinaXiv

Large-Model Inference Optimization Technologies for Token Operations

2026-06-18

ChinaRxiv

View the original and related papers at
https://chinarxiv.org/items/chinaxiv-202606.00238

Source: ChinaXiv. Machine translation. Verify with the original.

Abstract

Large model inference optimization is a critical foundation for supporting the large-scale, low-cost, and highly stable operation of large model services. Focusing on model inference optimization technologies for token-oriented operations, this paper for the first time proposes a four-layer inference technology architecture of "multi-model fusion—model optimization—model-computing fusion—computing-network-model fusion." It systematically reviews the key technologies and industry status at these four levels, analyzes the application value of related technologies in real-world business scenarios, and provides a practical technical path for reducing token production costs, improving token service efficiency, ensuring the stability of token supply, and promoting the evolution of large model services from "callable" to "operable."

Full Text

Large-Model Inference Optimization Technologies for Token Operations

Shiguo Lian1,2*, Kai Wang1,2, Zhaoxiang Liu1,2, Wen Liu1,2, Minjie Hua1,2, Yutong Liu1,2, Jiangze Yan1,2, Xin Wang1,2, Cong Wang1,2, Yilin Zhang1,2, Yi Shen1,2, Jieyun Huang1,2, Fang Zhao1,2, Huanlin Gao1,2, Ping Chen1,2, Xinyu Yang1,2, Kaikai Zhao1,2,3, Yao Zhao4, Xinggang Wang5, Huishuai Zhang6, Dongyan Zhao6, Junping Du7, Tao Chen8, Xiang Gao9, Qinghuai Ma10

1 Unicom Data Intelligence Co., Ltd.; 2 China Unicom Data Science and Artificial Intelligence Research Institute; 3 Tsinghua University; 4 Beijing Jiaotong University; 5 Huazhong University of Science and Technology; 6 Peking University; 7 Beijing University of Posts and Telecommunications; 8 Fudan University; 9 Zhejiang Lab; 10 Hygon Information Technology Co., Ltd.

* Corresponding author (liansg@chinaunicom.cn)

Abstract Large-model inference optimization is a key foundation for supporting large-model services at scale, at low cost, and with highly stable operation. Focusing on token-oriented model inference optimization technologies, this paper proposes for the first time a four-layer inference technology architecture comprising "multi-model fusion—model optimization—model-computing fusion—computing-network-model fusion." It systematically reviews the key technologies and industry status at these four levels, analyzes the application value of the relevant technologies in real business scenarios, and provides a practical and feasible technical pathway for reducing the cost of token production, improving the efficiency of token services, ensuring the stability of token supply, and promoting the evolution of large-model services from "callable" to "operable."

Keywords large model, Token, inference optimization, key-value cache, cache hit, model routing, model quantization, model distillation, linear attention,

MoE, speculative decoding, operator fusion, expert parallelism, dynamic batching, load balancing, memory management

Table of Contents

Introduction

At present, artificial intelligence is accelerating from the stage of technological breakthroughs toward scaled application, becoming an important foundational driving force for the development of new-quality productive forces and the intelligent upgrading of industries. The State Council' s *Opinions on Deeply Implementing the "AI+" Action* clearly states that artificial intelligence should be promoted for broad and deep integration with all sectors and fields of the economy and society, accelerating the formation of new forms of an intelligent economy and intelligent society characterized by human–machine collaboration, cross-boundary integration, and co-creation and sharing. Driven jointly by policy guidance, computing-power supply, and application demand, China' s large-model industry has entered a stage of rapid growth. Tokens, as the smallest information units processed by large models, are measurable, priceable, and tradable, and have become an important unit for measuring the activity of artificial-intelligence services and industrial value. According to statistics from the National Bureau of Statistics, as of March 2026, China' s average daily token calls had exceeded 140 trillion.

With the explosive growth of token calls, the large-model industry is rapidly

entering a stage of scaled token operation. The core capability of token platforms is no longer merely how many models they access or how much computing power they build, but whether they can continuously provide low-cost, low-latency, high-quality, and governable token services under massive requests, high-concurrency access, long-context interaction, multi-model collaboration, and complex agent tasks. However, scaled token operation also brings new systemic challenges. On the one hand, stronger models possess higher capability for handling complex tasks, but their per-token cost, memory footprint, and inference latency are also higher. On the other hand, real business traffic contains large volumes of simple question answering, structured extraction, long-document processing, multi-turn dialogue, and tool-calling requests. If refined scheduling, cache reuse, and inference acceleration mechanisms are lacking, this can easily lead to wasted computing power, increased tail latency, and reduced service stability. As agent applications evolve from being able to converse to being able to make decisions and execute actions, the model inference chain is further lengthened; stages such as prefill, decoding, key–value caching, model routing, tool invocation, and gateway scheduling all affect final service efficiency. Therefore, large-model inference optimization is becoming a key technical support within the token operation system. Its goal is not only to improve the inference speed of a single model, but also to achieve comprehensive optimization across quality, cost, latency, throughput, stability, and security.

This paper focuses on model inference optimization technologies for token operation. It systematically reviews key technologies at four levels: multi-model fusion, model optimization, model–computing fusion, and computing–network–model fusion, and analyzes their application value and implementation paths in real business scenarios. At the level of multi-model fusion, it focuses on model capability boundary quantification, intelligent routing, model cascading, and model integration, addressing the problem of how to match different requests with suitable models. At the level of model optimization, it focuses on low-complexity attention mechanisms, hybrid expert architecture optimization, speculative decoding, model quantization, model distillation, chain-of-thought optimization, memory management, key–value cache compression, and diffusion-model generation-path optimization, so as to reduce the generation cost per token. At the level of model–computing fusion, it focuses on operator fusion, memory-access optimization, acceleration of primitive operators, tuning of engine parameters, and adaptation of model architectures, improving execution efficiency between models and hardware. At the level of computing–network–model fusion, it focuses on multi-machine inference parallelism, dynamic batching, cache reuse, sticky session routing, load balancing, rate limiting and degradation, ensuring stable supply of large-model token services.

Overall, large-model inference optimization for token operation is a foundational engineering effort supporting the scaled deployment of the large-model industry. Only by conducting collaborative optimization of models, computing power, networks, gateways, and business traffic as a unified system can token generation costs be effectively reduced, token service efficiency improved, the stability of

token supply ensured, and large-model services advanced from being callable to being operational.

- **Application**
  - **Multimodal fusion:** model capability boundary quantification; intelligent model routing; model cascading; model ensembling; ···
- **Model**
  - **Model tuning:** low-complexity attention mechanisms; MoE architecture optimization; optimization of diffusion-model generation paths; chain-of-thought optimization; memory management; KV Cache compression; speculative decoding; model quantization; model distillation; ···
- **Compute**
  - **Model fusion:** operator fusion; GPU-memory access optimization; acceleration of basic operators; engine-parameter tuning; model-architecture adaptation; ···
  - **Compute-network-model fusion:** multi-machine inference parallelism; clustered KV Cache scheduling; gateway sticky-session routing; gateway semantic-cache reuse
- **Network**
  - **Compute-network-model fusion:** dynamic batching; high-performance communication libraries; load balancing; rate limiting and degradation; ···

Figure 0-1 Panorama of Optimization Technologies for Large-Model Inference

I. Multi-Model Fusion

Multi-model fusion is a key technical direction in which large-model services move from "a single model responding to all requests" toward "optimal allocation through multi-model collaboration." Its core lies in achieving a systematic balance among quality, cost, and latency through mechanisms such as model-capability assessment, adaptive request distribution, serial verification, and parallel fusion. This chapter focuses on four aspects: quantifying model capability boundaries, intelligent routing, model cascading, and model ensembling. Among them, quantifying model capability boundaries is the prerequisite for fusion decision-making: through standardized evaluation and runtime observation, it delineates the applicable boundaries of each model across dimensions such as knowledge, reasoning, code, and long context, thereby providing a comparable evidentiary basis for subsequent scheduling. Intelligent routing makes one-time distribution decisions at the request entry point according to task features and model profiles, aiming to achieve high cost-effectiveness—small models handling simple requests—or high performance—dispatching tasks to the strongest long-context model. Model cascading adopts a serial structure of "try first, then judge" : a lightweight model answers first, and a verifier dynamically evaluates quality, upgrading to a stronger model only when necessary, thereby balancing

cost and accuracy under scenarios with uncertain request distributions. Model ensembling invokes multiple mutually complementary models in parallel and integrates their respective strengths through ranking, voting, or generative fusion, breaking through the capability ceiling of a single model. The four technologies respectively undertake functions such as capability characterization, request distribution, quality verification, and result fusion, and jointly support fine-grained token operations for MaaS platforms under a multi-model matrix.

(I) Quantifying Model Capability Boundaries

1. Technical Definition Quantifying model capability boundaries refers to quantitatively characterizing the scope of applicability of large models across dimensions such as knowledge, reasoning, mathematics, code, long context, tool use, and multimodality, on the basis of standardized evaluation, scenario-based samples, runtime observation, and statistical analysis. It emphasizes answering "what the model can do, under which conditions it is stable, and on which tasks or input distributions it is prone to fail," rather than describing model capability merely with a single leaderboard ranking or a broad strong/weak judgment. For token operations, the core value of capability-boundary quantification lies in transforming model capability from experiential judgment into a comparable, reproducible, and traceable evidence system; its results can be used as input evidence for model selection, version admission, stress testing of key business scenarios, risk grading, and model-routing strategies.

Model Capability-Boundary Quantification Process

- **Tasks and Data**
  - Public benchmarks
  - Industry task sets
  - Online feedback
- **Evaluation Orchestration**
  - Model integration
  - Batch execution
  - Version traceability
- **Metric Computation**
  - Quality/robustness
  - Cost/latency
  - Safety/calibration
- **Boundary Modeling**
  - Capability profile
  - Confidence interval
  - Failure-cluster analysis
- **Admission and Governance**
  - Model tiering
  - Version admission
  - Continuous monitoring
- **Runtime Monitoring**

  - Quality drift
  - Cost fluctuation
  - Risk triggering
- **Policy Calibration**
  - Threshold updates
  - Metric weights
  - Evaluation-set iteration
- **Sample Feedback**
  - Anomalous samples
  - Failure cases
  - New business scenarios

Figure 1-1 Model capability-boundary quantification process

2. Industry Progress

Research in industry on model capability boundaries is shifting from rankings on individual benchmarks toward systematic evaluation methods. One survey study classifies evaluation tasks into categories such as general capabilities, domain capabilities, and target capabilities, and points out that issues including data contamination, cultural and linguistic bias, and insufficient evaluation in dynamic environments can affect evaluation conclusions [1]. Earlier surveys on large language model evaluation also emphasized that evaluation data, prompt templates, evaluators, statistical significance, and reproducibility conditions can all change judgments about model capabilities [2]. This indicates that a "capability boundary" is not a fixed label, but an observable range jointly determined by task distribution, input constraints, evaluation metrics, and the operating environment.

Industry Progress: Three Types of External Evaluation References

1. **General Evaluation Benchmarks**
   - MMLU / BIG-bench / HELM
   - Standardized benchmark tasks and multidimensional metric systems
2. **Public Commercial Evaluation Platforms**
   - Artificial Analysis
   - Continuously updated model comparisons, intelligence indices, and confidence intervals
3. **Human Preference Evaluation**
   - Arena / Chatbot Arena
   - Anonymous pairwise comparisons, human voting, and open-ended dialogue preferences

Mature Capability-Boundary Quantification System: A Closed Loop of Four Major Components

A. **Task and Data Layer** - Public benchmarks - Industry task sets - Private samples - Online failure samples

B. **Evaluation Orchestration Layer** - Model integration - Prompt configuration - Tool invocation - Batch execution

C. **Metrics and Analysis Layer** - Quality - Stability - Robustness - Safety - Latency cost

D. **Governance Closed-Loop Layer** - Model tiering - Version admission - Risk thresholds - Continuous monitoring

From external evaluation references toward enterprise-level capability-boundary profiling

Figure 1-2 Industry progress and system composition of model capability-boundary quantification

General evaluation benchmarks remain the foundation for quantifying capability boundaries. MMLU measures model knowledge and problem-solving ability through a multidisciplinary multiple-choice question set [3], while BIG-bench uses a larger-scale and more open task collection to observe extrapolation trends in language models' capabilities in reasoning, memory, symbolic operations, and other aspects [4]. HELM represents a more comprehensive evaluation paradigm: it compares models under unified conditions while simultaneously incorporating accuracy, calibration,

robustness, fairness, bias, toxicity, and efficiency [5]. These benchmarks have driven horizontal comparisons across models, but they also expose the limitations of aggregate scoring: the same model may lead on composite indicators yet reveal shortcomings in specific domains such as industry question answering, long-document comprehension, structured output, or safety scenarios.

Public commercial evaluation platforms have further promoted continuous observation of capability boundaries. Artificial Analysis provides continuously updated model evaluations and comparisons for frontier models. Its key evaluation metric, the "Intelligence Index," is composed of multiple public evaluations, covering dimensions such as agentic tasks, coding, general capabilities, and scientific reasoning, while also disclosing methodological information such as evaluation composition, weights, and confidence intervals [6]. The value of such platforms lies in providing continuously traceable horizontal references, enabling faster perception of model-version updates, capability drift, and differences among models from different vendors. Their limitation is that public benchmarks may still be affected by data contamination, question-type bias, and differences in business scenarios, and therefore cannot directly replace scenario evaluation within enterprises.

Human-preference evaluation, represented by Arena, complements traditional automated benchmarks. By using anonymity, pairwise comparison, and human voting to measure open-ended conversational experience, it can cover aspects that are difficult to evaluate through some automatic grading methods, such as expression quality, instruction-following ability, and interaction preferences [7]. Arena's leaderboard extends public arena-style evaluation into dynamic rankings

of capabilities in text, code, images, video, and other modalities, providing an important reference for observing the relative performance of models in real user interactions [8]. However, human-preference evaluation is likewise affected by voter populations, prompt distributions, language environments, and temporal changes. It is therefore more suitable as one class of evidence for depicting capability boundaries, rather than as the sole criterion for judgment.

From the perspective of engineering implementation, a relatively mature capability-boundary quantification system usually includes four types of components. First is the task and data layer, covering public benchmarks, industry task sets, private business samples, red-team samples, and online failure samples. Second is the evaluation orchestration layer, responsible for model access, prompt configuration, tool invocation, multi-round interaction, version recording, and batch execution. Third is the indicators and analysis layer, which calculates metrics such as quality, stability, calibration, robustness, safety, latency, and resource consumption, and identifies failed clusters, boundary samples, and out-of-distribution samples. Fourth is the governance closed-loop layer, which converts evaluation results into model grading, version admission, risk thresholds, regression testing for key scenarios, and continuous monitoring mechanisms. Domestic industrial practice is advancing along a bidirectional drive of "capability–scenario." For example, based on the A-Eval benchmark, China Unicom classifies typical application scenarios into multi-level capability categories and constructs a mapping graph of "model–capability category–capability level–scenario" [9–11]. In the OpenClaw agent scenarios, capability-boundary quantification is further extended to evidence such as real task sets, task metadata, and traceable decision-making, shifting evaluation from general horizontal assessment toward specialized assessment oriented to business chains [12]. This closed loop upgrades capability-boundary quantification from one-off horizontal evaluation into a foundational capability for long-term service-oriented operations.

(II) Intelligent Routing

1. Technical Definition

Figure 1-3 content

Two orthogonal and stackable technical main lines

- **Cost-effective routing (main line 1)**
  Reduce cost as much as possible under quality constraints.
  - Preferentially use low-cost models
  - Dynamically control invocation depth
  - Cache / reuse / compress
-
- **High-performance routing (main line 2)**
  Under a cost budget, distribute requests to the models best suited to them in order to improve overall quality.

– Capability-matched distribution
– Difficulty adaptation
– Expert models first

Figure 1-3 Cost-effective and high-performance routing

Intelligent routing is a core capability of a MaaS platform for collaborative scheduling among multiple models. It means that after a request enters the inference gateway, a lightweight decision-making module comprehensively evaluates the request characteristics, the model capability profile, and the cost-quality requirements, and then automatically selects the most appropriate model backend to complete inference. The motivation for this problem lies in the tension between cost and quality in single large-model services: the primary model delivers higher quality on complex tasks but is significantly more expensive, whereas real traffic contains a large number of low-complexity requests such as short Q&A, format conversion, and structured extraction, which can meet expectations when handled by small-parameter models. If all requests are sent to the primary model, unnecessary token costs and latency pressure are incurred. By objective, intelligent routing can be divided into two mutually orthogonal technical main lines that can also be deployed in a stacked manner. The first is cost-effective routing, which reduces inference cost as much as possible while satisfying quality constraints; a typical form is a binary strong-weak model routing scheme in which a small-parameter auxiliary model handles requests judged to be simple, while the primary model serves as the quality baseline and fallback. The second is high-performance routing, which, under a given cost budget, distributes requests to the models best suited to them according to task type, domain characteristics, and modality characteristics, thereby maximizing overall output quality and breaking through the capability ceiling of a single model.

2. Industry Progress

Figure 1-4 content

- **2023 –Early exploration and prototype stage**
  Routing prototypes emerged; feasibility was verified.
- **2024 –Method maturity and capability improvement stage**
  Algorithm systems became richer; performance continued to improve.
- **2025 –Engineering implementation and ecosystem expansion stage**
  Productization accelerated; standardization and the ecosystem improved.

Research prototype → Method optimization and evaluation → Engineering implementation → Standardized API / MaaS gateway capabilities

Inputs and context for routing

- **User requests**

  - Text / dialogue
  - Task instructions
  - Scenario / domain
  - Cost / latency requirements
- **System signals**
  - Model capability profiles
  - Real-time load / price
  - SLA / budget constraints
  - Historical quality feedback
- **Routing objectives**
  - Better quality
  - Lower cost
  - Lower latency
  - Higher stability

Three major methodological families (representative solutions and key capabilities)

1. **Routing based on semantic retrieval and rules**
   - OrchestraLLM
   - Semantic Router
   - LiteLLM Auto-Router
   - ...
   - **Core advantages**
     - Low cold-start cost
     - Strong interpretability
     - Small inference overhead
   - **Main limitation**
     - Relies on semantic quality; limited generalization
2. **Routing based on learned discriminators / preferences**
   - Hybrid LLM
   - RouteLLM
   - Benchmark Routing
   - RouterDC
   - ...
   - **Key capabilities**
     - Quality-difference prediction
     - Training on preference data
     - Cost–quality threshold adjustment
     - Contrastive learning
   - **Main limitation**
     - Training and annotation costs are relatively high; continuous feedback loops are required
3. **High-performance routing for multi-model complementarity**
   - Fusion of Experts
   - ...
   - **Core value**

  - Distribute requests to the most suitable expert models
  - Raise the capability ceiling
  - Support parallel / cascaded / dynamic composition
- **Main limitation**
  - Higher system complexity; cost control is more challenging

Evaluation dimensions (key metrics for routing systems)

- Quality improvement (relative to baseline)
- Cost savings (relative to baseline)
- Latency reduction (P95/P99)
- Stability (SLA attainment rate)
- Generalization capability (cross-domain / cross-expression)
- Interpretability (traceability)

Industrial productization and ecosystem (representative products)

- **Amazon Bedrock Intelligent Prompt Routing**
  Cost–quality trade-off; managed routing service
- **Martian**
  Performance first; intelligent model selection
- **Not Diamond**
  Cost first; cost-effective routing
- **OpenRouter**
  Unified API; multi-model gateway
- **LiteLLM**
  Unified API; routing and governance
- **GPT-5 built-in real-time routing**
  Real-time routing; automatic model adaptation
- **Alibaba Cloud PAI**
  Enterprise-grade gateway; controllable cost
- **Volcano Ark**
  Multi-model access; balance between performance and cost

Unified access · Policy orchestration · Monitoring and observability · Billing and settlement

Overall trends

1. **Cost-effective routing is the most mature: saving money without reducing quality**
   - Semantic retrieval + rules: low engineering cost
   - Learning-based routers: stronger quality–cost trade-off
   - Stable and controllable deployment: scaled implementation across industries

2. **High-performance routing and multi-model complementarity are becoming the focus of the next stage: routing is deeply integrating with foundation models and Agent frameworks**
    - Expert-model collaboration: the upper limit continues to rise
    - Routing and Agent planning: deep linkage
    - Dynamic composition and tool invocation: forming system-level capabilities
    - From point optimization toward end-to-end intelligent scheduling

Continuously moving toward higher quality, lower cost, and stronger generalization.

Figure 1-4 Industry progress in intelligent routing

Focusing on intelligent routing, since 2023 academia and industry have developed a relatively complete methodological spectrum. Related work can broadly be grouped into three categories: routing based on semantic retrieval and rules, routing based on learning-type discriminators, and high-performance routing oriented toward multi-model complementarity. On this basis, intelligent routing capabilities have also gradually been productized by cloud vendors, model platforms, and independent start-ups, becoming one of the important capabilities of large-model API gateways and MaaS platforms.

The first category is routing based on semantic retrieval and rules. Such methods usually transform the question of "whether to call a small model" or "which expert model should be called" into a similarity-matching problem in vector space. For dialogue-state tracking tasks, OrchestralLLM separately constructs a small-model expert library and a large-model expert library, and, based on the assumption that "semantically similar samples have similar difficulty," dynamically decides which expert model to invoke through nearest-neighbor retrieval [13]. The open-source project Semantic Router abstracts similar ideas into a general-purpose routing tool: developers maintain several routes and their corresponding utterances, and the system completes millisecond-level routing decisions through the cosine similarity between request embeddings and routes [14]; it has already been integrated by gateways such as LiteLLM as an optional automatic routing component [15]. The advantages of this line of work are low cold-start cost, strong interpretability, and low inference overhead; its limitation is that performance depends heavily on the coverage of the utterance library, and generalization is limited when facing new requests with large semantic distances.

The second category is routing based on learning-type discriminators or preferences. This line of work models routing as a classification, regression, or ranking problem; supervisory signals may come from manually annotated data, Judge-LLM evaluation results, human preference data, or public evaluation benchmarks. Hybrid LLM trains a query router to predict the quality difference between small models and large models for a query, and allows users to dynamically balance cost and quality through thresholds; the paper reports that,

without reducing response quality, large-model calls can be reduced by up to 40% [16]. RouteLLM uses human preference data from Chatbot Arena to train routers, compares implementations such as similarity-weighted Elo, matrix decomposition, BERT classifiers, and Causal-LLM classifiers, and releases them as an open-source framework; the authors report that in some scenarios it can reduce cost by more than a factor of two without sacrificing response quality [17]. Schnitzer et al. further formulate "selecting the best LLM for a new task" as a set of binary classification tasks, proposing a method that trains routers only from existing public evaluation benchmarks and avoids re-annotating the target task [18]. RouterDC addresses the difficulty traditional routers have in converging when multiple models are simultaneously suitable for a query, introducing a contrastive-learning objective and achieving results superior to existing methods on both in-distribution and out-of-distribution test sets [19].

The third category is high-performance routing oriented toward multi-model complementarity. When the goal shifts from "saving money" to "raising the upper bound of capability," the research focus of routing changes from "whether to downgrade" to "which most capable expert model to dispatch to." Fusion of Experts formalizes this problem by organizing multiple models with complementary domain expertise into an expert pool; a learned fusion module selects an expert for each request or weights their outputs, enabling overall performance on cross-domain mixed distributions to surpass that of any single expert [20]. The same idea is also applicable to deployment in routing form: instead of actually invoking all experts in parallel, the request is assigned only to the expert whose profile best matches it.

At the industrial product level, intelligent routing has gradually evolved from research prototypes into standardized API and gateway capabilities. Amazon Bedrock Intelligent Prompt Routing supports request-level routing within the same model family according to predicted response quality; according to official claims, it can reduce cost by up to 30% without sacrificing accuracy [21]. Martian Model Router provides ready-to-use replacement capability through an OpenAI-compatible interface; it estimates the performance of different models on a specific request through predictive model mapping, and exposes explicit cost–quality trade-off parameters [22]. Not Diamond is positioned as a model router for agents, emphasizing customized router training based on customer evaluation data; it can achieve more than 50% cost savings and more than 10% accuracy improvement [23]. OpenRouter, as a model aggregation layer, provides unified API and billing capabilities, has connected more than 400 active models from over 60 providers, and performs backend routing based on cost, speed, and availability [24]. LiteLLM is more oriented toward enterprise self-hosted gateways, providing capabilities such as a unified OpenAI protocol, virtual keys, quota management, observability, retries, fallbacks, and load balancing, and has introduced 由

semantic-routing-driven automatic routing option [15].

Intelligent routing has also begun to enter the systems of leading model plat-

forms and cloud vendors. In GPT-5, released by OpenAI in August 2025, a built-in real-time router was introduced, combining a fast general-purpose model with a deep reasoning model into a unified system. The router dynamically selects which submodel to invoke based on conversation type, complexity, tool requirements, and user intent, and continuously trains on real signals such as users' model-switching behavior, response preference rates, and correctness measurements [25]. Domestic cloud vendors have also launched related capabilities: Alibaba Cloud PAI' s "LLM Intelligent Routing" is mainly oriented toward load balancing and scheduling among multiple reasoning instances, with optimization based on indicators such as prompt length and generation length [26]; Volcano Engine Ark' s "Intelligent Model Routing," by contrast, belongs to the request-level model-selection paradigm, dynamically matching the most suitable model according to the prompt of each request, and supporting two strategies: "cost first" and "performance first" [27].

China Unicom has built intelligent routing capabilities based on the Yuanjing MaaS platform, forming a complete closed loop of "model foundation—multidimensional evaluation—capability profiling—routing training—gateway deployment—effectiveness assessment." The platform integrates a matrix of self-developed and external multimodal models, builds evaluation benchmarks around scenarios such as knowledge question answering, reasoning, coding, mathematics, and tool invocation, and forms capability profiles of models across the dimensions of quality, speed, cost, and stability. Cost-effective routing has evolved from rules and semantic matching to intelligent scheduling based on difficulty discrimination: low-complexity requests are diverted to auxiliary models, while requests that are difficult to judge or anomalous automatically fall back to the main model; for high-stability scenarios such as agents and tool invocation, a conservative strategy that prioritizes the main model is adopted. The system uses a pluggable proxy gateway to uniformly handle request distribution and logging, and relies on offline evaluation and data feedback to form an iterative closed loop. Pilot measurements show that auxiliary models can maintain performance close to that of the main model in natural-language question answering and some tool-invocation scenarios, reducing inference costs without significant loss of quality. In the future, the system will be upgraded from binary downgrading to fine-grained scheduling oriented toward multiple models and multiple scenarios, and will be extended to complex agents, multi-turn tasks, and multimodal scenarios.

Overall, current intelligent routing in the industry shows two clear trends. First, cost-effective routing remains the most mature and mainstream direction for deployment. Industrial products generally take "saving money without reducing quality" as their core selling point, corresponding academically to the routes of binary routing between strong and weak models and learning-based discriminators. Second, high-performance routing and multi-model complementarity are becoming key directions for the next stage. The GPT-5 real-time router evolves routing from a side-path gateway decision into an endogenous module of the model system, indicating that intelligent routing capabilities will be further and

deeply integrated with foundation models themselves and agent frameworks.

(III) Model Cascading

1. Technical Definition

Figure content

Scenario-Based Objective (1): Cascading for Cost Savings
Allow as many requests as possible to be completed by the front-end small model, upgrading only a small number of difficult requests, thereby reducing overall costs.

Applicable scenarios: knowledge question answering; text classification; information extraction; intent recognition; sentiment analysis; ···

Evaluation focus: pass-through rate improvement | average cost per request reduction | response speed improvement

Scenario-Based Objective (2): Cascading for Quality Improvement
The small model first attempts to generate reasoning paths and drafts, while the strong model makes the final decision, ensuring high correctness and stability.

Applicable scenarios: mathematical reasoning; code generation; tool invocation; complex planning; compliance review; ···

Evaluation focus: accuracy improvement | severe error rate reduction | final-result reliability improvement

Figure 1-5 Scenario-Based Objectives of Model Cascading

Model cascading means that the MaaS platform, in ascending order of cost, submits a request in sequence to multiple candidate models for attempted processing, and at each step uses a verifier or scoring function to determine whether the current answer has reached an acceptable quality threshold; if it has, the process terminates immediately and returns the answer; otherwise, it escalates to a stronger and more expensive model,

until a final answer is given. The motivation for this problem is that static difficulty prediction in intelligent routing is insufficiently reliable in scenarios such as complex reasoning, code generation, and domain-specific question answering, where it is hard to judge the difficulty in advance solely from the request text. Cascading, through the serial structure of "try first, then judge," transforms difficulty assessment from static prediction on the request side into dynamic evaluation of the model' s actual output, thereby achieving a balance between cost and quality over a broader request distribution. According to the objective, cascading can likewise be divided into two main lines: one is cost-saving-oriented cascading, which lets as many requests as possible stop at small models near the front of the chain, and is mainly suitable for tasks such as knowledge question answering and text classification where a one-time answer can measure quality; the other is quality-improvement-oriented cascading, which lets small models

try first and strong models make the final decision, so that overall accuracy on difficult problems exceeds that of any single model, and is mainly suitable for tasks with high requirements for answer correctness, such as mathematical reasoning, code generation, and tool use.

2. Industry Progress

Visible text in Figure 1-6:

- **2023 –1 FrugalGPT**
  Foundational work
  - Invoke candidate models in order of cost
  - Lightweight quality scoring function
  - Stop early when the threshold is reached
  - Budget-parameter optimization: minimize cost while satisfying quality
  - Equal-quality cost saving: reduce cost under quality constraints
  - Equal-cost quality improvement: improve quality under a fixed budget
  - Applicable scenarios: general Q&A / summarization / creation and other natural-language tasks
- **2023 –2 Mixture of Thought Cascade**
  - Use a weak model for multiple samples and use answer-consistency statistics as confidence
  - Combine Chain-of-Thought with Program-of-Thought
  - Escalate difficult questions to a stronger model
  - Weak model: multiple samples
  - Consistency (quality signal)
  - Escalate? No: output directly; Yes: escalate to a strong model
  - Applicable scenarios: mathematical reasoning / symbolic reasoning / proofs and other tasks requiring long-chain reasoning
- **2024 –3 AutoMix**
  - A small model generates an answer and outputs a self-verification probability (correctness / confidence)

  - Fuse multiple signals without a verifier to decide whether to escalate
  - Automatic mixing reduces manual threshold tuning
  - Small model: generate answer + self-verification probability, $p = 0.37$
  - Verifier-free (feature fusion)
  - Low confidence / uncertainty: escalate
  - High confidence: accept
  - Applicable scenarios: general Q&A / knowledge retrieval / multi-domain tasks
- **2024 –4 Tabi**
  - Multi-level reasoning system with stepwise evaluation
  - Calibrated confidence (speed / historical experience)
  - Escalate to a stronger LLM when confidence is low
  - Emphasizes engineering feasibility and latency control
  - Lightweight model (L0)
  - Medium model (L1)
  - Strong model (L2 / LLM)
  - Confidence calibration
  - Applicable scenarios: enterprise Q&A / content generation / multi-turn dialogue
- **2024 –5 EcoAssistant**
  - Assistant hierarchy + code executor
  - Iterative testing (execute → feedback → revise)
  - Escalate to a stronger assistant/model upon failure or high cost
  - Suitable for code-driven Q&A and Agents
  - Lightweight assistant (planning / retrieval)

  - Code executor (run / test)
  - Failure / cost overrun: escalate to a stronger assistant/model
  - Iterative testing (multiple rounds)
  - Applicable scenarios: code generation / debugging / data analysis / complex Agent workflows
- **2024-2025+ —6 Speculative Decoding**
  - Extend strong-model use to the token level
  - Draft (small model) generates candidate tokens
  - Target (large model) verifies and accepts them
  - Significantly reduces generation latency and cost
  - Draft model: generate candidate tokens
  - Target model: verify
  - Accept; discard; continue generation
  - Applicable scenarios: large-model generation / real-time dialogue / long-text generation

Figure 1-6 Development of Model Cascading

Model cascading is one of the most systematic research directions in the field of large-model cost optimization. Since 2023, it has formed a clear methodological spectrum and has gradually expanded toward complex tasks involving agents, reasoning, and tool use.

FrugalGPT is a foundational work in this direction. Chen, Zaharia, and Zou proposed the idea of an LLM Cascade: candidate models are called in order from low cost to high cost, and an independently trained lightweight scoring function assigns a reliability score between 0 and 1 to the answer at each step. If the threshold is reached, the process stops early; otherwise, the next model is queried. The authors formulate the selection of candidate model chains and thresholds as an optimization problem with budget constraints, and report that on multiple datasets FrugalGPT can reduce cost by up to 98% while matching the performance of strong models such as GPT-4, or improve accuracy by 4 percentage points at the same cost [28]. This work clearly proposed two typical usage patterns— “saving substantial money at equal quality” and “improving quality at equal cost”—which remain the most frequently cited design paradigms in industrial implementations of model cascading.

Mixture of Thought Cascade extends the cascading idea to reasoning tasks. Yue

et al. observed that relying only on the reliability score of a single answer is unstable in complex mathematical and symbolic reasoning. They therefore proposed using the consistency of weak-model answers as a difficulty signal: for the same problem, the weak model samples multiple reasoning paths; the consistency of the answers is statistically measured; and the two reasoning representations, Chain-of-Thought and Program-of-Thought, are combined for voting or verification. Only problems whose consistency is below a threshold are escalated to a stronger model. Experiments on six mathematical and symbolic reasoning datasets demonstrated that this cascade can achieve accuracy close to using GPT-4 alone, while consuming only about 40% of the cost [29]. The significance of this work lies in extending the verifier from an "independent small model" to "statistical judgment over the weak model' s own outputs," thereby avoiding the cost of training an additional verifier and making it more suitable for deployment in reasoning tasks.

AutoMix further combines cascading with few-shot self-verification. Madaan et al. proposed that a small model generate an answer and a self-verification probability, and then a meta-verifier based on a partially observable Markov decision process decides whether to escalate the problem to a stronger model. The paper reported, across multiple language models and datasets,

reported that computational cost can be reduced by more than 50% while maintaining comparable performance [30]. At the system level, HKUST' s multilevel hierarchical inference scheduling system, designed for discriminative NLP services, uses calibrated confidence to first let small models rapidly return low-risk requests; low-confidence requests are then escalated to an LLM. Attention pruning and weighted ensembling are introduced along the escalation path to compensate for system overhead, demonstrating from an engineering perspective the feasibility of cascade architectures in production inference services [31].

Model cascading has also been validated on complex agent tasks involving tool use and code generation. EcoAssistant introduces the cascading idea into code-driven question-answering agents, constructing a "from cheap to expensive" hierarchy of assistants: the cheapest assistant first interacts with a code executor to iteratively generate and debug code; only when the current assistant dialogue cannot solve the problem is the task escalated to a more expensive assistant, and historically successful question-answer pairs are fed back as examples to the low-cost assistant. The authors report that, on code-driven QA tasks such as weather, stocks, and locations, this scheme achieves a success rate more than 10 percentage points higher than a GPT-4 assistant at less than half the cost of GPT-4 [32]. This result is particularly instructive for agent workloads with intensive tool use: it shows that, in complex agent-oriented tasks, cascading not only saves money but may also be more robust than a single direct call to a strong model because of its iterative debugging mechanism.

In recent years, model cascading has further been integrated with inference-acceleration techniques at the system level. Speculative decoding can be regarded as the ultimate expression of the cascading idea at the token level: a

small draft model generates multiple candidate tokens in parallel, and the target model then verifies in a single forward pass which tokens can be accepted, thereby significantly improving decoding throughput while ensuring that the output distribution remains unchanged [33]. Mainstream inference frameworks such as vLLM and SGLang have already made speculative decoding one of their default capabilities, extending cascading mechanisms from the "request level" to the "token level" and further reducing the cost per token.

At the industrial product level, cascading and fallback capabilities have become standard features of mainstream LLM gateways. Databricks Unity AI Gateway provides a fallback mechanism that can automatically fall back to backup models in a preset order when the primary model returns rate limiting or 5xx errors, while recording all calls in unified usage and load logs [34]. OpenRouter also provides underlying automatic fallback capabilities based on cost, speed, and availability [24]. Most of these product-level solutions take "failure fallback" as their entry point, but their underlying invocation pattern is consistent with score-based discriminative cascading and can serve as an engineering baseline for cost-saving cascades.

On China Unicom' s Yuanjing MaaS platform, intelligent routing is the main implementation form for multi-model collaboration, and an evolution toward model cascading is being planned in parallel: routing addresses "predicting difficulty in advance," while cascading addresses "evaluating results during execution" ; together, they cover requests with different quality sensitivities and cost budgets. At the model level, self-developed language models form a multi-parameter tier, which, together with external strong models, can flexibly organize a cascade structure of "lightweight self-developed model → medium-scale self-developed model → primary model or external strong model." At the evaluation level, the multidimensional evaluation benchmarks and offline evaluation processes established around intelligent routing can simultaneously train request-level difficulty classifiers and answer-level reliability scorers, reusing the existing data loop. At the system level, a pluggable agent gateway supports the natural extension of a single decision into multiple serial calls, and records each model level' s answer, cost, latency, and final source through unified logs and an observability system, making the cascading effect quantifiable and auditable. Future efforts will focus on three types of scenarios: first, consistency-verifiable tasks such as mathematical reasoning and code generation, drawing on Mixture of Thought Cascade and AutoMix and using the consistency of weak-model answers as a difficulty signal; second, agent and tool-use-intensive tasks, drawing on EcoAssistant and using tool execution results as a cascade scoring signal; third, high-concurrency dialogue and discriminative tasks, combining cascading with speculative decoding to extend cost optimization from the request level to the token level. This will form, together with intelligent routing, a two-layer cost-optimization mechanism of "advance prediction + in-process verification."

Overall, research and industrial practice on model cascading show two clear evolutionary paths. First, verification signals are gradually moving from "inde-

pendently trained scoring functions" toward "the consistency, self-verification, and execution feedback of model outputs themselves,"making cascading easier to deploy on new tasks that lack annotations; second, the granularity of cascading is moving from the request

···extending from the level of sequences to the level of tokens, and from single-turn question answering to complex agent workflows, thereby deeply integrating cascading with inference acceleration and agent orchestration.

(IV) Model Integration

1. Technical Definition

Figure 1-7 Three Forms of Integration

Three Typical Forms of Integration

1. **Self-Integration of the Same Model**
   The same model is invoked multiple times under different sampling or reasoning paths.

   - Same request
   - Model A (sampling 1 / path 1)
   - Model A (sampling 2 / path 2)
   - Model A (sampling 3 / path 3)
   - ···
   - Model A (sampling K / path K)
   - Majority voting or consistency
   - Final answer

   Feature: Reduces variance caused by randomness and improves stability and consistency.

2. **Heterogeneous Model Integration**
   Models with different architectures, different training data, or different specialties produce outputs in parallel.

   - Same request
   - Model (mathematics specialist)
   - Model (coding specialist)
   - Model (general-purpose large model)
   - Model (retrieval-augmented)
   - ···
   - Ranker or fusion module
   - Final answer

   Feature: Leverages complementary advantages among models and covers a broader capability space.

3. **Multi-Agent Collaboration**
   Multiple models assume different roles and collaborate layer by layer to

complete the task.

- Same request
- Proposer (generates a solution)
- Critic (identifies problems)
- Improver (optimizes the solution)
- Aggregator (integrates conclusions)
- ···
- Final answer

Feature: Improves performance on complex tasks through role division and iterative feedback.

Model integration refers to a MaaS platform invoking multiple candidate models in parallel for the same request, after which a ranker, aggregator, or fusion module evaluates, weights, or generatively merges multiple candidate outputs, ultimately producing an answer that combines the strengths of multiple models. Through statistical averaging, voting, or learned fusion, it reduces the variance and bias of a single model, enabling the overall output to surpass the standalone level of any participating model. The motivation lies in the fact that, for complex tasks, a single model may have capability ceilings and domain blind spots. Different models exhibit markedly different distributions of strengths across task subsets such as mathematical reasoning, code generation, and long-form writing; therefore, parallel invocation and explicit fusion mechanisms are needed to draw on each model' s strengths and compensate for its weaknesses. Integration mainly serves high-performance scenarios and can be divided into three typical forms: first, self-integration of the same model, in which the same model is invoked multiple times using different sampling or reasoning paths, and the most stable answer is selected through majority voting or self-consistency, in order to reduce stochastic errors and improve accuracy in reasoning tasks; second, heterogeneous model integration, in which different models with complementary architectures and training data are invoked in parallel, and their outputs are integrated by a ranker or fusion module, so as to combine the domain strengths of different models; third, multi-agent collaboration, in which multiple models are organized into an agent structure, with front-layer models generating candidates and back-layer models aggregating and refining them, for use in complex creation, debate, and decision-making tasks.

2. Industry Progress

Text in Figure 1-8

I. **Same-Model Self-Ensembling**

- **Self-Consistency**
  - Multiple sampling
  - Multiple reasoning paths
  - Aggregation and selection

    - Majority voting
    - Similarity scoring
  - Input / question → Sample 1, Sample 2, ···, Sample N → Final output
- **More Agents Is All You Need**
  - Multiple homogeneous agents generate in parallel
  - Input / question → Agent 1, Agent 2, ···, Agent K
  - Aggregation strategy
    - Majority voting
    - Similarity scoring
  - Performance improves as the number of agents increases
- **Key mechanisms:** multiple sampling; multi-path exploration; voting / similarity fusion; clear gains from scaling

II. **Heterogeneous-Model Integration**

- **LLM-Blender**
  (PairRanker + GenFuser)
  - Model A, Model B, ···, Model N
  - Pairwise comparison and ranking
    - Pairwise comparison
    - Ranking and scoring
  - Generative fusion
    - GenFuser
    - Integrated generation
  - Fused output
- **Fusion of Experts**
  - Candidate model pool
  - Expert selection / weighting
    - Gating selection
    - Weight allocation
  - Weighted fused output
- **FuseLLM / Knowledge Fusion**
  - Base model
  - External models / knowledge
  - Parameter-level knowledge fusion
    - Weight fusion
    - Knowledge alignment
  - Enhanced model
- **Key mechanisms:** pairwise ranking; generative fusion; expert selection / weighting; parameter-level fusion

III. **Multi-Agent Collaboration**

- **Mixture-of-Agents**
  - Multi-layer agent architecture
  - High-level planning agent *(routing / decomposition)*
  - Low-level execution agents *(generation / retrieval / calculation ···)*

- **Collaboration workflow**
  - Problem input
  - Front-layer proposals
    - Multiple agents propose solutions in parallel
  - Back-layer aggregation
    - Evaluation / deduplication / selection
  - Refined output
    - Integrated optimization / self-reflection
- Open-source models alone can also approach or surpass closed-source strong models
- **Key mechanisms:** multi-layer collaboration; division of labor and complementarity; aggregation and refinement; open and optional

Figure 1-8 Three major industry approaches to model integration

Around model integration, academia and industry have already formed a complete methodological spectrum encompassing same-model self-ensembling, heterogeneous-model integration, and multi-agent collaboration, and significant quality improvements have been verified on tasks such as mathematical reasoning, code generation, and open-ended writing.

The representative work on same-model self-ensembling is Self-Consistency. On the basis of Chain-of-Thought prompting, Wang et al. proposed the Self-Consistency method: for the same problem, the model generates multiple reasoning paths under temperature-based sampling, and then majority voting is performed over the final answers, replacing single greedy decoding with marginalization over multiple reasoning paths, thereby significantly improving accuracy on mathematical and symbolic reasoning tasks [35]. This method is one of the most influential implementations of model integration in the era of large models, and remains a common baseline for various reasoning evaluations. The work "More Agents Is All You Need" by the Tencent team further extends the idea of self-consistency into a more general sampling-voting framework: multiple independent agents are instantiated from the same LLM, each answers the same question, and the answers are then aggregated by majority voting or similarity scoring. The paper reports that the method' s performance improves steadily as the number of agents increases, and that it is orthogonal to existing prompt engineering, CoT, debate frameworks, and so on, allowing them to be stacked [36]. Such methods have controllable costs, are easy to implement engineering-wise, and are the most direct form of same-model self-ensembling to deploy on MaaS platforms.

The representative work on heterogeneous-model integration is LLM-Blender. Jiang, Ren, and Lin proposed using PairRanker to perform pairwise comparisons among multiple candidate LLM outputs, followed by GenFuser to generate a fused output. The authors report that its overall performance is significantly better than any single model and multiple baseline methods, and they

constructed the MixInstruct dataset to support ensemble training based on pairwise comparisons [37]. Fusion of Experts formalizes a similar problem: multiple expert models with complementary domain strengths are combined, and a supervised-learning-trained fusion module selects a single expert or weights the outputs of multiple experts for each request. It proves that FoE outperforms any single expert under cross-domain mixed distributions and explicitly controls the number of expert calls under budget-constrained settings [20]. Knowledge Fusion, by contrast, advances fusion from the output level to the parameter level. FuseLLM distills knowledge from the generation distributions of source models, integrating the capabilities of multiple open-source LLMs with different architectures and training data into a target LLM. It verifies on inference, commonsense, and code-generation tasks that the fused model outperforms any individual source model [38]. These works provide a complete methodology for heterogeneous-model integration, ranging from shallow output fusion to deep parameter fusion.

The representative work on multi-agent collaboration is Mixture-of-Agents. Wang et al. proposed organizing multiple open-source LLMs into a multi-layer agent architecture: front-layer models propose multiple candidate answers, and back-layer models aggregate and refine the outputs from the previous layer, progressively improving answer quality. The paper reports on benchmarks such as AlpacaEval 2.0 that MoA, using only open-source models, achieves results surpassing GPT-4 Omni [39]. This result implies that, through reasonable parallel invocation and layer-by-layer aggregation, ensembles of heterogeneous open-source models are entirely capable of approaching or even

···and even surpass the capability ceiling of a single strong closed-source model, thereby providing an important reference for high-performance ensemble routes.

The idea of model ensembling also appears in a more fine-grained form in the field of inference acceleration. Speculative decoding uses both a draft model and a verification model at every step of the decoding process; the verification model validates multiple candidate tokens from the draft model in parallel. In a sense, this is a lightweight ensemble at the token level, and it can significantly improve throughput while ensuring that the output distribution remains unchanged [33]. This idea is philosophically consistent with request-level ensembling: both invoke multiple models simultaneously and aggregate outputs through a trusted verification mechanism in exchange for improvements in quality or efficiency.

At the level of industrial products, because model ensembling naturally increases costs, it currently appears more often as an optional capability or in forms targeted at specific high-value scenarios. Databricks Unity AI Gateway provides fallback orchestration capabilities, while also supporting multi-model A/B testing and gradual traffic shifting, laying the foundation for ensemble forms based on parallel verification by multiple models [34]. Mechanisms such as multi-model review and multi-model voting for agent workflows have also begun to appear in enterprise-level agent platforms. Overall, the pace at which industry is implementing ensembling is clearly slower than that of routing and cascading. On

the one hand, ensembling shows stable performance on verifiable tasks such as mathematical reasoning and code generation, but it is difficult to find a general-purpose, low-cost fusion mechanism for open-ended generation. On the other hand, the cost structure of ensembling makes it suitable only for high-value tasks, and it must work together with routing capabilities to determine "when ensembling is worthwhile."

On the Yuanjing MaaS platform, model ensembling is positioned as a core means for high-performance scenarios, with the goal of accurately scheduling complex tasks to the strongest models, integrating the complementary strengths of multiple models, and breaking through the capability ceiling of a single model. At the model level, the Yuanjing multi-model symbiosis matrix covers a self-developed language-model hierarchy ranging from lightweight models to main models, as well as multimodal models and external strong models. Combined with capability profiles produced by multidimensional evaluation, ensemble combinations can be organized on the basis of complementarity rather than similarity. At the evaluation level, the two-stage offline evaluation process established around intelligent routing also serves ensembling: first-stage labels are used to train rankers and fusion models, while second-stage evaluation is extended to ensemble outputs, making ensembles and single models comparable and observable under the same metric system. At the system level, a pluggable proxy gateway uniformly undertakes authentication, parallel invocation, logging, and timeout control, and records candidate outputs, weights, and final results in the fusion stage; streaming and asynchronous capabilities support long-form generation and multi-turn scenarios. Future priorities focus on three types of directions: first, verifiable tasks such as mathematical reasoning and code generation, introducing Self-Consistency sampling–voting; second, long-text and cross-modal creation, in which a fusion model integrates the strengths of multiple models; and third, complex agents and multi-step decision-making, adopting the proposal–critique–aggregation hierarchical structure of Mixture-of-Agents. Considering that ensembling naturally brings a several-fold increase in inference cost, the platform does not enable ensembling for all requests. Instead, intelligent routing first filters high-value requests before handing them over to the ensemble, ultimately forming a three-layer collaborative system in which "routing identifies value, cascading controls depth, and ensembling breaks through the ceiling."

Overall, model ensembling is the route within the multi-model collaboration paradigm that is most capable of breaking through the capability ceiling of a single model, but it is also the most computationally expensive. Its reasonable form within a MaaS platform is to combine it with routing and cascading: routing determines which requests are worth enabling ensembling for, cascading controls the depth and cost of the ensemble, and ultimately, for high-value complex tasks, a controllable cost premium is exchanged for output quality that is significantly higher than that of a single model.

II. Model Optimization

Model optimization is a core technical direction for reducing the unit token-generation cost of large-model inference and improving generation efficiency. Its essence lies in carrying out systematic optimization around model-structure characteristics, inference-process control, compression of intermediate states, and reduction of computational precision, so as to generate outputs of equal or higher quality with less computational overhead. This chapter focuses on analysis from nine aspects: optimization of low-complexity attention mechanisms, optimization of MoE architectures, optimization of diffusion-model generation paths, chain-of-thought optimization, memory management, key–value cache compression, speculative decoding, model quantization, and model distillation. Among them, low-complexity attention mechanisms reduce the computational, caching, and memory-access complexity of self-attention through routes such as key–value representation compression, sparse/hybrid attention, and linearized state recurrence; MoE archi-

Structural optimization uses sparse activation and expert parallelism to control per-token computation while expanding model capacity; generation-path optimization for diffusion models targets diffusion/flow-matching models, accelerating multi-step generation through trainable trajectory compression or training-free cache reuse; chain-of-thought optimization reduces invalid reasoning tokens through budget control, difficulty adaptation, and redundancy suppression while maintaining reasoning quality; memory management structures, stores, retrieves, and updates historical interactions and experience, reducing repeated computation in long contexts and multi-turn dialogues; key–value cache compression reduces GPU-memory occupation and memory-access pressure in long-context inference through importance-based eviction, low-bit quantization, and prefix reuse; speculative decoding introduces a lightweight draft model to generate candidate tokens in parallel, which are then verified in one pass by the target model, producing equivalent outputs with fewer calls; model quantization converts weights and activations into low-bit representations, compressing GPU memory and accelerating matrix operations; model distillation transfers the knowledge and reasoning capabilities of large models to smaller models, enabling low-cost and efficient deployment. From the structural layer (attention, MoE, diffusion paths), to the process layer (chain of thought, memory management), and then to the compression layer (key–value cache, speculative decoding, quantization, distillation), these technologies form a multi-level optimization system and constitute an important foundation supporting the scaled deployment of large models as they move from "being able to reason" toward "low cost and high efficiency."

(I) Low-Complexity Attention Mechanisms

1. Technical Definition

A low-complexity attention mechanism refers to improvements made to the algorithmic structure of the Transformer self-attention layer, including the mode of information interaction among tokens, the organizational form of Q/K/V (query/key/value) representations, the visible range of context, and long-range memory pathways. Its purpose is to reduce computational complexity in long-sequence modeling, key-value cache occupation, and memory-access pressure during decoding, while preserving the model' s ability to represent global semantics, local details, cross-segment dependencies, and complex reasoning chains.

From the perspective of technical paradigms, current low-complexity attention mechanisms mainly include three categories. The first category consists of full-attention key-value compression variants. Among them, MHA (Multi-Head Attention, multi-head attention) is usually used as the standard full-attention baseline, while MQA (Multi-Query Attention, multi-query attention), GQA (Grouped-Query Attention, grouped-query attention), MLA (Multi-Head Latent Attention, multi-head latent attention), and related methods reduce key-value cache and memory-access costs while retaining global modeling capability by decreasing the number of key-value heads or compressing key-value representations. The second category is sparse/hybrid attention, which reduces invalid attention connections in long contexts through mechanisms such as sliding windows, global attention, dynamic sparse selection, compressed key-values, and attention aggregation. The third category is linear or more broadly efficient attention, which alleviates the quadratic growth of standard Softmax attention with context length through linearized computation, state recurrence, or the combination of attention with state-space structures. For token-level operation, low-complexity attention mechanisms directly affect long-context carrying capacity, per-token inference cost, the growth rate of the key-value cache, stability in multi-turn tasks, and the quality of long-document/long-code tasks.

Low-Complexity Attention Mechanism Optimization Roadmap

Algorithm layer: balancing global modeling capability, long-context cost, and key-value cache pressure

- **Full-attention compression optimization**
  - Full context visible
  - Attention matrix $O(n^2)$
  - Key-value cache grows with sequence length
- **Full-attention compression variants**
  - MQA / GQA / MLA
  - Reduce key-value heads or latent representations
  - Preserve global modeling capability
- **Sparse / hybrid attention**
  - Sliding window + global connections

  - DSA / CSA-HCA / MSA
  - Reduce ineffective attention links
- **Linear / efficient attention**
  - Lightning / state recurrence
  - Approximately linear complexity
  - Trade-off between quality and infrastructure
- **Basis for structural selection**
  - Context length
  - Task type
  - Key-value cache budget
  - Risk of quality degradation
- **Token-level operational gains**
  - Longer context capacity
  - Lower per-token cost
  - Lower key-value cache occupancy
  - More stable quality on long tasks

Optimization objective: under the premise of maintaining generation quality and context usability, reduce attention computation, key-value cache occupancy, and memory-access overhead.

Figure 2-1 Pathways for low-complexity attention mechanisms

2. Industry Progress

In recent years, research on low-complexity attention has shifted from "accelerating standard attention implementations" to "reconstructing long-context information flow." Relevant surveys argue that long-context large models require systematic trade-offs among global modeling capability, computational complexity, cache cost, and the risk of quality degradation [40]. This means that reducing complexity is not the sole objective: mainstream models do not simply pursue the lowest possible complexity order, but instead combine full-attention compression, sparse/hybrid attention, and linear/efficient attention under different model scales and task scenarios, seeking a balance among complexity, capability, and cost.

The first category is full attention and its key-value compression variants. MHA remains the most robust standard full-attention architecture in terms of quality, but during long-context decoding it incurs relatively high key-value cache and memory-access costs. GQA reduces the cache scale by decreasing the number of key-value heads. Qwen3 shows that it adopts GQA in both dense models and MoE models, and combines it with query-key normalization to enhance training stability [41]. DeepSeek-V3 adopts MLA, compressing keys and values into a latent representation space in order to reduce key-value cache costs during inference [42]. MiniMax-M2 uses full-context self-attention in all layers and adopts a GQA configuration in the organization of Q/K/V heads. Its paper explicitly points out that although linear and sparse attention have efficiency

advantages, in production tasks such as complex reasoning, code, and agents, the assessment of quality loss and the maturity of infrastructure remain key constraints [43]. This indicates that key–value-compressed full attention remains an important low-complexity baseline for today' s high-quality general-purpose models.

The second category is sparse/hybrid attention. This route no longer requires all tokens to be fully mutually visible; instead, it reduces long-context computational cost through local windows, global connections, dynamic sparse selection, or compressed keys and values. Qwen3.5-Omni adopts a hybrid-attention MoE framework to support 256K long contexts and multimodal long-sequence reasoning [50]. DeepSeek-V3.2 proposes DeepSeek Sparse Attention (DSA) to reduce the computational complexity of long-context attention [51]; DeepSeek-V4 further adopts a hybrid attention architecture composed of Compressed Sparse Attention (CSA) and Heavily Compressed Attention (HCA), reducing per-token computation and key–value cache occupancy in million-token scenarios [46]. GLM-5 and GLM-5.1 also demonstrate the application of the DSA route in optimizing long-context efficiency [47]. MiMo-V2.5 inherits the hybrid sliding-window attention language backbone of MiMo-V2-Flash and supports contexts of up to one million tokens [213].

Low-Complexity Attention Mechanism Optimization Roadmap

Foundation stage → Compression-enhancement stage → Sparse-attention breakthrough stage → Hybrid / sparse acceleration stage → Ultra-long-context exploration stage → Multi-path convergence stage

- **Foundation stage**: MHA serves as the baseline for all attention models.
- **Compression-enhancement stage**: GQA / MQA become mainstream, reducing KV-cache and computation costs.
- **Sparse-attention breakthrough stage**: Methods such as MLA emerge, significantly reducing KV and computation overhead during inference.
- **Hybrid / sparse acceleration stage**: DSA and hybrid methods support contexts more than 10× longer.
- **Ultra-long-context exploration stage**: Methods such as CSA / HCA / MSA move toward contexts more than 10× longer.
- **Multi-path convergence stage**: Models, kernels, and hardware are jointly optimized.

Three major technical directions advance in parallel

1. **Compression optimization for full attention**
   (Preserving capability while reducing cost)
   - **Fully available**; **stable capability**; **reduced KV cost**
   - **GQA / MQA** *(grouped / multi-query attention)*: Reduces the number of KV heads and improves training stability.
   - **KV compression**: Reduces KV and cache occupancy during inference.

- **Attention-structure optimization**: Optimizes attention layout and head allocation, reducing computational burden while ensuring quality.
- **Summary**: Full-attention compression is the core direction for today' s high-quality general-purpose models.

2. **Sparse / hybrid attention**
   (Lower cost, support for longer contexts)

   - **Local + global**; **dynamic selection**; **reduced complexity**
   - **Sparse patterns**: Sliding window / block sparsity / stripes, etc.; focus on key tokens and reduce computation.
   - **Hybrid attention**: Combines sparse attention with full attention, striking a balance between quality and efficiency.
   - **Adaptive routing**: Dynamically routes important tokens, further reducing computational overhead.
   - **Summary**: Sparse / hybrid attention is the main path for supporting contexts more than 10× longer.

3. **Linear / efficient attention**
   (Breaking the quadratic-complexity bottleneck and exploring the upper bound)

   - **Linear complexity**; **easier to scale**; **support for longer contexts**
   - **Kernelized / linear attention**: Feature mapping or kernel approximation; linear complexity and higher efficiency.
   - **Low-rank / projection methods**: Low-rank approximation or projection; reduces memory footprint and computation.
   - **More advanced variants**: State-space / channel-mixing / diffusion-architecture designs; explore the upper limit of efficiency.
   - **Summary**: Linear / efficient attention explores higher scalability and the capability ceiling for longer contexts.

Overall effects

- **Reduced GPU-memory usage**: Reduces KV-cache and intermediate-activation occupancy; can save **30%–80% GPU memory**.
- **Reduced memory access**: Improves data locality and lowers high-bandwidth memory (HBM) traffic; **memory access reduced by 2×–10×+**.
- **Improved prefill / decoding efficiency**: Higher computational density and hardware utilization; **throughput improved by 1.5×–5×+**.
- **Expanded context length**: Supports longer contexts under controllable costs; **supports 4×–20× longer contexts**.

Figure 2-2. Industry progress in low-complexity attention mechanisms

The third category is linear or, more broadly, efficient attention. This direction

attempts to break through the quadratic-complexity bottleneck of the standard attention mechanism, but in production-grade large models it usually must simultaneously address issues such as quality stability, low-precision representation, prefix caching, speculative decoding, and adaptation to inference frameworks. A representative work along this route is the Mamba architecture based on the State Space Model (SSM). Through selective state recurrence, it models long sequences with linear time complexity, compressing historical context into a fixed-size hidden state and thereby fundamentally eliminating the problem of KV cache growing linearly with context length [214]. A typical example is Falcon Mamba, which trained a 7B foundation model containing no attention layers at all on 5.8 trillion tokens, demonstrating that a pure Mamba architecture at this scale can match or even outperform mainstream Transformer models, while maintaining constant GPU-memory usage during long-sequence generation [215]. Jamba was the first to interleave Transformer attention layers and Mamba layers in proportion and combine them with MoE; while supporting a 256K long context, it reduces KV-cache usage by roughly one order of magnitude, validating the feasibility of an "attention + state recurrence" hybrid architecture for scaled deployment [216]. In addition, the MiniMax series of models provides a relatively clear evolutionary case: MiniMax-Text-01 once adopted a hybrid design interleaving Lightning linear attention with full attention; MiniMax-M2 returned to the full-attention GQA route out of considerations of quality reliability; and MiniMax-M3 further proposed MiniMax Sparse Attention (MSA), supporting million-token ultra-long contexts through more precise KV blocking and sparse selection [43][49]. This shows that efficient attention is not simply about pursuing low complexity, but rather about continuously balancing model capability, infrastructure maturity, and the cost of ultra-long contexts.

Overall, viewed as a whole, low-complexity attention mechanisms are forming a technical landscape in which three categories coexist: the KV-compressed full-attention route emphasizes capability stability and deployment certainty; the sparse / hybrid-attention route emphasizes cost control under million-token contexts; and the linear / efficient-attention route continues to explore longer contexts and lower complexity. Future large models will not be dominated by a single mechanism, but will instead adopt combined designs of multiple low-complexity structures according to model scale, task type, context length, KV-cache budget, and inference-cost objectives.

(II) MoE Architecture Optimization

1. Technical Definition

MoE (Mixture of Experts) architecture optimization refers to replacing part of the dense feed-forward networks in a large model with multiple expert networks, and using a router to dynamically select a small number of experts for each token to participate in computation. In this way, the total parameter scale of the model can be expanded while controlling the amount of activated computation per token. Unlike dense models, in which all parameters participate in

computation for every input, sparse MoE allows different tokens to access different combinations of experts. In theory, this can improve model capacity and task coverage while concentrating computation on the small number of experts selected by the router.

For inference-oriented optimization, MoE brings both advantages and challenges. It can provide a larger total parameter capacity under a similar amount of activated computation, but the inference system must handle issues such as expert-weight residency, uneven routing distributions, cross-GPU/cross-node All-to-All communication, fluctuations in in-batch expert load, and expert-parallel scheduling. For token operations, the core of MoE architecture optimization is not simply to pursue a larger parameter count, but to strike a balance among model quality, activated computation, communication overhead, GPU-memory occupancy, and service stability.

MoE Architecture Optimization and Inference Execution Pipeline

- **Input tokens**
  - Embedding
  - In-batch requests
- **Routing / gating**
  - Top-K expert selection
  - Routing scores
  - Load-balancing constraints
- **Token dispatch**
  - All-to-All communication
  - Capacity management
  - Token rearrangement
- **Expert 1**
  - Feed-forward network computation
  - Tensor parallelism / expert parallelism
- **Expert 2**
  - Feed-forward network computation
  - Tensor parallelism / expert parallelism
- **Expert 3**
  - Feed-forward network computation
  - Tensor parallelism / expert parallelism
- **Merge**
  - Weighted merge
  - Output hidden states
- **Monitoring and constraints**
  - Expert load
  - Congestion / overflow
  - Popular-expert statistics
- **Parallel runtime**
  - Data parallelism / tensor parallelism / pipeline parallelism / expert parallelism

– Communication–computation overlap
– Low-precision communication

Figure 2-3. MoE Architecture Optimization and Inference Execution Pipeline

2. Industry Progress

A 2025 survey study on MoE organized the mixture-of-experts architecture in large language models along dimensions such as algorithm design, system design, and application scenarios, and pointed out that the advantage of MoE lies in expanding model capacity with relatively low activated computation, while deployment gains depend on routing stability, expert load balancing, and system-level parallel capability [52]. This assessment is consistent with inference-system practice: whether MoE reduces token cost cannot be judged solely by total parameter count or activated parameter count; communication, batching patterns, and expert-load distribution must also be considered.

Industry Progress in MoE Architecture Optimization

1. **Early large-scale MoE**
   GShard / Switch Transformer
2. **Expert specialization**
   Mixtral / DeepSeekMoE
3. **Production-grade MoE models**
   DeepSeek-V3 / large-scale parallel training
4. **Inference-system optimization**
   Expert parallelism / all-to-all communication / load balancing

MoE inference-optimization framework

A. **Routing / gating**
- Top-K selection
- Routing scores

B. **Expert dispatch**
- Token reordering
- All-to-all communication

C. **Expert parallelism**
- EP / TP / PP hybrid parallelism

D. **Runtime monitoring**
- Hot experts / overflow
- Communication time / throughput fluctuations

Key trade-offs in inference deployment

- Model capacity
- Activation computation
- GPU-memory residency
- Communication overhead
- Service stability

From scaling model capacity through sparse activation toward coordination between expert parallelism and runtime governance

Figure 2-4 Industry progress in MoE architecture optimization

The scaling route of MoE can be traced back to GShard and Switch Transformer. GShard combines conditional computation with automatic partitioning, expanding model capacity through the expert layer [53]; Switch Transformer simplifies routing to Top-1 expert selection and alleviates expert congestion and training instability through mechanisms such as the capacity factor and load-balancing loss [54]. This line of work established the basic paradigm of MoE: using a router to select experts, using sparse activation to control the computation for each token, and using load-balancing mechanisms to prevent a small number of experts from overheating.

After entering the era of large language models, MoE placed greater emphasis on expert specialization and inference usability. Mixtral 8x7B sets 8 experts in each MoE layer and selects 2 experts for each token, enabling the model to obtain stronger overall capability with a relatively small number of activated parameters [55]. DeepSeekMoE proposes the idea of fine-grained expert partitioning and the separation of shared experts, with the goal of reducing expert redundancy and enhancing expert division of labor [56]. Subsequent models such as DeepSeek-V3 further combine MoE with large-scale parallel training, load balancing, and inference deployment, indicating that MoE has gradually moved from a research architecture into a production-grade large-model system [42].

A large part of the difficulty in optimizing MoE architectures lies at the system level. Expert routing distributes different tokens to experts on different GPUs or nodes, causing all-to-all communication and uneven intra-batch load to become bottlenecks. Inference frameworks such as TensorRT-LLM support expert parallelism, tensor parallelism, and their hybrid forms, for distributed hosting of expert weights across multiple GPUs [58]. In response to the Straggler Effect caused by uneven expert load, Capacity-Aware Inference regards the global waiting caused by overloaded experts as an important source of MoE inference latency, and proposes capacity-aware Token Drop and Expanded Drop mechanisms to improve expert utilization and end-to-end inference efficiency [59]. More broadly, MoE research is shifting from homogeneous experts and

a single sparse-activation mechanism toward heterogeneous experts, grouped experts, and resource-aware routing design. Heterogeneous expert combinations address the problem that traditional MoE experts are relatively uniform in scale and have difficulty matching the complexity of different tokens; they propose heterogeneous grouped-expert architectures, implement more flexible resource-aware expert combinations through a two-level routing mechanism, and introduce expert-group constraints and expert-assignment strategies around uneven GPU load and parameter-utilization efficiency, so that tokens of different complexity can be routed to more suitable expert groups while also accounting for balanced computation distribution across multiple GPUs [60]. Training and inference ecosystems such as DeepSpeed-MoE, Megatron-Core, and Tutel are also continuously optimizing expert parallelism, communication reordering, low-precision communication, overlap of computation and communication, and dynamic scheduling. The industry consensus is that the deployment efficiency of MoE cannot be judged only by the number of activated parameters; routing distribution, expert communication, batch shape, load balancing, and kernel support must also be considered simultaneously.

From the perspective of token operations, MoE architectures introduce new optimization dimensions: on the one hand, sparse activation helps

it helps increase model capacity under a limited compute budget; on the other hand, the per-token cost of MoE models is more strongly affected by expert distribution, parallel topology, and runtime scheduling. In practice, it is necessary to monitor expert load, overflow, Token Drop, hot experts, cross-node communication time, and throughput fluctuations under different batches, and to form a closed loop across model training, deployment configuration, and scheduling strategies. Only when routing stability, expert specialization, and system-level parallelism jointly mature can MoE be stably converted into gains in inference cost.

(III) Optimization of Generation Paths for Diffusion Models

1. Technical Definition

Optimization of generation paths for diffusion models refers to optimizing key links that affect generation efficiency and quality in the generation mechanisms of diffusion models and flow-matching models— "progressive denoising and gradual approximation of the target distribution" —from the perspectives of denoising trajectories or generation paths, so as to accelerate high-quality generation, reduce costs, and improve stability. Its core focus is not merely "reducing computation," but rather how, while maintaining generation quality, semantic consistency, and content stability as much as possible, to reconstruct the time-step progression, intermediate-state updating, and cross-step information reuse methods in the generation process, thereby improving overall inference efficiency.

Text in figure: “Focus on path optimization in the diffusion process”; “1. Training optimization: Distillation; Consistency Models; MeanFlow” ; “2. Training-free optimization: Sampler; Cache” ; x_t; x_0; x_0'.

Figure 2-5 Schematic diagram of generation-path optimization for diffusion models

2. Industry Progress

Generation-path optimization for diffusion models has evolved from an early single route centered on “reducing the number of sampling steps” into a technical system in which training-based optimization and training-free optimization develop in parallel. The former mainly compresses the generation path or reconstructs the generation mechanism through additional training, while the latter modifies the existing inference chain as much as possible without changing the parameters of the original model, in order to improve generation efficiency and engineering adaptability.

In the direction of training-based optimization, representative methods include distillation and consistency models. Such methods usually relearn a shorter generation path, compressing the original diffusion generation process that requires multiple iterative steps into a small number of steps or even single-step generation, and thus have strong potential for extreme latency optimization. Their advantages lie in a high degree of compression and a high theoretical upper bound for acceleration, making them suitable for generation scenarios with extremely high real-time requirements. At the same time, however, training-based optimization often depends on additional training data, training resources, and model-adaptation processes; research and development costs are relatively high, migration cycles are longer, and for rapidly iterating open-source models, commercial models, and different task scenarios, its generalization capability and deployment flexibility are also subject to certain constraints.

Trajectory Compression (TC)
Upper-bound tightening theorem: under the condition $L_t < 2$, $E_{\text{indirect}}(t,k) < E_{\text{direct}}(t)$

ShortDF
Discrete special case $\cdot\,\ell_1$ analogy (prior work)

Visible diagram labels: x_t, x_k, x_0; $\text{edge}(k,t)$; $\text{dist}(x_t,t) = |R(t,0)|$; $\text{dist}(x_k,k)$.

Relaxation condition (ℓ_1 form):

$$|R(t,0)| > \text{edge}(k,t) + \text{dist}(x_k,k)$$

Compress the iterative long path into a short path:

$$x_t \to x_k \to x_0 \Rightarrow x_t \to x_0$$

$\rightarrow$ a discrete ℓ_1 instance of upper-bound tightening

TrajXfer
Continuous instance · main-line method

Visible trajectory labels: $x_T \rightarrow x_0$, $x_t \rightarrow x_0$, $x_t \rightarrow x_0$, $x_t \rightarrow x_0$.

Direct mapping (target)

Base trajectory (curved · iterative sampling)

Guidance: $k \rightarrow 0$ (already-linearized anchor point), reverse $t \rightarrow k$ correction

Completely linearized trajectory (1–2-step sampling)

Upper-bound alignment loss:

$$\left[\max\left(0, E(x_t, t) - \operatorname{sg} E(x_k, k)\right)\right]^2$$

Trajectory LoRA: geometric correction
Frozen base model · 16 anchors · plug-and-play

ShortDF: ℓ_1 discrete relaxation TrajXfer: continuous statistical upper-bound alignment · DDPM/DDIM ($32^2 \rightarrow 256^2$) Flow Matching (1024^2)

Figure 2-6. Two complementary instances under the unified TC framework–ShortDF and TrajXfer

In the direction of training-based optimization, a representative approach is the idea of "trajectory compression" : compressing a multi-step decoding trajectory into fewer steps through training. Based on the unified TC framework, China Unicom characterizes both diffusion and flow-matching generative models from the perspective of probability-flow ODEs, and proves, on the basis of an upper-bound tightening theorem, that under mild stability conditions, the upper bound of the global truncation error of a segmented trajectory is strictly smaller than that of a direct single step. This provides a formal explanation for "exchanging trajectory subdivision for generation quality," and also lays a theoretical foundation for subsequent unified acceleration schemes that extend from low-resolution diffusion to high-resolution flow-matching foundation models. Along the same theoretical axis, the TC framework is further instantiated in two complementary classes of methods under discrete and continuous mathematical formulations. As a discrete method based on ℓ_1-norm constraints, ShortDF models reverse diffusion as a shortest-path problem on a weighted graph. By constructing an inter-step reverse graph and using residual relaxation loss to iteratively compress redundant paths, it compresses the number of sampling steps on CIFAR-10 from 10 to 2 and achieves an 18.5% improvement in FID; it also obtains leading results on benchmarks such as CelebA and LSUN Churches. The related work was accepted as a highlight at CVPR 2025

[61]. As the continuous main-line instance, TrajXfer targets large-scale text-to-image foundation models that are difficult to retrain in full. It transforms upper-bound tightening into a few-shot geometric alignment objective, and uses a low-rank adapter as a pluggable trajectory-correction module—using only 16 in-domain prompts, it can compress the inference steps of FLUX.1-dev from 20 to 2, achieving about 15× acceleration, and it can be naturally combined with community style LoRAs without degrading semantic expression.

At the same time, there is another class of training-free generation-path optimization methods. Compared with training-based optimization, training-free optimization places greater emphasis on low-intrusion modification of existing models. This route usually does not change the parameters of the original model; instead, it mainly improves inference efficiency by optimizing the sampling process or reusing intermediate computational results. Representative directions include sampler optimization and cache reuse. Sampler optimization mainly shortens the generation path while keeping the main model architecture unchanged by improving the denoising solver process, reducing ineffective time steps, or increasing the efficiency of single-step updates. Cache-reuse methods further exploit redundancy among adjacent time steps, different network levels, tokens, or cross-step states, selectively reusing intermediate results so as to reduce repeated forward computation and the read/write overhead of intermediate states. Related survey studies usually categorize this route under Diffusion Caching or Cache-Based Acceleration [62].

Within the training-free path, cache reuse has become one of the most active research directions in recent years. An early representative work is DeepCache, whose core idea is to exploit feature similarity between adjacent time steps in diffusion models and cache and reuse some intermediate features [63]. Methods at this stage mainly rely on fixed intervals and static reuse, verifying that temporally redundant information that can be systematically mined exists in diffusion-generation tasks, and also laying the foundation for subsequent research on path optimization. With the development of Diffusion Transformers, video generation, and larger-scale generative models, path optimization has gradually moved from static caching toward dynamic scheduling. Methods represented by TeaCache and DiCache have begun to introduce time-step-related change-estimation mechanisms, based on changes in state

the situation determines whether the cache is refreshed, thereby improving the ability to balance speed and quality [64]. At the same time, methods such as DuCa further push cache granularity down to the token level [65], indicating that the industry' s focus has shifted from "whether reuse is possible" to "which content is more suitable for reuse, and which content must be updated precisely" [66].

$$\sum_{t=t_a}^{t_b-1} f(\mathrm{L1}_{\mathrm{rel}}(\mathbf{F},t)) \le \delta < \sum_{t=t_a}^{t_b} f(\mathrm{L1}_{\mathrm{rel}}(\mathbf{F},t))$$

Figure 2-7. TeaCache trajectory-optimization rule; cited from Reference [64]

As AIGC transitions from diffusion models to flow matching, cache-reuse techniques for flow-matching models have likewise attracted considerable attention. Flow-matching models learn diffusion velocity fields, and the perspective of cache reuse has expanded from caching intermediate states in the diffusion process to exploring how to reuse early-stage velocity fields. However, overly aggressive reuse rates can cause drift in the decoding trajectory. To address this issue, China Unicom proposed MeanCache (ICLR 2026), which starts from the perspective of average velocity, uses the Jacobian–Vector Product to construct interval-average velocity, and combines it with a scheduling strategy oriented toward trajectory stability, thereby reducing local bias accumulation and trajectory drift under high-ratio cache reuse [67].

$$\hat{u}(z_t, t, s) = v(z_t, t) + (s - t)\mathrm{JVP}_{r \to t}$$

Figure 2-8. MeanCache optimizes generation trajectories from the perspective of average velocity

Related reviews further point out that the overall evolutionary trend in cache reuse is moving from "Static Reuse" toward "Dynamic Prediction," and is gradually transitioning from empirical local rules to more fine-grained and more globally oriented path-scheduling frameworks.

Overall, training-based path optimization is better suited to scenarios pursuing extreme compression, while training-free path optimization is more suitable for rapid deployment and continuous optimization of existing models. Within the training-free route, cache-based path optimization, owing to its low intrusiveness, transferability, and capacity for sustained evolution, is becoming an important development direction for inference optimization in current generative diffusion models. In particular, in scenarios such as text-to-video generation, high-resolution image generation, and next-generation flow-matching models, how to perform cache scheduling, error control, and stability optimization around the complete denoising path has become a key issue in industry competition.

(IV) Inference Chain-of-Thought Optimization

1. Technical Definition

Reasoning chain-of-thought optimization refers to a set of techniques for generating, using, and controlling intermediate reasoning processes when large models solve complex problems. Traditional large language models usually provide answers directly in an "input–output" manner, whereas chain-of-thought methods explicitly generate intermediate reasoning steps, enabling the model to decompose complex problems into multiple subproblems and thereby improve performance on tasks such as mathematics, coding, commonsense reasoning, and

symbolic reasoning. In reasoning-oriented large models, a chain of thought is not only a piece of readable text; it may also appear as internal reasoning tokens, a scratchpad, a tool-calling plan, search-tree nodes, or a compressed implicit state. Therefore, the core of reasoning chain-of-thought optimization is not to make the model think indefinitely, but to find the optimal balance among accuracy, latency, cost, interpretability, and safety.

From the perspective of technical objectives, reasoning chain-of-thought optimization mainly addresses four types of problems. The first is optimization of reasoning quality, that is, improving answer accuracy through more reasonable decomposition, verification, reflection, and path selection. The second is optimization of reasoning efficiency, that is, reducing redundant, repetitive, or ineffective intermediate steps and lowering output tokens, reasoning latency, and deployment cost. The third is optimization of reasoning structure, that is, extending a linear chain of thought into branching, tree search, programmatic reasoning, or alternating "reasoning–action" structures. The fourth is optimization of reasoning controllability, that is, dynamically allocating a reasoning budget according to task difficulty so that simple tasks can be answered quickly while complex tasks receive sufficient reasoning. Recent work on efficient reasoning commonly uses key evaluation indicators such as per-token accuracy, token error-correction trade-offs, and the Pareto curve between budget and accuracy.

From the perspective of implementation layers, reasoning chain-of-thought optimization can be divided into five categories. The first is prompt-level optimization, such as few-shot chain of thought, zero-shot chain of thought, and easy-to-hard prompting, which break complex tasks into smaller, solvable steps. The second is decoding-level optimization, such as improving robustness through self-consistency by sampling multiple reasoning paths and voting, at the cost of increasing reasoning overhead. The third is search and planning-level optimization, such as treating reasoning as a search over thinking nodes in a thought tree, supporting backtracking and self-evaluation. The fourth is training-level optimization, such as STaR, which uses correct reasoning samples generated by the model itself to iteratively train the model, while DeepSeek-R1 demonstrates a path for eliciting reasoning ability through reinforcement learning. The fifth is budget and compression-level optimization, which reduces reasoning tokens while maintaining effectiveness through methods such as token budgets, length penalties, draft shortening, and suppression of reflection.

What reasoning chain-of-thought optimization addresses is how a model should think, how long it should think, what structure it should use for thinking, and which parts of its thinking should be retained or compressed. In product practice, the complete original chain of thought does not necessarily need to be shown directly to users; it is often presented in the form of internal reasoning tokens, summaries, controllable drafts, or final explanations. This helps reduce information leakage and security risks, and also prevents users from mistaking intermediate text that is not fully faithful for the model' s actual causal process. Anthropic' s extended thinking, Google Gemini' s thinking budget, and Qwen3'

s thinking/non-thinking modes all reflect the trend of treating the reasoning process as an adjustable resource.

2. Industry Progress

Development Path for Optimizing Reasoning Chains of Thought in Industry

From explicit chains of thought, to reasoning-oriented models, and then to Efficient Reasoning

Early → Current / Future

Stage 1

Initiation stage: enabling models to reason

Core question: How can models explicitly write out reasoning steps?

1. **CoT Prompting**
   A small number of reasoning examples significantly improves performance on complex reasoning.
2. **Self-Consistency**
   Sample multiple CoTs and select the answer with the highest consistency.
3. **Least-to-Most**
   First decompose the problem, then solve it step by step, enhancing generalization.

Established the basic paradigm: explicit intermediate steps; decomposable, verifiable, and searchable.

Stage 2

Expansion stage: structured, tool-based, and search-based reasoning

Research focus: extending from linear chains of thought to more complex reasoning structures.

1. **STaR**
   Generate reasoning → filter correct answers → retrain; self-bootstrapping improvement.
2. **ReAct**
   Alternates Reasoning + Action, invoking tools and environmental information.
3. **Tree of Thoughts**
   Organizes thoughts in a tree structure, supporting evaluation, backtracking, and search.

4. **PAL**
   Converts problems into program steps, executed by an interpreter.

Drives chains of thought from textual explanation toward structured solving, tool augmentation, and search-based reasoning.

Stage 3

Leap stage: reasoning-oriented models become the focus of competition

After 2024: turning "reasoning strength" into a trainable and controllable capability.

1. **OpenAI o1**
   Invests more reasoning computation before answering, attempting different strategies, identifying errors, and improving.
2. **o1-mini**
   Cost-optimized for STEM scenarios, reflecting the direction of specialized small reasoning models.
3. **DeepSeek-R1 / R1-Zero**
   Large-scale reinforcement learning stimulates reasoning ability and spreads it across the open-source ecosystem.
4. **Claude 3.7 Sonnet / Claude 4**
   extended thinking
5. **Gemini 2.5**
   thinking budget
6. **Qwen3**
   thinking / non-thinking mode

Through RL, distillation, and post-training, models endogenously acquire long-reasoning capability.

Stage 4

Optimization stage: Efficient Reasoning

New bottleneck: overthinking—cost, latency, and context occupation.

A. Budgeting and dynamic control

1. **Token-Budget-Aware LLM Reasoning**
   Dynamically estimates the reasoning token budget according to problem complexity.
2. **Training Language Models to Reason Efficiently**
   Reduces unnecessary reasoning overhead while maintaining accuracy.

3. **Just Enough Thinking**
   Think less on simple problems and retain more computation for difficult ones.

B. Compressing and rewriting chains of thought

4. **Chain of Draft**
   Replaces lengthy CoT with shorter drafts of higher information density.
5. **ConciseR**
   Two-stage reinforcement learning that constrains repetition, circular reasoning, and ineffective reflection.
6. **NoWait**
   Suppresses reflective tokens such as "Wait"and "Hmm,"reducing redundant reasoning.
7. **Abstract-CoT**
   Completes reasoning in abstract vocabulary, reducing tokens by 11.6×.
8. **CoThink**
   An instruction model first provides a high-level outline, then the reasoning model expands it, reducing tokens by 22.3%.
9. **NAT**
   Makes the token budget a first-class optimization objective, matching performance with only 50% of the tokens.
10. **"Stop Overthinking"**
    Systematically summarizes three directions: model-based, reasoning-output-based, and input-prompt-based.

The core metric shifts from benchmark accuracy to: the optimal accuracy under a given cost.

Overall evolutionary trajectory:
1. Explicit steps → 2. Multiple paths / decomposition / tools / search → 3. Endogenous long-reasoning capability → 4. Optimal accuracy under cost constraints

Chain-of-thought optimization for reasoning is moving from "being able to reason" toward "high-quality, controllable, and scalable-deployable reasoning."

Figure 2-9 Industry development path for chain-of-thought optimization

In the early stage, the industry mainly focused on how to make models capable of reasoning. Chain-of-thought reasoning demonstrated that, for sufficiently large models, providing a small number of examples with reasoning steps can significantly improve performance on complex reasoning tasks [68]. Self-consistency further pointed out that complex problems often have multiple reasoning paths

that can reach the correct answer; therefore, multiple chains of thought can be sampled and the answer with the highest consistency selected [35]. The least-to-most approach first decomposes a problem and then solves it step by step, strengthening the ability to generalize from simple examples to more difficult problems [69]. These methods established the basic paradigm of reasoning optimization: through explicit intermediate steps, one-shot generation is transformed into a decomposable, verifiable, and searchable process.

Subsequently, the research focus expanded from linear chains of thought to more complex reasoning structures. STaR enables the model to self-improve using its own reasoning data through a cycle of generating reasoning–filtering correct answers–retraining [70]. ReAct alternately generates reasoning trajectories and actions, enabling the model to think while invoking external tools or environmental information, thereby alleviating the hallucinations and error propagation of pure chains of thought in factual queries [71]. Tree of Thoughts organizes the reasoning process into a tree structure, allowing the model to evaluate, backtrack, and search across multiple candidate lines of thought [72]. PAL converts natural-language problems into intermediate program steps and hands the computation to an interpreter, improving reliability in arithmetic and symbolic tasks [73]. These representative works pushed chains of thought from textual explanation toward structured solving, tool augmentation, and search-based reasoning.

After 2024, reasoning-oriented models became the focus of industry competition. The OpenAI o1 series emphasizes investing more reasoning computation before answering, and through training enables the model to try different strategies, identify errors, and improve its line of thought [74]; o1-mini is cost-optimized for STEM scenarios, reflecting the direction of specialized small reasoning models [75]. DeepSeek-R1 demonstrated the feasibility of stimulating reasoning ability through large-scale reinforcement learning, and open-sourced R1, R1-Zero, and distilled models, pushing reasoning capability from closed-source frontier models into the open-source ecosystem [76]. Products such as Anthropic Claude 3.7 Sonnet, Claude 4, Google Gemini 2.5, and Qwen3 further design reasoning strength as a resource controllable by users or developers, such as extended thinking, thinking budgets, and thinking/non-thinking modes [77][78][79].

As reasoning models enter real-world business settings, the new bottleneck becomes overthinking. Taking DeepSeek-R1 as

For example, for the simple arithmetic problem “100+200−300=?” , it may generate hundreds of tokens of lengthy reasoning, bringing significant resource waste and increased latency. Recent research on efficient reasoning has centered on the contradiction of “how to reduce reasoning cost while preserving reasoning capability.”Token-budget-aware large-model reasoning proposes dynamically estimating the reasoning token budget according to problem complexity and internalizing budget awareness through prompting or post-training, so that the model knows before reasoning how many tokens it should spend on thinking [80]. Reinforcement learning is used to directly optimize token efficiency: lan-

guage models are trained for efficient reasoning, and a length-penalty term is added to the reward function to encourage the model to reduce unnecessary reasoning overhead while maintaining accuracy [81]. On the basis of thinking "just enough," adaptive length penalties are further introduced, dynamically adjusting the penalty strength according to problem difficulty so that the model thinks less on simple problems and retains more computation for difficult ones [82]. These works point to a common trend: the core metric of reasoning optimization is no longer merely benchmark accuracy, but the optimal accuracy under a given cost.

Another class of efficient-reasoning work focuses on compressing or rewriting the chain of thought. Chain of Draft proposes replacing lengthy chains of thought with shorter drafts of higher information density, in order to reduce latency and token cost [83]. ConciseR uses two-stage reinforcement learning to pursue concise reasoning, constraining repetition, detours, and ineffective reflection in long chains of thought [84]. Researchers have observed that explicit self-reflection tokens such as "Wait" and "Hmm" in some R1-style models can induce overthinking; therefore, by suppressing the relevant tokens during inference, redundant reasoning can be reduced without training the model [85]. The Abstract-CoT scheme takes another route: by constructing a discrete latent reasoning mechanism, it allows the model to complete reasoning in an abstract vocabulary rather than in natural language, achieving comparable performance while reducing the number of reasoning tokens by 11.6 times [86]. The CoThink method adopts a dual-model collaboration paradigm—the instruction model first generates a high-level problem-solving outline, and the reasoning model then expands on it to solve the problem—achieving a 22.3% token reduction on three mainstream datasets [87]. The NAT framework introduces token budget as a first-level optimization objective into the reinforcement-learning process, using only 50% of the tokens to match the GRPO performance obtained by full-sequence training [88]. Rice University' s survey "Stop Overthinking" systematically reviews this area, organizing existing work into three major directions: model-based optimization, optimization based on reasoning outputs, and optimization based on input prompts [89].

However, most of the efficient-reasoning methods above adopt a "one-size-fits-all" compression strategy—regardless of problem difficulty, they uniformly reduce reasoning tokens or truncate the chain of thought. This may work acceptably on simple problems, but it damages the model' s reasoning ability on complex problems. In response to this limitation, China Unicom proposed a difficulty-adaptive optimization approach, becoming a representative industrial practice in the shift of efficient reasoning from uniform compression toward differentiated control. In January 2025, China Unicom released the Yuanjing Chain-of-Thought large model, the first open-source general-purpose chain-of-thought large model from a central state-owned enterprise. It outperformed Tongyi Qianwen QwQ and was comparable to OpenAI o1 on reasoning tasks across multiple disciplines and scenarios. The core innovation of the Yuanjing model is the "adaptive slow thinking" technical framework, which includes two

dimensions: task adaptation and difficulty adaptation. Task adaptation performs mixed fine-tuning by reasonably combining general-domain instruction data with long-chain-of-thought data for reasoning tasks, enabling the model to use a long-chain-of-thought mode in reasoning tasks while outputting concise answers in routine tasks. In empirical tests, while ensuring accuracy, the Yuanjing model' s answer length on non-reasoning tasks was significantly shorter than that of Tongyi Qianwen QwQ. Difficulty adaptation is embodied in the Difficulty-Adaptive Slow-Thinking framework [90]. This framework introduces the Token Length Budget (TLB) metric, combining the sampling accuracy of a problem with the average length of correct answers to establish a mapping between problem difficulty and target answer length: the higher the sampling accuracy, the closer the TLB is to the average length of correct answers, meaning that simple problems should generate shorter answers; the lower the sampling accuracy, the closer the TLB is to the maximum generation length, meaning that difficult problems should be encouraged to reason deeply. Based on TLB, DAST designs a reward calibration mechanism: if a correct answer exceeds the budget, its reward is decayed (the simpler the problem, the more severe the decay), while if an incorrect answer falls below the budget, additional reasoning is encouraged—and two types of preference data, double-correct pairs and double-wrong pairs, are constructed for SimPO optimization. In terms of practical deployment, China Unicom has applied adaptive slow-thinking technology to scenarios such as the "Hundred Questions and Hundred Answers" in the communications supply chain and port large-model safety operations, saving approximately 30% of reasoning computation while maintaining complex reasoning capability.

Taken together, optimization of the reasoning chain of thought is forming a clear evolutionary path: the first stage is chain-of-thought prompting, which asks the model to write out steps explicitly; the second stage is multi-path reasoning, decomposition, tools, and search, which make the reasoning process more reliable; the third stage is reasoning-oriented models, in which reinforcement learning, distillation, and post-training enable the model to acquire long-reasoning capability internally; and the fourth stage is efficient reasoning, which achieves scalable deployment through budget control, dynamic length allocation, concise reasoning, and suppression of excessive reflection. The difficulty-adaptive paradigm represented by China Unicom DAST is pushing this stage from uniform compression toward differentiated regulation, so that the allocation of reasoning resources can be precisely matched to problem complexity.

(V) Memory Management

1. Technical Definition

Memory management for large language models refers to the systematic mechanisms used during large-model inference to selectively write, organize, compress, retrieve, update, and forget historical interactions, task states, external knowledge, execution experience, user preferences, and intermediate reasoning traces.

Its core objective is not simply to "extend the context," but rather, under limited context windows, limited reasoning budgets, and dynamic task environments, to provide the model with sustained, controllable, low-noise informational support, thereby improving long-horizon reasoning, personalized interaction, multi-round task execution, and cross-task transfer capabilities.

From the perspective of inference optimization, memory management usually consists of three parts: memory formation, memory evolution, and memory retrieval. Memory formation is responsible for transforming raw interactions, tool-call results, environmental feedback, code-execution trajectories, planning processes, or self-reflection outcomes into reusable memories, rather than preserving the entire context intact. Common approaches include summary compression, experience extraction, knowledge distillation, graph-structure construction, vectorized encoding, and parameterized internalization. Memory evolution is responsible for maintaining the quality of the memory repository, including deduplication and merging, conflict resolution, elimination of low-value information, cleanup of expired information, and elevation of repeatedly occurring experiences into more abstract strategies. Memory retrieval is responsible for recalling relevant memories before or during inference according to the current task, user intent, and context state, and injecting them into the model context after reranking, filtering, aggregation, or compression. This life cycle can be summarized as a closed loop of "write—organize—recall—use—update." Related definitions can be found in the uploaded materials' descriptions of memory states and of formation, evolution, and retrieval operators.

Classified by carrier form, large-language-model memory can be divided into three types. The first is lexical-level explicit memory, in which discrete units such as text fragments, dialogue records, task trajectories, code snippets, user profiles, knowledge-graph nodes, or multimodal entries are stored, usually implemented through vector databases, key-value storage, databases, graph databases, or hierarchical indexes. This type of memory is transparent, editable, and auditable, and is currently the mainstream form in engineering practice. The second is parametric memory, in which knowledge and experience are written into model weights through full-parameter fine-tuning, parameter-efficient fine-tuning, model editing, or continual learning, allowing them to function implicitly in subsequent inference. Its advantages are low invocation cost and no need for explicit retrieval, but it carries higher risks in updating, forgetting control, and interpretability. The third is latent memory, in which historical information is encoded into hidden states, memory vectors, long-term key-value caches, or dedicated memories, and is continuously accessed by the model during inference; it is suitable for low-latency, long-sequence, and multi-turn state-maintenance scenarios.

Classified by function, memory management mainly serves three types of reasoning needs. Factual memory records user preferences, environmental states, external knowledge, and long-term facts, addressing the question of "what the model should know" ; experiential memory records success/failure cases, strate-

gies, tool-use methods, and reusable skills from historical tasks, addressing the question of "how the model has done things before"; working memory maintains the plans, temporary variables, intermediate conclusions, and tool-returned results in the current task, addressing the question of "what the model is currently processing." For inference optimization

More specifically, working memory reduces state loss in long-chain tasks; factual memory improves personalization and consistency; and experiential memory improves cross-task self-improvement capabilities.

2. Industry Progress

Figure 2-10 Technical architecture of memory management for optimizing large language model inference

The figure is titled "Memory Management Architecture for LLM Inference Optimization" and includes the following visible components:

- **Input Sources**
  - User request
  - Historical dialogue
  - External documents
  - Retrieval results
  - Environment state
  - Tool feedback
  - Multi-turn task trajectory
- **LLM Agent inference execution layer**
  - Task parsing
  - Context assembly
  - Planning and decision-making
  - Chain-of-thought reasoning
  - Tool invocation
  - Result generation
  - Memory injection
  - Reasoning feedback
- **Memory Management Engine**
  - Memory Formation: abstraction, compression, event extraction, model encoding, structured representation
  - Memory Retrieval: semantic recall, graph retrieval, reranking, filtering, memory injection
  - Memory Evolution: update, merge, conflict handling, forgetting, deduplication, hierarchical storage
  - Write
  - Retrieve
  - Update
- **Functional Memory Types**
  - Working Memory: task state, temporary context, intermediate con-

clusions
  - Factual Memory: user profile, long-term facts, environmental knowledge
  - Experiential Memory: success/failure cases, strategy templates, tool-use experience, skill library
- **Memory Representation**
  - Token-level Memory: dialogue fragments, summary blocks, task trajectories, knowledge graphs, vector entries
  - Parametric Memory: fine-tuned parameters, LoRA adapters, model-edited knowledge
  - Latent Memory: hidden states, long-KV cache, memory tokens, latent representation
- **Infrastructure**
  - Vector Database
  - Graph Database
  - Relational Database
  - Object Storage
  - Cache
  - Model Services
- **Benefits and Outcomes**
  - Extended Context Capability
  - Improved Reasoning Stability
  - Cross-turn Memory Retention
  - Personalized Services
  - Experience Reuse
  - Improved Response Accuracy
  - Reduced Inference Cost

Early memory management mainly centered on long dialogues and personalization. MemoryBank maintains cross-session consistency and personalized responses in chat agents by persistently storing user profiles, emotional states, and historical conversations [91]. MemGPT further proposed an idea analogous to operating-system virtual memory: treating the limited context as "main memory" and external storage as "disk," with the model actively deciding when to page, retrieve, and update memories [92]. Such methods established the engineering paradigm for explicit memory management: upgrading the context window from a passive input area into a schedulable runtime resource.

Subsequently, memory management gradually expanded from "storing facts" to "distilling experience." Reflexion writes linguistic feedback after task failures into memory, guiding the model to correct its behavior in subsequent attempts [93]; Voyager distills successful exploration experience in the Minecraft environment into a reusable code library of skills, enabling continual learning in open-ended tasks [94]; ExpeL induces general experience and few-shot examples from historical trajectories to improve performance on subsequent reasoning tasks [95]. These works show that memory can not only preserve user facts, but also serve as an important mechanism by which models accumulate strategies, reuse skills,

and reduce repeated trial and error during reasoning.

In the past two years, industry attention has shifted toward structured, automated, and evolvable memory. Mem0 extracts dialogue history into updatable long-term memory and supports more fine-grained management of user preferences, emphasizing low latency, high hit rates, and controllable updates [96]. Engineering systems such as Zep introduce time-aware and graph-structured memory, supporting long-term dialogue and complex retrieval through modeling of entities, events, and relations [97]. Recent works such as A-MEM and G-Memory further adopt networked notes, hierarchical graphs, and multilevel summaries, organizing fragmented memories into relational structures to improve multi-hop reasoning, context aggregation, and conflict-handling capabilities [98][99]. This marks a transition of memory systems from "vector recall" toward a knowledge/experience foundation that is "organizable, reason-able, and maintainable."

In the direction of inference optimization, the combination of reinforcement learning and memory management has become a representative recent trend. Works such as MemAgent, Mem1, and Memory-R1 attempt to enable models to learn when to write, what to write, when to retrieve, and how to compress memories, transforming memory scheduling that previously relied on rules and prompts into trainable policies [100][101][102]. The value of such methods lies in the fact that they no longer treat memory as a fixed external component,

Rather, it incorporates memory operations into the model' s reasoning behavior itself, enabling the model to actively weigh information gain, context cost, and noise risk during inference.

Another important line of work is to combine memory with long context, key-value caches, and latent states. Traditional long-context methods mainly rely on enlarging the window or compressing tokens, but their cost rises markedly with length. In recent years, methods such as key–value compression, key–value reuse, long-term memory tokens, and latent-vector memory have attempted to retain key information without fully expanding historical text. Representative works such as MemoryLLM, M+, and MemGen explore transforming historical information into continuously accessible latent memory representations, thereby reducing the cost of explicit text retrieval and context concatenation [103][104][105]. In high-frequency interaction, real-time agents, and long-task execution, such mechanisms are expected to become important inference-acceleration tools beyond explicit memory.

Overall, memory management in large language models is evolving from a "context patch" into core infrastructure for reasoning systems. In engineering practice, hybrid architectures are usually adopted: short-term working memory is responsible for the current task state; vector/graph databases handle the recall of long-term facts and experience; summarization and forgetting mechanisms control cost and noise; and parametric or latent memory carries high-frequency, stable capabilities. Key future directions include automated memory scheduling,

cross-modal memory, verifiable memory, privacy and permission control, shared memory across agents, and memory compression and hierarchical caching oriented toward inference cost.

(VI) Key-Value Cache Compression

1. Technical Definition

Key-value cache compression refers to the more efficient storage, reuse, quantization, eviction, or hierarchical management of the key-value states of historical tokens during the autoregressive inference process of large models, in order to reduce GPU memory usage and access-bandwidth pressure. The role of the key-value cache is to avoid recomputing the keys and values of the entire historical context every time a new token is generated; however, its size grows linearly with the number of layers, the number of attention heads, the hidden dimension, batch size, and context length. In long-context, high-concurrency, and multi-turn dialogue scenarios, the key-value cache may become the primary resource limiting throughput, context length, and per-token cost.

From the perspective of optimizing long-context inference, key-value cache compression should be considered in coordination with context compression in a broader sense. Context compression refers to transforming the original context into a shorter, cheaper, more reusable, or more computation-friendly representation before the model processes long documents, multi-turn histories, RAG (Retrieval Augmented Generation) retrieval results, or code-repository context. Its objects may be input tokens, soft tokens, summary vectors, memory states, retrieval embeddings, sparse-attention indices, or compressed key-value states. Unlike simply enlarging the context window, context compression emphasizes reducing prefilling cost, attention-computation overhead, key-value cache occupancy, and end-to-end latency while preserving task-relevant information as much as possible.

For token operations, key-value cache compression and context compression jointly affect serviceable concurrency, long-context capacity, first-token latency, per-token cost, and the quality of long-document tasks. The former mainly acts on inference runtime states, addressing cache GPU-memory usage, access bandwidth, and cache reuse; the latter acts more on the input side and the prefilling stage, reducing the scale of context or attention connections that the model must directly process. For long-document question answering, multi-turn dialogue, meeting minutes, enterprise knowledge bases, and RAG systems, practical deployment usually requires the simultaneous design of a closed loop of "input-side compression—prefix/context reuse—key-value cache management—quality evaluation—exception fallback," rather than adopting a single compression algorithm in isolation.

Collaborative Mechanism of Key-Value Cache Compression and Context Compression

- **Long-context requests**
  - Long documents / RAG
  - Multi-turn history
  - Codebase context

→

- **Input-side context compression**
  - Prompt compression
  - Soft tokens / summaries
  - Token selection / sparse indexing

→

- **Key-value cache management**
  - Block pool
  - Page table
  - Prefix / context reuse

→

- **Paging and reuse**
  - PagedAttention
  - Prefix / Radix cache
  - Shared key-value cache blocks
- **Compression and storage**
  - Important-token retention
  - Key-value cache quantization / sliding window
  - Tiered storage / offloading

→

- **Decoding**
  - Attention kernels
  - Quality fallback
  - Metric monitoring

Key trade-off: the higher the compression ratio, the more attention must be paid to information fidelity, long-context accuracy, decoding latency, kernel support, and exception fallback.

Production systems usually combine input-side compression, prefix reuse, paging management, importance-based eviction, low-bit key-value cache quantization, and tiered caching.

Figure 2-11 Collaborative mechanism of key-value cache compression and context compression

2. Industry Progress

A study on key-value cache compression systems points out that, as context length increases, attention computation and key-value cache storage jointly amplify inference costs. It also classifies existing methods according to their underlying principles and implementation mechanisms, emphasizing that there are significant trade-offs among compression ratio, inference latency, and long-context quality [106]. This indicates that key-value cache compression cannot be evaluated solely by the proportion of GPU memory saved; decoding speed, cache hit rate, task accuracy, and exception fallback must also be assessed simultaneously.

Industry Progress in Key-Value Cache Compression

Industry progress directions

1. **Paging and reuse**
   - PagedAttention / prefix cache / RadixAttention
2. **Importance retention**
   - Scissorhands / H2O / SnapKV
3. **Key-value quantization**
   - 8-bit
   - KIVI / KVzip / TurboQuant
4. **Context compression**
   - Gist tokens / LLMLingua / Minference / xRAG

Production-grade collaborative pipeline

A. **Request and context profiling**
- Prefill / decoding / long context

B. **Key-value cache management**
- Block pool / page table / prefix reuse

C. **Compression-strategy selection**
- Token importance / sliding window / budget allocation

D. **Low-bit quantization and tiered storage**
- FP8 / INT8 / INT4 / GPU-CPU offloading

E. **Quality and performance monitoring**
- Hit rate / TTFT / TPOT / long-context accuracy

Trade-off among compression ratio, latency, and quality

From cache management toward collaboration among importance compression, low-bit quantization, and context compression

Figure 2-12 Industry progress in key-value cache compression

At the engineering-system level, vLLM' s PagedAttention is a representative advance in key-value cache management. This method draws on the idea of

virtual memory in operating systems, partitions the key-value cache into fixed-size blocks, and through

mapping from logical blocks to physical blocks supports non-contiguous storage, thereby reducing the internal and external fragmentation caused by traditional contiguous pre-allocation [48]. On this basis, vLLM' s automatic prefix caching reuses shared key-value caches through hashing [107], while SGLang' s Radix-Attention uses a prefix tree to organize key-value caches, supporting the reuse of shared prefixes for structured generation, multi-branch inference, and multi-turn tasks [108]. These mechanisms mainly address cache reuse under dynamic requests, shared prefixes, and multi-turn interactions.

Academia has also proposed a variety of key-value cache compression methods based on importance and low-bit representations. Scissorhands, H2O, and SnapKV respectively select and retain more critical historical states from perspectives such as importance persistence, Heavy Hitter Tokens, and pre-generation attention patterns, thereby compressing the key-value cache under a fixed budget [109-111]. KIVI, KVzip, and TurboQuant further reduce the representation cost of caches from the perspectives of key-value cache quantization or context reconstruction [112-114]. The common challenge for such methods is the need to strike a balance among compression ratio, long-context quality, decoding latency, and compute-kernel support.

Context-compression techniques adjacent to key-value cache compression are becoming an important supplement for reducing the cost of long-context inference. Methods such as Gist Tokens, AutoCompressor, and ICAE attempt to compress the original context into a small number of soft tokens, summary vectors, or memory slots, thereby reducing the context length that the model must directly process [115-117]. LLMLingua and LongLLMLingua reduce the number of input tokens through explicit prompt compression and token selection, making them suitable for one-off question answering, summarization, and query-explicit RAG scenarios [118-119]. Such methods mainly operate on the input side and during the prefill stage, complementing runtime key-value cache compression.

Recent work has also begun to focus on context compression on the computation side and the retrieval side. MInference 1.0 reduces the attention connections that actually need to be computed during long-context prefill through dynamic sparse attention, making it suitable for scenarios with high prefill costs, such as long prompts, long-code contexts, and long RAG inputs [120]. xRAG, for RAG scenarios, integrates the dense embeddings of retrieved documents into the representation space of the language model, thereby reducing or even replacing raw document-token input [121]. Overall, production systems usually need to combine input-side compression, prefix reuse, paging management, importance-driven eviction, key-value quantization, and necessary quality rollback mechanisms, while also monitoring cache hit rate, GPU memory occupancy, TTFT, TPOT, hallucination, and long-context accuracy. Key-value cache compression should therefore be regarded as a runtime core capability within a long-context

compression system, rather than as an isolated GPU-memory optimization technique.

(VII) Speculative Decoding

1. Technical Definition Speculative decoding is an inference-acceleration technique for autoregressive large models. Its core idea is as follows: instead of allowing the target large model to generate tokens serially one by one, a lighter-weight "drafter" first proposes multiple candidate tokens at once—such as a draft model, draft head, early-exit layer, or n-gram proposer—and the target model then completes verification and correction in a single parallel forward pass, thereby generating the same number of valid tokens with fewer calls to the target model [33][122]. The value of this method is highly suited to token operations: it pushes the object of cost optimization down from the "request level" to the "token level," directly reducing inference latency and the unit cost per token.

A typical procedure is as follows: given a context, the drafter generates candidate tokens of length K; the target model computes the probabilities at these positions in one pass; the system accepts the candidates one by one according to an acceptance criterion; if a token is not accepted, the system resamples from the target-model distribution and truncates the subsequent candidates. Under a strict accept–reject rule, classical speculative sampling can preserve the same output distribution as the target model, and therefore constitutes "lossless acceleration" ; but some

Some engineering implementations adopt approximate acceptance, typicality acceptance, or greedy-specialized strategies, requiring trade-offs among quality, throughput, and consistency [33][123].

From the perspective of token-level operations, speculative decoding is not a single-point algorithm, but a system capability for inference comprising "draft generation—parallel verification—dynamic scheduling—closed-loop metrics." Key metrics include the draft acceptance rate, the average number of accepted tokens per round, draft-model overhead, target-model verification overhead, throughput changes, benefits in long-output scenarios, and fallback strategies under different business traffic patterns. It is usually more suitable for scenarios in which the decoding phase accounts for a large proportion, outputs are relatively long, the target model is constrained by memory bandwidth, and the distributions of the draft model and target model are close. In scenarios with a high prefill proportion, already large batches, a low draft acceptance rate, or an overly heavy draft model, the gains decrease markedly.

2. Industry Progress

LLM Speculative Decoding Technology Architecture Diagram

1. User input / Prompt →
2. Context encoding (Prefill) →
3. KV cache initialization (KV Cache) →
4. Draft generator (Draft Model / Proposer); lightweight draft model / auxiliary generation module →
5. Generate multiple candidate tokens; multi-token candidates →
6. Target-model parallel verification (Target LLM Verifier); parallel verification of multiple candidate positions →
7. Accept/reject decision.

Accept path (Accept):
7A. Write output tokens → 7B. Update KV Cache → output sequence / final generation result.

Reject path (Reject):
7C. Target model generates a correction token → 7D. Truncate subsequent candidates; retain the valid prefix and discard subsequent candidates → output sequence / final generation result.

Decode loop (Decode Loop): iteratively generate the next round of tokens until an end-of-generation symbol is produced or the maximum length is reached.

Legend: main flow (one-way forward); accept path (Accept); reject path (Reject); decode loop (Decode Loop); KV Cache (key-value cache); model module; parallel verification positions (multiple candidate positions); verification result (accepted position).

Figure 2-13 Architecture diagram of speculative decoding technology for large language models

Early representative work was dominated by "dual-model speculative sampling." Leviathan et al. proposed using a small model to generate candidates and the target model to verify them in parallel, thereby reducing the number of serial steps without changing the output distribution of the target model. Chen et al. also systematized this paradigm from the perspective of speculative sampling [33][122]. The core bottleneck at this stage was that a high-quality draft model had to be maintained for each target model; if the draft model was too weak, the acceptance rate would be low, whereas if it was too strong, it would offset the acceleration gains.

Subsequently, research moved from "linear drafts"toward "tree-structured drafts and batched verification." SpecInfer organizes multiple candidate sequences into a token tree and uses the target LLM for tree-structured parallel verification,

reporting gains on the order of 1.5–3.5× in distributed and offloaded inference scenarios [124]. Such methods advance speculative decoding from an algorithmic technique to inference-serving system design: key-value cache management, tree-structured attention masks, concurrent scheduling, GPU memory consumption, and batch morphology all become key factors affecting the gains.

After 2024, the industry began to reduce its reliance on independent small models. Medusa adds multiple decoding heads to the target model, directly predicts multiple subsequent tokens, and verifies candidates simultaneously through a tree-structured attention mechanism. The paper reports that Medusa-1 can achieve approximately 2.2× acceleration while maintaining generation quality, and that Medusa-2 further improves under joint training [125]. LayerSkip, by contrast, uses early-exit loss and layer dropping to enable earlier layers of the model to acquire predictive capability; during inference, shallow layers perform "self-speculation" while deeper layers verify, reducing the additional memory and deployment complexity of dual-model schemes [126].

Another main line is feature-level drafting. EAGLE argues that token-level autoregression is less stable than feature-level prediction, and instead predicts the penultimate-layer features of the target model before generating candidate tokens; on LLaMA2, Vicuna,

Mixtral and other models reported 2.7–3.5× level acceleration [127]. EAGLE-2 further introduced a context-aware dynamic draft tree, dynamically adjusting the draft tree according to contextual confidence, and reported a 3.05–4.26× acceleration, with a further 20%–40% improvement over EAGLE-1 [128]. EAGLE-3 in 2025 shifted from feature prediction to direct token prediction, and fused multi-layer features with a test-time mechanism; covering both chat models and reasoning models, it reported acceleration of up to 6.5×. This has made the EAGLE series one of the most important speculative-decoding routes in current open-source inference engines [129].

In terms of engineering and productization, inference frameworks such as vLLM, SGLang, and TensorRT-LLM have incorporated multiple speculative-decoding capabilities. The vLLM documentation lists many categories of candidate generators, including EAGLE, multi-token prediction, draft models, PARD, MLP, n-gram, and suffix decoding. SGLang supports EAGLE-2/EAGLE-3, multi-token prediction, conventional draft models, and n-gram variants. NVIDIA TensorRT-LLM also provides EAGLE, Medusa, and draft-model solutions, and reports in its official blog token-throughput improvements of up to 3.6× [130][131][132]. This indicates that speculative decoding has moved from research prototypes into production inference stacks; the focus has shifted from "whether it can accelerate" to "under which constraints of business traffic, batching, hardware, and service-level protocols it can make money reliably."

Recent representative work continues to unfold around "stronger drafters, lower draft overhead, and greater operability." ReDrafter uses a recurrent-neural-network drafter, beam search, and a dynamic tree-attention mechanism,

and validates inference gains in H100 and Apple in-house chip environments [133]. SpecForge / SpecBundle further addresses the scarcity of high-quality EAGLE-3 draft-model weights and insufficient training infrastructure, providing a production-oriented training framework and draft weights for mainstream open-source models [134]. P-EAGLE in 2026 changes EAGLE' s autoregressive drafting into parallel drafting, predicting multiple draft tokens in a single forward pass, and has been integrated with vLLM. At the same time, SPEED-Bench begins to emphasize evaluation of speculative decoding in multilingual domains and real service scenarios, avoiding overestimation of gains only on low batch sizes or synthetic data [135][136].

Overall, the evolutionary path of speculative decoding for large language models is: from a general lossless algorithm of "small-model drafting + large-model verification,"it has developed into a compositional system optimization of "multi-head / self-speculative / feature-level / dynamic-tree / parallel drafting." For token-serving operations teams, the implementation priority is not to blindly pursue the highest multiplier reported in papers, but to establish strategies for selecting small models according to business traffic profiles: code, summarization, RAG, and multi-turn question answering may first test n-gram, EAGLE-3, and P-EAGLE; scenarios requiring strong consistency should preferentially use strict acceptance criteria; high-concurrency throughput scenarios need to coordinate continuous batching, key-value caching, quantization, and routing strategies, ultimately taking "effective token cost" and "acceptance under service-level-agreement constraints" as the launch criteria.

(VIII) Model Quantization

1. Technical Definition

Model quantization refers to converting the weights, activations, or key-value caches in large models from high-precision representations such as FP16/BF16 into low-precision representations such as INT8, INT4, FP8, and FP4, and, through calibration, scaling, outlier handling, and high-performance kernel design, reducing memory footprint, memory-access bandwidth, and computational cost while preserving model performance as much as possible. According to the stage of intervention, quantization can be divided into PTQ, QAT, and low-bit fine-tuning; according to the object, it can be divided into weight quantization, activation quantization, weight-activation joint quantization, and key-value cache quantization.

For token-serving operations, the core value of quantization lies in lowering the threshold for model deployment and the resource cost per token. Weight quantization can reduce the resident GPU memory of a model, enabling the same hardware to host a larger model or more replicas; activation and key-value cache quantization affect runtime GPU memory and bandwidth in high-concurrency and long-context scenarios. It should be noted that lower bit width does not necessarily mean faster end-to-end execution; actual gains depend on hardware

generation and computational

whether the kernel fuses dequantization with matrix multiplication, batch size, context length, and the task' s tolerance for quality.

Model quantization process and inference deployment chain

- **High-precision model**
  - FP16 / BF16
  - Weight and activation distributions
  - Baseline quality
- **Calibration data**
  - Representative prompts
  - Short and long contexts
  - Business-task samples
- **Quantization method**
  - GPTQ / AWQ
  - SmoothQuant
  - FP8 / FP4 / NF4
- **Quantization analysis**
  - Outlier detection
  - Layer sensitivity
  - Group size
- **Quantization artifacts**
  - INT4 / INT8 / FP8 weights
  - Scale factors / zero points
  - Key–value cache quantization
- **Inference runtime**
  - Dequantization + general matrix-multiplication fusion
  - Tensor cores / compute kernels
  - vLLM / TensorRT-LLM
- **Evaluation and fallback**
  - Perplexity / task sets
  - Throughput / latency / GPU memory
  - High-precision fallback

Figure 2-14 Model quantization process and inference deployment chain

2. Industry Progress

A 2025 survey on low-bit large language models organized LLM quantization from three levels: basic numerical formats, system support, and algorithmic strategies. It pointed out that low-bit quantization already covers weights, activations, gradients, and inference runtime, and that PTQ, QAT, mixed precision, and hardware adaptation need to be considered jointly [137]. This shows that model quantization is not a single compression algorithm, but rather a deployment system jointly determined by model formats, calibration data, compute kernels, inference frameworks, and target hardware.

Industry progress directions

1. **Post-training weight quantization**
   - GPTQ / AWQ / INT4
2. **Joint weight-activation quantization**
   - SmoothQuant / W8A8 / FP8
3. **Low-bit fine-tuning**
   - QLoRA / NF4 / LoRA adapters
4. **Engine and hardware adaptation**
   - TensorRT-LLM / vLLM / kernel fusion

Production implementation process

A. **High-precision base model** - FP16 / BF16

B. **Calibration data and sensitivity analysis** - Representative samples - Outlier values - Group size

C. **Quantization-method selection** - GPTQ - AWQ - SmoothQuant - FP8

D. **Deployment of quantization artifacts** - INT4 / INT8 - FP8 - Key-value cache quantization

E. **Effectiveness and performance validation** - Accuracy degradation - GPU memory / throughput - First-token latency

F. **Fallback and gray-release strategy** - High-precision fallback - Scenario admission - Continuous monitoring

Low bit width does not necessarily mean faster end-to-end performance; gains depend on hardware, kernels, and task quality.

From model compression toward low-bit formats, engine kernels, and hardware co-design

Figure 2-15 Industry progress in model quantization

Mainstream practice in large-model quantization first focused on post-training weight quantization. GPTQ uses approximate second-order information to perform layer-by-layer, block-wise quantization, enabling models with tens of billions to hundreds of billions of parameters to maintain relatively low accuracy loss under 3-bit or 4-bit weights [138]. AWQ identifies weights that are more critical to model outputs from an activation-aware perspective.

channels, and improve the generalization stability after low-bit quantization by protecting a small number of key weights [139]. These methods have promoted the development of the open-source ecosystem for 4-bit model inference, and have also made it possible to deploy larger models on a single machine or with fewer graphics processing units.

Weight-activation joint quantization focuses more on hardware acceleration. SmoothQuant observes that activation outliers in LLMs make activation quantization difficult; therefore, it transfers the difficulty of quantization from ac-

tivations to weights, achieving post-training quantization in the W8A8 format and facilitating the use of INT8 matrix multiplication on hardware [140]. On the new generation of graphics processors, FP8 has also become an important low-precision format. Frameworks such as TensorRT-LLM and vLLM provide inference support around methods including FP8 W8A8, FP8 key–value caching, INT4 W4A16, AWQ, and GPTQ [141][142]. This indicates that model quantization has moved from algorithmic papers toward engineering recipes tailored to specific hardware and inference engines.

Low-bit fine-tuning expands the application boundaries of quantization. QLoRA combines a frozen 4-bit base model with LoRA adapters, and significantly reduces the GPU-memory requirements of fine-tuning through NF4, double quantization, and paged optimizers [143]. A systematic evaluation of quantization technologies in 2025 also suggests that the differences among low-bit methods arise not only from bit width, but also from calibration data, grouping granularity, outlier handling, dequantization kernels, and evaluation-task settings [144]. In enterprise scenarios, base models can be deployed in low-bit form, and different business capabilities can be managed through lightweight adapters; however, at runtime, issues such as adapter composition, compatibility of quantization formats, precision fallback, and multi-tenant resource isolation must also be handled.

From a production perspective, model quantization requires the establishment of a complete validation chain. First, representative calibration data should be selected, covering short question answering, long contexts, code, structured outputs, and high-risk tasks. Second, outliers in each layer, sensitive layers, and group sizes should be analyzed, and GPTQ, AWQ, SmoothQuant, FP8, or other routes should be chosen. Thereafter, end-to-end metrics must be validated on the target inference engine and target hardware, including GPU-memory usage, throughput, time to first token, per-token latency, long-context stability, and degradation in task accuracy. Only when the quantization format, compute kernels, engine, and hardware form a closed loop can quantization be stably converted into token-cost optimization, rather than being reflected only in the size of the model files.

(IX) Model Distillation

1. Technical Definition

Model distillation, in the context of the era of large language models, refers to a series of technologies and engineering practices that transfer the knowledge, reasoning capabilities, alignment preferences, or complex-task-solving skills of a teacher model with a huge number of parameters and high computational cost into a student model with fewer parameters and lower inference cost. Distillation in traditional deep learning was often limited to fitting probability distributions in classification tasks, whereas in the era of large models, the concept of distillation has been greatly generalized. Its core objective is not only to make a small

model output the same results as a large model, but also, under constraints on parameter count, to achieve efficient compression of knowledge, faithful reproduction of logical reasoning chains, and deep internalization of complex agent skills.

From the perspective of technical drivers, model distillation mainly addresses problems such as latency in edge-side deployment, the cost of cloud services (for example, improving throughput through frameworks such as vLLM), and the extreme optimization of capabilities in specific vertical domains. Seeking the Pareto optimum among accuracy, GPU-memory usage, inference speed, and training cost is the central thread in the evolution of distillation technology.

From the perspective of implementation paradigms, current large-model distillation is mainly divided into the following five technical levels:

White-box distillation / logical feature matching is the most classical form of distillation, requiring developers to have access to the weights of both the teacher and student models. By minimizing the KL divergence between the output vocabulary probability distributions of the two models, or by aligning the hidden states and attention matrices of intermediate layers, the student model is made to approximate the teacher model in feature space. This method has high information-transfer bandwidth, but due to the widespread adoption of closed-source large-model APIs, in the current

In cutting-edge applications, it is often used for size scaling among homologous models—for example, distilling from 70B to 8B.

Black-box distillation / data synthesis based on prompting and generation: when the parameters of the teacher model cannot be obtained, powerful closed-source models such as GPT-4 and Claude 3.5 are treated as data generators. Through carefully designed instructions, the teacher model is induced to generate high-quality question–answer pairs, code snippets, or multi-turn dialogues, which are then used as synthetic data for supervised fine-tuning of the student model. The core challenges of this paradigm lie in controlling the diversity of synthetic data, filtering noise, and addressing distribution shift.

Distillation of reasoning processes and chains of thought: for mathematical computation, code generation, and complex logical reasoning, merely having the student model learn the final answer often leads to "knowing what is so without knowing why," thereby producing severe hallucinations. Reasoning distillation extracts high-quality chains of thought, multi-step solution trajectories, or verification-and-reflection processes generated by the teacher model, and constructs triplet data consisting of instructions, reasoning processes, and final answers. This enables the student model to implicitly learn the logical patterns of task decomposition and step-by-step solution, and even to internalize part of the optimal strategies discovered through reinforcement-learning exploration.

Internalization of agent skills and memory compression: as models evolve toward agents, the goal of distillation has expanded from simple text generation

to environmental interaction. Optimization at this level focuses on extracting the successful trajectories obtained by the teacher model through trial and error, tool invocation, and multi-step planning in complex environments such as ALFWorld and WebShop. By "compressing" these global memories and dynamic interaction experiences into high-quality SFT data, the student model can internalize specific execution actions and decision flows without relying on an enormous number of parameters. This allows small-scale models to acquire task decomposition and state-machine transition capabilities similar to those of large models.

Online policy distillation and correction of distribution shift: traditional SFT-based instruction distillation is essentially an "offline policy," in which the student model passively learns static data generated by the teacher model. This approach suffers from serious "distribution shift" or "exposure bias" : during training, the student model predicts the next token under the teacher' s perfect contextual prefix; but during actual inference, the student must continue generation based on prefixes it has generated itself, which may contain small errors. One wrong step leads to further wrong steps, causing errors to accumulate continuously. Online policy distillation instead requires the student model to actively generate outputs during training—that is, to sample according to its own policy—and then has a powerful teacher model, or a verifier/reward model trained on teacher data, correct, score, or provide comparative preferences for these "trajectories from the student' s perspective." This approach forces the student model to explore within the space of its own probability distribution, learning how to cope with and correct its own specific errors. Mathematically, this represents an important shift from minimizing forward KL divergence, which requires the student to comprehensively cover the teacher' s output distribution, to reverse KL divergence, which encourages the student to seek high-certainty answers within the scope of its own capabilities, thereby greatly improving the model' s generation quality in real-world scenarios.

In summary, large-model distillation optimizes "how to efficiently acquire, clean, and inject high-quality data, and how to enable small models to inherit the generalization and reasoning ceiling of large models through alignment algorithms." In engineering practice, it is often necessary to combine distributed training frameworks such as Megatron-LM, efficient inference post-processing mechanisms, and dynamic sampling strategies, so as to form a complete closed loop from data synthesis to SFT training and then to evaluation-driven iteration.

2. Industry Progress

The development trajectory of large-model distillation technology is closely intertwined with the evolution of paradigms in natural language processing. From early feature alignment, to the explosion of instruction-tuning data, and then to today' s deep transfer oriented toward reinforcement learning and complex reasoning capabilities, the industry has gone through several notable stages.

Distillation drivers

- **End-side deployment**: reduce latency / GPU-memory footprint
- **Cloud-service cost**: increase throughput / reduce inference cost (*vLLM*, etc.)
- **Vertical-domain optimization**: maximize specific capabilities
- **Optimal solution under resource constraints**: balance accuracy × cost × efficiency

I. Technical definitions

Five technical levels *(from lower to higher)*

1. **White-box distillation (White-box): logical feature matching**
   - Access the weights of both teacher and student models
   - Align logits / hidden states / attention matrices
   - Minimize KL divergence and feature-space distance
   - **High information bandwidth; suitable for scaling models of the same family**
2. **Black-box distillation (Black-box): data synthesis based on prompts and generation**
   - The teacher model serves as a data generator *(closed source)*
   - Carefully design prompts to synthesize high-quality data
   - Train the student model with SFT
   - **Challenges: data diversity, noise filtering, distribution shift**
3. **Reasoning-process and chain-of-thought distillation: CoT / Reasoning Distillation**
   - Extract the teacher model' s chain of thought / solution trajectory
   - Construct triples of *(instruction, reasoning process, answer)*
   - Learn task decomposition and step-by-step solving logic
   - **The reasoning process is the highest-quality distillation data**
4. **Internalization of agent skills and memory compression: Agent Skill Internalization**
   - Extract successful trajectories in complex environments *(tool use / planning / interaction)*
   - Compress global memory and dynamic experience into SFT data
   - Internalize execution actions and decision flows
   - **Compress agent experience into transferable skills**
5. **Online policy distillation (On-Policy) and distribution-shift correction**
   - The student model samples and generates according to its own policy
   - The teacher / verifier performs real-time correction and scoring
   - Correct distribution shift / exposure bias
   - Forward KL → Reverse KL
   - **Mitigate distribution shift and improve generation quality in real scenarios**

Core objectives

Knowledge-efficient compression | faithful replication of reasoning chains | deep internalization of complex skills | improved generalization and robustness *(distribution-shift correction)* | seeking the Pareto optimum among accuracy, GPU memory, speed, and training cost

II. Industry progress *(evolution timeline)*

1. **Early stage: classic distillation based on logits and feature alignment**
   ~2015–2019
   - Relied on white-box information; achieved compression and transfer among models of the same family through probability-distribution and feature alignment
   - **Knowledge Distillation** *(proposed)*
   - **DistilBERT** *(compressed BERT by 40%)*
   - **MiniLM** *(aligned attention)*
   - **Laid the foundation for distillation; high information bandwidth, but dependent on homogeneous architectures and white-box access**
2. **Explosion of black-box instruction distillation: Self-Instruct and imitation learning**
   2020–2022
   - Used teacher models to generate high-quality instruction data for large-scale SFT fine-tuning
   - **Self-Instruct** *(autonomously generated instruction data)*
   - **Alpaca / Vicuna**, etc.
   - **The False Promise** *(pointed out limitations)*
   - **Lowered the threshold for data acquisition and promoted the prosperity of the open-source ecosystem, but black-box distribution shift and superficial imitation emerged**
3. **From imitation to explanation: chain-of-thought and step-by-step reasoning distillation**
   2022–2024
   - Learned the teacher model' s reasoning process, improving complex reasoning and generalization ability
   - **Distilling Step-by-Step** *(using rationales, reducing data requirements)*
   - **Orca series** *(explanation tuning)*
   - **Established that "the reasoning process is the optimal distillation data," significantly improving complex-task and generalization capabilities**
4. **Toward the Agent era: skill internalization and trajectory compression**
   2023–2024
   - The distillation target expanded to environment interaction and tool use, internalizing agent skills
   - **AgentTuning** *(trajectory → SFT)*

- **Trajectory compression and skill abstraction**
- **Enabled small models to acquire complex-task execution and planning capabilities, but was constrained by trajectory quality and noise control**

5. **Current focus: post-training and reinforcement-learning distillation**
   2024-2025+
   - Injected high-quality long-chain thinking and policies discovered by RL into small models, breaking through the upper limits of reasoning
   - **DeepSeek-R1 / R1-Zero** *(RL distillation)*
   - **Magpie** *(autonomous data-generation framework)*
   - **Injecting high-quality long-chain thinking and policies into small models pushes reasoning toward a higher ceiling**

Online exploration for distribution shift (Off-Policy → On-Policy)

- **MiniLLM** *(reverse KL, reduces hallucination)*
- **GKD** *(guidance on student trajectories)*
- **RLAIF and online iterative DPO** *(real-time feedback and preference alignment)*
- **Exploration within the student distribution + real-time feedback significantly improves quality in real scenarios**

III. Complete loop for engineering practice

- **Data synthesis**
  - Instruction design / trajectory collection
  - Quality screening / denoising
- **Data cleaning and filtering**
  - Deduplication / denoising / distribution alignment
  - Difficulty stratification / bias removal
- **SFT training**
  - Distributed training *(Megatron-LM, etc.)*
  - Mixed precision / efficient optimization
- **Post-training (optional)**
  - RLHF / DPO / PPO / ORPO
  - Reasoning enhancement / self-reflection
- **Evaluation and iteration**
  - Benchmark evaluation / business evaluation
  - Error analysis / data feedback loop
- **Efficient inference deployment**
  - vLLM / TensorRT-LLM
  - Quantization / KV Cache / parallel inference

IV. Development-thread summary

Probability fitting *(logits / feature alignment)* → behavioral imitation *(instruction and output imitation)* → logical parsing *(chain-of-thought and reasoning distillation)* → skill internalization *(agent trajectory compression)* → reinforcement-

learning policy transfer *(RL exploration capability injection)* → online policy distillation *(distribution-shift correction)*

Core objective (future)

Against the backdrop of a gradually converging resource-capability ceiling, compress the knowledge and long-range logical structures of large agent systems to the limit and deploy them on end-side devices.

Figure 2-16 Industry evolution of model distillation

1. **Early stage: classic distillation based on logits and features**

In the era of pretrained language models such as BERT and RoBERTa, the main form of distillation was white-box knowledge transfer. Hinton et al. formally proposed the concept of knowledge distillation in 2015. By introducing a temperature parameter to soften the output probability distribution, they demonstrated the transfer potential of "dark knowledge" [145]. Subsequently, the industry saw a large amount of optimization work targeting Transformer architectures. For example, DistilBERT compressed the BERT model by 40% by aligning the output distributions in the pretraining stage, while retaining 97% of its language-understanding capability [146]. MiniLM further proposed a method for aligning the depth-value and value/query-key matrices of the self-attention module, achieving deeper structural compression [147]. Distillation at this stage focused on model compression and was highly dependent on homogeneous architectures.

2. **The explosion of black-box instruction distillation: Self-Instruct and imitation learning**

With the release of ultra-large instruction-fine-tuned models such as ChatGPT, the open-source community found that direct access to high-dimensional logits was no longer possible. The industry focus quickly shifted toward generation-based data-synthesis distillation. The Self-Instruct framework became the foundational work of this period: it demonstrated how, with a small number of seed prompts, a powerful teacher model could be used to autonomously generate large volumes of diverse instruction-following data, and how these data could then be used to fine-tune high-performing small models [148]. Based on this paradigm, models such as Stanford Alpaca and Vicuna rapidly narrowed the gap between open-source small models and closed-source giants in everyday conversational experience.

However, researchers soon pointed out the limitations of pure black-box imitation. The paper *The False Promise of Imitating Proprietary LLMs* argued through rigorous evaluation that, when SFT relies only on the superficial responses output by a teacher model, the student model often learns only the tone and style of the large model, rather than truly acquiring deep reasoning and factual knowledge; it may even perform worse on unseen tasks [149]. This finding prompted the industry to begin exploring deeper distillation mechanisms.

3. **From imitation to explanation: chain-of-thought and step-by-step reasoning distillation**

To overcome the bottleneck of "knowing what is so but not knowing why it is so," the industry began to shift the goal of distillation toward the reasoning process itself. Google' s Distilling Step-by-Step method used explanations generated by the teacher model as additional supervision signals and, while greatly reducing the amount of training data required, significantly improved the performance of small models on natural-language reasoning tasks [150].

Microsoft' s Orca series pushed explanation tuning to a new peak. Orca not only asks the teacher model to provide answers, but also requires it to output in detail the step-by-step thinking process, the logic by which system instructions are followed, and the reasoning path. By learning such trajectories containing rich causal relationships, Orca-13B' s performance on complex tasks

has even come close to larger-scale foundation models [151]. This stage established the consensus that "the reasoning process is the highest-quality distillation data."

4. Toward the Agent Era: Skill Internalization and Trajectory Compression

After large language models are endowed with tool-use and multi-step environment-interaction capabilities, how to distill these agentic skills into small models has become a frontier focus. Work such as AgentTuning has shown how powerful large language models can be used to explore multiple virtual environments and tool APIs, collect successful interaction trajectories, and then convert them into SFT datasets through specific formatting methods, thereby endowing open-source small models with zero-shot agent capabilities [152].

In this area, the recent core challenge lies in the signal-to-noise ratio problem of "memory compression" and "skill distillation." When large models interact with environments, they often produce large amounts of trial and error, redundant reflection, and backtracking operations; if all of these are accepted wholesale, the student model may learn to "hesitate" and "overthink." Therefore, studies similar to skill abstraction and trajectory pruning have continued to emerge, aiming to extract the most essential skill primitives so that, under limited-parameter conditions, the model can precisely internalize specific vertical scenarios, such as code execution and multi-step question answering.

5. Current Focus: Post-Training, Reinforcement Learning, and Distillation of Frontier Reasoning Models

From the end of 2024 into 2025, as the OpenAI o1 series and DeepSeek-R1 series shifted the focus toward test-time computation and reinforcement learning, the distillation paradigm underwent another leap. The release of DeepSeek-R1 not only demonstrated that large-scale pure reinforcement learning can spontaneously give rise to powerful long-chain reasoning capabilities; through the

open-source distilled versions of R1-Zero, such as dense models based on the Qwen and Llama architectures, it also proved to the industry that high-quality chains of thought discovered by ultra-large models through reinforcement learning can be efficiently "injected" directly into small models with only several billion parameters via SFT, enabling them to achieve above-tier performance on mathematics and code benchmarks [76].

At the same time, self-generated data frameworks such as Magpie have further improved the distillation process for alignment data without human intervention. By cleverly leveraging the autoregressive properties of pretrained models, Magpie can directly batch-extract high-quality dialogue and reasoning instructions from teacher models without guidance from specific prompts, greatly lowering the threshold for constructing distillation datasets [153].

For online exploration under distribution shift, the industry has found that offline imitation relying purely on static data can easily cause small models to develop "overconfident hallucinations,"and that they are highly prone to collapse when encountering out-of-distribution prompts. Therefore, online policy distillation has gradually become an indispensable path for leaps in model capability. MiniLLM incisively revealed this pain point: forcing a small model to fit the large and flat output probability distribution of a large model leads to degraded generation quality. It therefore introduces reverse KL divergence and optimizes on text trajectories generated by the student model itself based on an online policy, so that the student no longer blindly imitates all of the teacher' s wording habits, but instead chooses high-confidence expressions that both conform to the teacher' s logic and lie within the student' s own capability boundaries, significantly reducing the hallucination rate [154]. GKD, in turn, proposes a generalized distillation framework in which the student model first generates output trajectories, and the teacher model then provides real-time probability-distribution guidance for these student trajectories, effectively bridging the gap between the training and inference stages [155]. In the preference-alignment stage, online policy distillation further evolves into the paradigms of RLAIF and online iterative DPO: the student model generates candidate answers online, a closed-source large model scores them to construct preference data pairs, and the model is then updated through reinforcement learning. This is precisely the key means by which today' s leading open-source models achieve logical refinement and alignment during post-training [156].

However, the reasoning-distillation work described above generally relies on the rejection-sampling paradigm: the teacher model generates multiple reasoning trajectories for a given problem, and only the correct trajectories are retained for student training. This process treats the teacher model as a static filter: for complex boundary problems that the teacher itself cannot solve independently, all sampled trajectories are marked as unsolvable and discarded, so the student model' s training data are mainly composed of simple and medium-difficulty samples, creating an artificial teacher ceiling—the upper limit of the student' s ability is constrained by the teacher' s independent problem-solving ability.

Empirical probing by China Unicom shows that even when the sampling budget is expanded to $N = 64$, the Qwen3-32B teacher model still for

For 13% of the difficult AIME problems, no valid trajectory could be generated at all.

To address this bottleneck, China Unicom proposed the HEAL (Hindsight Entropy-Assisted Learning) framework [157]—a distillation method for reasoning ability that does not require reinforcement learning. Its core idea draws on the "Zone of Proximal Development" (ZPD) theory in education: when a teacher model fails independently on a difficult problem, this does not mean that it lacks the relevant latent reasoning ability; it may only need a prompt to enter the correct search space. HEAL reconstructs the distillation process along three dimensions: active synthesis, quality filtering, and progressive training.

The GEAR (Guided Entropy-Assisted Repair) module implements reasoning repair based on entropy dynamics. It simulates the self-repair process of an expert teacher: first attempting to solve the problem independently, and consulting the answer only when stuck. GEAR monitors changes in the entropy gradient during reasoning and identifies the step with the largest entropy surge as a "critical reasoning breakpoint" —that is, the boundary at which the model' s cognition moves from certainty to uncertainty. At the breakpoint, the standard answer is injected as a local prompt, guiding the teacher to continue reasoning from the last stable logical state and repair the fractured reasoning trajectory. The search range is constrained to the first third of the sequence, because in complex mathematical reasoning, early logical deviations often cause subsequent reasoning to fail irreversibly; what GEAR precisely repairs are exactly these fundamental structural errors.

The PURE (Perplexity-Uncertainty Ratio Estimator) module addresses the risk of "logical shortcuts" introduced by answer prompts—reasoning that is grammatically coherent but logically discontinuous, such as "since the reference answer is 36, the final answer is 36." PURE identifies shortcuts by computing the "suspicion ratio" at each step: if a step has high perplexity but low answer uncertainty, it suggests that the step may be a forced logical jump. PURE uses the global peak of the suspicion ratio as the anomaly score of the trajectory, removes the top 20% most severe shortcut trajectories, and ensures the logical integrity of the distillation data.

The PACE (Progressive Answer-guided Curriculum Evolution) module organizes the distillation process into a three-stage progressive curriculum: in the first stage, "basic alignment," training is performed only on D_{base}, obtained through independent sampling by the teacher; in the second stage, "latent expansion," D_{hint}, sampled under global answer prompts, is introduced and mixed with the basic data for training; in the third stage, "frontier breakthrough," D_{repair} data repaired by GEAR is added and upsampled, enabling the student to overcome the hardest reasoning breakpoint problems. This easy-to-hard strategy avoids training instability and the forgetting of difficult cases.

The practice of HEAL shows that by actively intervening in the reasoning process of the teacher model, repairing its cognitive breakpoints, filtering logical shortcuts, and progressively transferring knowledge, the student model can break through the traditional teacher ceiling imposed by refusal-to-sample settings and achieve leapfrog improvements in reasoning ability. In practice for token-oriented operations, China Unicom used HEAL distillation technology as a key means of "high-quality knowledge refinement," successfully compressing the capabilities of large models into lightweight models deployable on enterprise-grade servers and substantially reducing the invocation cost of the MaaS platform.

Overall, large-model distillation is moving further along the path of "probability fitting → behavior imitation → logical parsing → skill internalization → reinforcement-learning strategy transfer." The "active repair distillation" paradigm represented by China Unicom' s HEAL is driving this path from static knowledge transfer toward dynamic cognitive elicitation—not simply screening for problems that the teacher has already solved, but actively helping the teacher break through its independent problem-solving boundary, thereby transforming previously discarded difficult problems into high-quality distillation signals. In the future, as the capability ceiling of closed-source models gradually converges, how to use more efficient distributed training architectures and cleaning algorithms to compress the knowledge and long-range logical structures of vast intelligent systems to edge devices, and how to migrate HEAL' s entropy-guided repair mechanism to self-distillation and online reinforcement-learning paradigms, will be core challenges to tackle continuously.

III. Model-Computing Fusion

Model–compute fusion is a key technical direction in optimizing the inference performance of large models. Its core lies in jointly optimizing model structural features, operator execution modes, memory-access paths, and inference-engine scheduling mechanisms, so as to achieve efficient matching between model algorithms and underlying computing resources. This chapter focuses on five aspects: operator fusion, memory-access optimization, acceleration of basic operators, tuning of engine parameters, and adaptation of model architectures. Among them, model-architecture adaptation is the prerequisite for model–compute fusion; it determines how an inference system identifies attention mechanisms, MoE expert layers, positional encodings, quantization formats, and multimodal components in different models. Operator fusion and acceleration of basic operators are core means for improving the efficiency of a single computation, respectively starting from reducing intermediate tensor reads and writes, lowering scheduling overhead, and improving the execution efficiency of high-frequency operators such as GEMM, attention, Softmax, and normalization. Memory-access optimization targets the memory bottlenecks brought by key-value caches, long contexts, and high concurrency in large-model inference, improving resource utilization through cache management, data layout, hierar-

Figure 3-1. Schematic of fusion patterns for Transformer attention subgraphs; cited from Reference [160]

Figure 1: Figure 3-1. Schematic of fusion patterns for Transformer attention subgraphs; cited from Reference [160]

chical storage, and access-path optimization. Engine-parameter tuning further optimizes parameters such as parallel strategies, batching, prefill and decoding scheduling, and key-value cache quotas at the service runtime level, in order to balance time to first token, throughput, stability, and cost. Overall, these five categories of technologies together form a complete technical chain from "model-structure identification—operator and memory optimization—inference-engine scheduling—service-oriented deployment evaluation," and constitute an important foundation for supporting high performance, low cost, and stable launch of large models in heterogeneous computing environments.

(I) Operator Fusion

1. Technical Definition

Operator fusion refers to identifying mergeable adjacent operators or subgraphs in a deep-learning computation graph and compiling them into a fused operator or fewer execution units, so as to reduce intermediate tensor reads and writes, kernel launches, synchronization, and scheduling overhead. oneDNN Graph defines operator fusion as follows: it accepts a complete computation graph, performs graph partitioning oriented toward the execution engine, and compiles candidate fused subgraphs to execute as fused operations. TVM also lists high-level operator fusion, hardware mapping, and memory-latency hiding as core optimization problems for deep-learning compilers.

2. Industry Progress

Operator fusion has evolved from regularized graph fusion in early convolutional neural-network scenarios—such as Conv+BatchNorm+ReLU, Conv+Add, GELU, LayerNorm, and SkipLayerNorm—into a multilayer optimization system covering graph compilers, operator libraries, inference engines, and service runtime. ONNX Runtime divides graph optimizations into levels such as Basic, Extended, and Layout; among them, Extended optimization includes fusions such as GELU, LayerNorm, attention, SkipLayerNorm, and BiasGELU. oneDNN Graph identifies candidate subgraphs through graph partitioning oriented toward the execution engine and compiles and executes fused partitions [158].

Figure 3-1. Schematic of fusion patterns for Transformer attention subgraphs; cited from Reference [160]

In Transformer and large language model inference scenarios, attention has become the core target of fusion optimization. TensorRT supports constructing

Figure 3-2

Figure 2: Figure 3-2

attention through the IAttention API or through MatMul, Softmax, and other operations.

computation graph to trigger MHA fusion, and supports GQA/MQA scenarios. Its documentation explicitly states that, for long sequences, MHA fusion can reduce the memory footprint of intermediate attention results from $O(S^2)$ to $O(S)$, while also reducing memory traffic, kernel launches, and synchronization overhead. TensorRT-LLM further distinguishes between the prefilling stage and the decoding stage for autoregressive GPT-like models: in the prefilling stage, FMHA can execute MHA/MQA with a single operator, and for long sequences it adopts the ideas of FlashAttention/FlashAttention-2; in the decoding stage, it uses a masked MHA operator and supports decoding-oriented optimizations such as XQA, paged key–value cache, and INT8/FP8 key–value cache.

Figure 3-2 Process of constructing an attention computation graph through the IAttention API to trigger MHA fusion; cited from Reference [141]

Attention fusion has developed into a specialized kernel system that is IO-aware, hardware-aware, and precision-aware. FlashAttention reduces reads and writes between HBM and on-chip SRAM through tiling and blocking [44]; FlashAttention-2 improves parallelism and work partitioning [45]; FlashAttention-3 introduces asynchronous execution, warp specialization, and FP8 for Hopper [159].

NVIDIA cuDNN Frontend also treats SDPA/FlashAttention-type kernels as a key capability [160]. Its SDPA operation uses the FlashAttention-2 algorithm, supports both training and inference, and covers paths such as FP16/BF16 forward/backward and FP8 forward/backward; the cuDNN Frontend README also lists high-performance kernel directions such as SDPA/Flash Attention, MoE Grouped GEMM fusions, and fused normalization + activation.

MoE and MLP subgraphs have also become hotspots for fusion. In the MLP of a traditional Transformer, the commonly seen linear layer + bias + GELU/SwiGLU + linear layer can reduce intermediate tensor reads and writes through methods such as epilogue fusion, GEMM + activation fusion, and GEMM + SwiGLU fusion. In MoE scenarios, the key lies in coordinated optimization among token routing, permute/unpermute, expert grouping, Grouped GEMM, and activation functions. cuDNN Frontend already provides Grouped GEMM + SwiGLU fusion for Blackwell SM100+ [161].

Schematic diagram of the PagedAttention algorithm

Figure 3: Schematic diagram of the PagedAttention algorithm

(II) GPU-Memory Access Optimization

1. Technical Definition

GPU-memory access optimization refers to optimizing data layout, access order, access granularity, cache reuse, and data-movement paths around hierarchical storage resources such as graphics-processor global memory/HBM, L2/L1 cache, shared memory, registers, and, when necessary, CPU DRAM, solid-state drives, networks, and key–value caches, so as to reduce invalid mem-

memory transactions, redundant reads and writes, GPU-memory fragmentation, cross-hierarchy copying, and bursty out-of-memory (OOM) events. Typical methods include contiguous and aligned access, warp-level coalesced access, tensor-layout rearrangement, tiling/blocking, kernel fusion, shared memory, asynchronous copying with overlap of computation, block-based key-value cache management, prefix-cache reuse, GPU-memory pooling, and hierarchical cache management.

2. Industry Progress

In large-model inference, the importance of optimizing GPU-memory access continues to rise. The reason is that inference for decoder-only large language models can be divided into two types of workloads: prefill and decoding. Prefill processes the complete input sequence and is more compute-intensive; decoding generates tokens autoregressively, and at every step it must read the historical key-value cache. As context length, the number of concurrent requests, and batch size grow, the capacity occupied by the key-value cache and its read bandwidth quickly become bottlenecks. Quantitative studies of the key-value cache show that the key-value cache not only significantly increases memory demand, but its loading process may also leave compute cores idle, thereby limiting inference speed [112].

At the key-value cache management level, PagedAttention draws on the operating system' s paging idea: it partitions the key-value cache for each request into fixed-size key-value blocks, allowing logically contiguous key-value caches to be stored in non-contiguous physical GPU memory, and supporting sharing of the key-value cache both within and across requests. The vLLM paper reports that its goal is to achieve near-zero waste of key-value-cache memory and flexible sharing, and that at the same latency level it brings a 2–4× throughput improvement over systems such as FasterTransformer and Orca [48].

Figure 3-3 Schematic diagram of the PagedAttention algorithm; adapted from Ref. [48]

SGLang proposes RadixAttention for scenarios such as complex programs, multi-round calls, few-shot prompting, retrieval-augmented generation, and multi-turn dialogue. It organizes reusable prefix key-value caches to avoid repeated prefill computation. The paper states that, at runtime, key-value cache reuse is performed through RadixAttention, achieving up to a 6.4× throughput improvement over existing inference systems across multiple types of tasks [108].

FlashInfer is an attention engine for large language model inference services. Targeting the heterogeneity of key-value cache storage, it introduces block-sparse formats and composable formats to optimize memory access and reduce redundancy, and uses JIT templates to adapt to different attention variants. At the same time, its load-balancing scheduling algorithm accommodates both dynamic requests and the static-configuration requirements of CUDAGraph. The paper reports that FlashInfer has been integrated into systems such as SGLang and vLLM, achieving a 29%–69% reduction in inter-token latency and a 28%–30% latency reduction for long-context inference across different inference scenarios [162].

Mooncake designs the service architecture as a decoupled architecture centered on the key-value cache. It separates the prefill and decoding clusters, and uses underutilized resources in the GPU cluster—such as central processing units, dynamic random-access memory, and solid-state drives—to build a disaggregated key-value cache. The paper reports that, in long-context scenarios, throughput can be improved by up to 525% over the baseline, and that under real workloads it enables Kimi to process requests

increased by 75% [163].

Low-precision and compressed key-value caching is another main line of work for long-context and high-concurrency inference. KIVI analyzes the distribution of KV-cache elements and proposes an asymmetric 2-bit scheme in which the key cache is quantized by channel and the value cache by token. The paper reports that, on Llama, Falcon, and Mistral, the method can reduce peak memory by a factor of 2.6 while essentially preserving quality, increase the maximum batch size by up to 4 times, and deliver a 2.35–3.47× throughput improvement [112]. TurboQuant further formulates KV-cache compression as an online vector-quantization problem. By using random rotation, scalar quantization, and 1-bit Quantized Johnson–Lindenstrauss residual correction to reduce inner-product estimation bias, it reports quality-neutral KV-cache quantization at 3.5 bits/channel, and only slight quality degradation at 2.5 bits/channel [114].

(III) Basic Operator Acceleration

1. Technical Definition Basic operator acceleration refers to optimizing the high-frequency, general-purpose, and performance-sensitive basic computational units in deep learning models, such as GEMM/MatMul, convolution, attention, Softmax, normalization, pooling, activation functions, and tensor transformations. NVIDIA cuDNN is defined as a GPU-accelerated library of basic opera-

tors for deep neural networks, providing highly optimized implementations of standard routines such as convolution, attention, MatMul, pooling, and normalization. cuBLAS, in turn, is a GPU-accelerated BLAS/GEMM library for AI and HPC, supporting batched processing, multi-GPU execution, mixed- and low-precision execution, and fusion.

2. Industry Progress Industry-level basic operator acceleration has evolved from "calling a single high-performance library" into a layered system comprising "standard libraries, compilers/inference engines, model-structure-aware kernels, and coordinated optimization of communication operators." At the bottom layer, cuBLAS, cuDNN, CUTLASS, and related libraries remain central: cuDNN covers deep neural network operator libraries such as convolution, attention, MatMul, normalization, Softmax, pooling, and pointwise operations; cuBLAS focuses on serving GEMM and MatMul, supporting batched processing, multi-GPU processing, mixed precision, low precision, and fused GEMM; and CUTLASS provides customizable CUDA building blocks for GEMM, convolution, and tensor computation [164][165][166].

Large-model inference has further shifted the focus of operator optimization. In the prefill stage, sequence lengths and matrices are large, placing greater emphasis on the throughput of GEMM/attention and on HBM bandwidth utilization. In the decoding stage, batches are small and tokens are generated one by one, so the emphasis shifts more toward GEMM, low-latency scheduling, CUDA Graph, and memory-access efficiency. MoE models introduce new critical paths, including token routing, Grouped GEMM, and expert load balancing. The DeepSeek-V3 technical report shows that DeepSeek-V3 adopts a large-scale MoE architecture and combines FP8 mixed-precision training, DualPipe, cross-node All-to-All communication optimization, and computation-communication overlap to reduce training and inference costs. This indicates that, in MoE scenarios, "basic operator acceleration" also includes communication kernels, load balancing, and low-precision data movement [42].

A development that has attracted particular attention in the past two years is DeepSeek DeepGEMM [167]. DeepGEMM was initially open-sourced as a CUDA Kernel library for FP8 GEMM, used to support dense GEMM and MoE Grouped GEMM in DeepSeek V3/R1 training and inference. By April 2026, its README described it as a unified high-performance Tensor Core Kernel library covering key basic operators for large language models, including FP8, FP4, and BF16 GEMM, MoE, MQA scoring, and HyperConnection, and using lightweight JIT to compile kernels at runtime.

DeepGEMM' s core value lies in the fact that it is a structure-aware kernel for large language models / MoE-form models. Its Grouped GEMM design groups only along the M dimension, while N/K remain fixed, making it suitable for MoE scenarios in which multiple experts share the same weight shape but each expert receives a different number of tokens: during prefill/training, a contiguous memory layout can be adopted, concatenating the tokens of different experts

into a continuous tensor; during decoding, under a CUDA Graph scenario, a masked layout can be used, with a mask indicating the valid token intervals.

DeepGEMM also has clear boundaries: at present, it is mainly oriented toward NVIDIA SM90/SM100 architectures; users need to handle FP8 type conversion, input transposition, and partial pre-operation fusion on their own. The contiguous memory layout of Grouped GEMM imposes M-dimension compute-block alignment requirements on each expert segment, which introduces computation waste when the batch size is small or the load is imbalanced. Around this pain point, the 2025 TMA-adaptive FP8 Grouped GEMM proposed the use of TMA descriptor pools and a two-stage load–store mechanism, eliminating the padding overhead caused by fixed 128 alignment; in experiments, compared with the "padding + DeepGEMM" baseline, it achieved speedups of 1.7%–20.4% and memory savings of up to 23.8%. This indicates that DeepGEMM has become an important baseline for subsequent research on MoE Grouped GEMM [168].

(IV) Engine Parameter Tuning

1. Technical Definition

Engine parameter tuning refers to the systematic optimization, for large-model inference services, of the inference engine' s parallelism parameters and scheduling strategies under given constraints on task type, computing resources, and service-quality objectives, so that model serving achieves an optimal balance among time to first token, throughput, stability, and resource utilization. Its core tuning objects include parameters such as TP, PP, DP, PD disaggregation, prefill block size, and KV-cache quota.

2. Industry Progress

Mainstream industry inference frameworks have already upgraded "engine parameter tuning" from static deployment parameter configuration to a system-level optimization problem. vLLM is one representative in this direction. The vLLM V1 documentation incorporates chunked prefill, TP/PP/DP/EP, NUMA binding, attention backends, and input-processing parallelism into its tuning system; among these, chunked prefill is enabled by default in available scenarios, and the mixing ratio between prefill and decoding is controlled through `max_num_batched_tokens`: smaller values are more favorable for ITL, while larger values are more favorable for TTFT and throughput, but overly large values may degrade end-to-end latency [169].

At the scheduling-algorithm level, Sarathi-Serve proposes splitting the prefill of long prompts into multiple chunks and combining them with decoding requests into more balanced batches, thereby reducing the blocking of decoding by prefill and achieving a better balance between throughput and tail latency [170]. DistServe, from the perspective of system architecture, proposes PD disaggregation: compute-intensive prefill and access-sensitive decoding are assigned to

Diagram labels: LLM; Rank0; PyExecutor(Worker); Scheduler; ModelEngine; KVCacheManager; Sampler; Rank1; PyExecutor; Rank N-1; PyExecutor.

Figure 4: Diagram labels: LLM; Rank0; PyExecutor(Worker); Scheduler; ModelEngine; KVCacheManager; Sampler; Rank1; PyExecutor; Rank N-1; PyExecutor.

different GPU pools, and resource allocation, parallel strategies, and cross-stage placement are optimized separately for TTFT and TPOT, thereby improving effective throughput under SLO constraints [171].

SGLang also extends high-performance runtime capabilities to RadixAttention, a zero-overhead CPU scheduler, PD disaggregation, speculative decoding, TP, PP, EP, DP, and chunked prefill. In the PD-disaggregation documentation, SGLang explicitly defines prefill as a compute-intensive stage and decoding as a memory/KV-cache-intensive stage, and supports KV-cache transfer backends such as Mooncake and NIXL. In addition, SGLang' s tuning documentation provides parameter recommendations for `--mem-fraction-static`, `--chunked-prefill-size`, `--max-running-requests`, `--cuda-graph-max-bs`, `--dp-size`, `--tp-size`, and other parameters.

recommendations, indicating that its tuning priorities already cover queue depth, KV-cache utilization, CUDA Graph, parallelism forms, and prefix-cache hit rate [172].

TensorRT-LLM supports chunked prefill, paged KV cache, `max_batch_size`, `max_num_tokens`, a KV-cache block manager, KV-cache reuse/offloading, quantization, and multiple parallelism strategies. It also explicitly states that `max_num_tokens` affects the trade-off among available GPU memory for the KV cache, throughput, TTFT, and TPOT [173]. NVIDIA Dynamo further incorporates upper-layer inference-engine request routing, PD separation, KV-aware routing, and GPU-to-GPU KV-cache transfer into a distributed inference framework. Its official design document decomposes a disaggregated request into three steps—prefill computation of the KV cache, KV-cache transfer, and decoding computation—and emphasizes that the key to high-performance disaggregated inference lies in efficient KV-cache transfer and routing orchestration [174]. Such systems show that, for large-scale inference clusters, engine-parameter tuning has become a cluster-level optimization problem spanning instance grouping, GPU-pool ratios, network interconnection, KV-cache transfer backends, and cache-routing strategies.

Figure 3-4 Internal structure of the TensorRT-LLM inference engine; reproduced from Reference [173]

TGI provides production capabilities such as continuous batching, SSE streaming output, FlashAttention, PagedAttention, KV cache, quantization, tensor parallelism, and an OpenAI-compatible API. However, its official documentation also states that TGI has entered maintenance mode and recommends using

inference engines such as vLLM or SGLang for newly built inference services [175]. Therefore, in terms of the technical roadmap, TGI should be positioned as an object for "legacy compatibility and baseline comparison," rather than as a focus for subsequent development of tuning capabilities.

In the direction of domestic computing power, Ascend MindIE LLM provides large-model inference capabilities based on Ascend hardware, supporting acceleration features such as multi-concurrent request scheduling, continuous batching, PagedAttention, and FlashDecoding. Its LLM Manager is responsible for state management, task scheduling, unified memory-pool management of the KV cache, and state-monitoring interfaces, while the Modeling layer supports built-in models, Tensor partitioning, and multiple quantization methods [176]. This shows that domestic inference stacks are likewise carrying out systematic optimization around scheduling, caching, GPU-memory pools, and runtime parameters.

(V) Model Architecture Adaptation

1. Technical Definition Model architecture adaptation refers to adapting the inference framework, compiler, operator library, and service-scheduling logic to a specific model structure without changing the model's core semantics—for example, MHA/MQA/GQA/MLA attention, RoPE or ALiBi positional encoding, MoE expert layers, multimodal encoders, LoRA adapters, quantization formats, and custom layers. The Hugging Face Transformers documentation states that model code and configurations can be customized, and users can also add layers or optimize attention mechanisms by modifying model components; TensorRT-

The LLM documentation explicitly supports attention implementations for GPT-like models, including MHA, MQA, and GQA.

2. Industry Progress

With the rapid iteration of Llama, Qwen, DeepSeek, Gemma, multimodal models, and MoE models, inference frameworks must identify a model' s attention type, positional encoding, expert-layer structure, vision/speech encoders, Adapter injection points, quantization formats, context length, and chat templates, and map these structural features to executable operators, cache layouts, parallel partitioning, and scheduling strategies.

Hugging Face Transformers enables new models to be integrated through configurations, custom model classes, AutoClass, and AttentionInterface, allowing them to enter the ecosystem via configuration and model code [177]. vLLM further provides a Transformers modeling backend, so that some models not natively supported by vLLM can also be incorporated into the inference-serving system through Transformers. For MoE models, vLLM requires sparse MoE blocks to expose their expert structures, and requires attention layers to invoke the backend through a unified attention-function interface [178]. For MLA introduced in the DeepSeek series, adaptation involves compressed representations

of key-value caches, Latent Projection, different computation modes in the prefill/decode stages, and backend Kernel choices such as FlashMLA, FlashInfer, Triton, and CUTLASS [57][179].

MoE models are pushing inference systems from tensor-parallel scaling toward expert parallelism and load balancing. For models such as DeepSeek-V3/R1, Mixtral, and Qwen-MoE, deployment bottlenecks lie not only in parameter count, but also in expert-weight distribution, token-to-expert routing, All-to-All communication, and Grouped GEMM. vLLM has already incorporated expert parallelism into its distributed inference capability [178]; SGLang explicitly applies expert parallelism to distribute MoE expert weights across multiple devices, and combines All-to-All communication with Grouped GEMM to reduce latency and improve throughput [180].

At the same time, inference frameworks also need to handle vision encoders, audio encoders, projection layers, Connectors, special tokens, image/video preprocessing, prompt templates, and multimodal key-value caches. TensorRT-LLM' s support matrix already covers language models, multimodal models, and vision generation models; vLLM is also expanding in directions such as LoRA, multimodal LoRA, prompt embeddings, and multimodal data processing [181].

Overall, the industry is forming a relatively clear adaptation path for model architectures: model-structure identification → configuration and weight conversion → Tokenizer/Processor/Chat Template adaptation → selection of attention and key-value cache strategies → MoE/multimodal/LoRA/quantization compatibility verification → mapping to Kernel and parallel strategies → service scheduling and performance stress testing → canary release and stability monitoring. The core goal is to translate differences in model architecture into maintainable, reusable, and evaluable engineering rules, thereby shortening the integration cycle for new models and maintaining stable throughput and latency across different GPUs, NPUs, or hybrid compute environments.

- **Differences in Model Structure**
  - **Attention**: MHA / MQA / GQA / MLA
  - **Positional Encoding**: RoPE / ALiBi
  - **Model Architecture**: Dense / MoE / MTP
  - **Multimodal Components**: Vision Encoder / Audio Encoder / Connector
  - **Adaptation Components**: LoRA / Adapter / Quantization / Custom Layer
  - **Example Model Families**: Llama / Qwen / DeepSeek / Gemma / multimodal models

→

- **Structural Parsing and Unified Abstraction**
  - Configuration parsing (Config)
  - Weight-format conversion
  - Tokenizer / Processor / Chat Template

  - Model-class registration
  - Attention interface abstraction
  - MoE / multimodal / LoRA compatibility checks

→

- **Adaptation Actions**
  1. **Inference-framework adaptation**
     - vLLM / TensorRT-LLM / SGLang / MindIE
     - ModelRunner / ModelEngine
     - OpenAI-compatible service interfaces
  2. **Kernel adaptation**
     - Attention Kernel selection
     - FlashAttention / FlashMLA / Triton / CUDA
     - MoE Kernel / Grouped GEMM
  3. **KV Cache adaptation**
     - Paged KV Cache
     - Prefix Cache / Radix Cache
     - FP8 KV Cache
     - Different paths for Prefill / Decode
  4. **Parallelization-strategy adaptation**
     - TP / PP / DP / EP / CP
     - Expert Parallel
     - Multi-machine, multi-card mapping
  5. **Scheduling-strategy adaptation**
     - Continuous Batching
     - Chunked Prefill
     - PD separation
     - Speculative Decoding
     - Load balancing / session affinity

→

- **Runtime Optimization Results**
  - Rapid integration of new models
  - Matching of operators and backends
  - Efficient reuse of KV Cache
  - Stable deployment of MoE / multimodal models
  - Optimization for long context and high concurrency

→

- **Service Output**
  - Low latency
  - High throughput
  - Stability
  - Deployability
  - Maintainability

Figure 3-5 Schematic diagram of the technical process for model-architecture adaptation

IV. Compute-Network-Model Fusion

Compute-network-model fusion is a key technical direction for advancing large models from single-machine inference optimization toward clustered, platform-based, and service-oriented deployment. Its core lies in establishing a collaboratively optimized relationship among model inference requirements, the organization of computing resources, network communication capabilities, and gateway scheduling mechanisms. Unlike model-compute fusion, which emphasizes the execution efficiency of individual operators and individual instances, compute-network-model fusion focuses more on how large models can operate stably, efficiently, and at low cost in multi-machine multi-card, multi-instance, multi-tenant, and high-concurrency scenarios. This chapter focuses on eight aspects: multi-machine inference parallelism, clustered scheduling of key-value caches, gateway sticky-session routing, gateway semantic-cache reuse, dynamic batching, high-performance communication libraries, load balancing, and rate limiting with degradation. Among these, multi-machine inference parallelism is the foundation of compute-resource scaling: through tensor parallelism, pipeline parallelism, data parallelism, expert parallelism, and context parallelism, it breaks through the memory and compute limitations of a single card. High-performance communication libraries provide the data-plane support for multi-machine parallelism and cache migration, and are responsible for efficiently completing collective communication, point-to-point communication, and key-value cache transmission among graphics processors, nodes, and resource pools. Clustered scheduling of key-value caches, gateway sticky-session routing, and semantic-cache reuse jointly revolve around "maximizing cache value," reducing the overhead of repeated computation at three levels: system-level cache management, request-routing binding, and semantic-level result reuse. Dynamic batching further improves hardware utilization at the request-scheduling level, striking a balance among throughput, first-token latency, and tail latency. Load balancing, rate limiting, and degradation together constitute service-governance capabilities, ensuring that model services can maintain stable operation despite traffic fluctuations, resource contention, and local failures. Overall, compute-network-model fusion forms a complete technical chain from "model parallel partitioning—network communication transmission—key-value cache reuse—request scheduling orchestration—gateway governance and protection," and is an important foundation for supporting the evolution of large-model inference services from single-point deployment to large-scale production operation.

(1) Multi-Machine Inference Parallelism

1. Technical Definition Multi-machine inference parallelism refers to distributing large-model inference computation across multiple graphics processors and multiple compute nod[[unclear: text cut off at bottom of page]]

on nodes, using tensor parallelism, pipeline parallelism, data parallelism, expert parallelism, context parallelism, and other approaches to address insufficient memory on a single graphics processing unit, inadequate single-machine compute, or insufficient throughput. TensorRT-LLM classifies parallel strategies as TP, PP, DP, EP, CP, and Wide-EP: TP partitions model weights; PP distributes the model by layer; DP replicates the model to handle different requests; EP is used for MoE expert distribution; and CP is used for long-context processing.

2. Industry Progress

DeepSeek' s EPLB (Expert Parallelism Load Balancer) is currently one of the most noteworthy advances in multi-machine MoE inference parallelism. Its objective is to generate an expert replication and placement plan during EP deployment based on expert-load statistics: highly loaded experts are redundantly replicated, logical experts are mapped to multiple physical expert replicas, and the replicas are then packed onto graphics processing units so as to balance, across GPUs, both the number of received tokens and the expert computation load. The core inputs to EPLB are the load statistics of the logical experts at each layer, and its outputs are mappings such as physical-to-logical and logical-to-physical, as well as the number of replicas for each logical expert [182].

The key design of EPLB includes two types of strategies. The first is hierarchical load balancing: when the number of expert groups is divisible by the number of nodes, the group-limited routing mechanism of DeepSeek-V3 is used to first pack expert groups evenly onto different nodes, and then experts are replicated within a node and placed onto a single graphics processing unit, thereby reducing cross-node communication traffic as much as possible. This strategy is more suitable for scenarios in the prefill stage where the scale of expert parallelism is relatively small. The second is global load balancing: experts are no longer restricted by expert groups, but are replicated and placed across the global scope; this is more suitable for scenarios in the decoding stage where the scale of expert parallelism is relatively large [182].

From the perspective of the open-source ecosystem, EPLB has already been incorporated into multiple inference systems. The expert-parallel deployment documentation of vLLM already provides the `--enable-eplb` configuration option, and explains that vLLM collects expert-load statistics during each forward inference pass and periodically rebalances the expert distribution. Its configuration options include `window_size`, `step_interval`, `num_redundant_experts`, `use_async`, and the communication backend [183]. SGLang has also integrated DeepSeek EPLB, and recommends stabilizing expert-activation statistics by increasing the `batch size`, and adapting to changes in business traffic through periodic rebalancing [184].

On the communication side, DeepSeek' s open-sourced DeepEP further complements the low-level capabilities required for large-scale expert parallelism.

DeepEP provides high-throughput, low-latency All-to-All GPU kernels for MoE dispatch/combine operations, supports low-precision communication such as FP8, and in V2 unifies the high-throughput and low-latency interfaces into ElasticBuffer, supporting larger expert-parallel domains for scale-up/scale-out expansion. Its significance is that the bottleneck of MoE inference lies not only in imbalanced expert placement, but also in the irregular All-to-All communication caused by dynamic routing of tokens across graphics processing units. EPLB is responsible for "how experts are placed," while DeepEP is responsible for "how tokens are delivered efficiently to experts" [185].

When reproducing a DeepSeek-style large-scale inference system in 2025, SGLang used 12 nodes, each with 8 H100s, combining PD disaggregation, large-scale EP, DeepEP, DeepGEMM, and EPLB. Under 2,000-token input sequences, it reported throughput of 52.3k input tokens/s and 22.3k output tokens/s per node. SGLang also noted that the effectiveness of EPLB depends on the match between the input distribution and the real serving load, and that larger batch sizes and periodic rebalancing help alleviate random fluctuations in expert load [184].

(a) DP Dense FFN with DP Attention *(b) EP Sparse FFN with DP Attention*

Figure 4-1 Schematic comparison of hybrid parallel deployment architectures for DP-based dense FFNs and EP-based sparse FFNs; adapted from Ref. [184]

After EPLB, DeepSeek also open-sourced LPLB (Linear-Programming-Based Load Balancer) as an extension at the research stage. LPLB uses linear programming to dynamically optimize token allocation at the per-batch/per-layer level; whereas EPLB is more oriented toward addressing long-term or static imbalance based on historical load statistics, LPLB attempts to handle instantaneous dynamic imbalance caused by small-batch randomness. However, DeepSeek also explicitly notes that LPLB is still at an early research stage, and lists limitations including solver overhead, balancing only the number of tokens without modeling the nonlinear time cost of Grouped GEMM, and the possibility of underperforming EPLB under extreme global imbalance [186]. Therefore, the current industry trend in multi-machine inference parallelism can be summarized as follows: dense models are still based on combinations of TP/PP/DP/CP; MoE models rely more on DP Attention + EP MoE + PD separation + dedicated All-to-All communication + EPLB load balancing.

(II) Clustered Scheduling of Key-Value Caches

1. Technical Definition

Clustered scheduling of key-value caches refers to, in clusters with multiple graphics processors, multiple nodes, multiple inference instances, or PD separation, elevating the key-value cache from an intermediate GPU-memory state of a single request into a manageable, indexable, migratable, reusable system-level resource, and performing block allocation, index matching, hit evaluation,

cache-aware routing, cross-instance transmission, hierarchical storage, priority-based eviction, tenant isolation, and security control on it, so as to reduce TTFT, TPOT, and inference cost, and to improve GPU-memory utilization and system throughput.

2. Industry Progress

At the level of single-machine GPU-memory management, vLLM' s PagedAttention is a representative work. This approach partitions the key-value cache of each request into fixed-size blocks/pages, and avoids allocating large contiguous chunks of GPU memory by mapping logical blocks to physical blocks, thereby reducing GPU-memory fragmentation and redundant copying, while supporting block-level sharing within the same request or across requests. The vLLM paper reports that, at the same latency level, compared with systems such as FasterTransformer and Orca, it can achieve a 2-4× throughput improvement, with more pronounced gains in long-sequence, large-model, and complex decoding scenarios [48]. At the same time, vAttention uses the CUDA virtual memory management API to decouple virtual addresses from physical GPU-memory allocation, keeping the key-value cache contiguous in virtual address space while allocating physical GPU memory on demand, in order to reduce PagedAttention' s requirements on attention kernels and framework memory management

the complexity of the modifications [187].

At the level of prefix caching and cache-aware scheduling, vLLM' s automatic prefix cache identifies identical prefixes through hash-based blocks, and incorporates the parent-block hash, tokens within the block, LoRA ID, multimodal input hash, cache salt value, and other information into the cache key. This indicates that production systems have begun to address the integrated issues of cache hits, cache isolation, and multi-tenant security. SGLang' s RadixAttention further maintains the key-value caches of all requests in a radix tree and, combined with LRU eviction and cache-aware scheduling, realizes automatic key-value cache reuse in complex large language model programs such as multi-turn dialogue, few-shot prompting, agents, and multi-branch inference [108]. TensorRT-LLM also uses a radix tree to store reusable blocks and supports mechanisms such as priority-based LRU, retention policies, and cache salting.

At the level of hierarchical storage and cross-engine sharing, LMCache represents the direction of an "independent key-value cache layer." Its core idea is to detach the key-value cache from the HBM of the graphics processor of a single inference engine, extend it to the graphics processor, CPU, storage, and network layers, and share it among engines such as vLLM and SGLang. The paper emphasizes cache orchestration through bulk data movement, computation and I/O pipelining, modular connectors, and control APIs [188].

At the level of PD disaggregation scheduling, Mooncake, for Kimi' s production services, proposes an architecture centered on the key-value cache: it separates

Mooncake: a decoupled architecture centered on the key-value cache

Figure 5: Mooncake: a decoupled architecture centered on the key-value cache

the prefill and decoding clusters, and organizes CPU, runtime memory, solid-state disks, network cards/RDMA, and remote direct memory access resources into a distributed key-value cache. A global cache and scheduler trade off throughput against latency SLO [163].

Figure 4-2 Mooncake: a decoupled architecture centered on the key-value cache; cited from Reference [163]

At the level of platformization and production deployment, TensorRT-LLM and NVIDIA Dynamo incorporate key-value cache transfer, routing, offloading, and disaggregated serving into inference-platform capabilities. TensorRT-LLM' s disaggregated serving supports multiple access modes such as trtllm-serve, Dynamo, and Triton; its key-value cache exchange module is responsible for sending and receiving the key-value cache, releasing cache space, and converting cache layouts. It supports communication backends including MPI, UCX, and NIXL, and reduces communication overhead by overlapping transfer across multiple requests with computation [189]. NVIDIA Dynamo combines disaggregated serving, key-value-cache-aware routing, and key-value cache offloading into data-center-level inference capabilities, and uses NIXL as a low-latency data-transfer layer to support key-value cache movement between nodes [190].

(III) Gateway Sticky-Session Routing

1. Technical Definition

Gateway sticky-session routing refers to a mechanism at the AI gateway layer that, through a specific hash algorithm or state-mapping logic, continuously directs requests with the same contextual features to the same backend inference replica.

In traditional Web development, sticky sessions are mainly used to maintain server-side memory state, such as helping users keep the items in their shopping carts. In large language model inference scenarios, however, their core mission has become the ultimate reuse of key-value caches. Because inference with large models is an autoregressive generation process, each round of dialogue deposits a large amount of intermediate computation results in GPU memory. If, when facing a context as long as 32k, every new conversation had to recompute the prefix, users would not only wait several more seconds, but expensive compute resources would also be wasted.

To solve this pain point, the AI gateway needs a tightly interlocking set of streaming processing logic. First is feature extraction: when the gateway receives a request, it rapidly parses the session ID, user ID, or directly hashes the prompt prefix in the request header. Through a consistent hashing algorithm

or a dynamic lookup table, it accurately and continuously routes requests carrying the same features to the same backend inference replica. In this process, advanced gateways also maintain real-time awareness of the state of backend nodes, avoiding GPU-memory overflow on a single card caused by an excessive pursuit of stickiness.

This routing architecture, which pursues maximal reuse, is usually supported at the lower level by three core components working together. The prefix recognizer plays the role of a "microscope," segment-hashing the system prompt or historical conversation and accurately identifying reusable text fragments. The routing controller is the "chief dispatcher," responsible for executing the stickiness strategy and finding a balance between "maximizing cache hits" and "keeping backend load balanced." Finally, the global cache manager serves as the metadata center, recording which context fragments are currently cached in the GPU memory or memory of which nodes, thereby providing the entire scheduling system with a global "field of view."

2. Industry Progress

Industry scheduling strategies are undergoing a paradigm shift from "simple hashing" to "cache-aware scheduling." Early routing methods were very crude, relying only on user ID or IP address for fixed binding. Today' s advances have gone deeper into fine-grained, semantically aware routing. The most typical breakthrough is the upward migration of prefix-tree management logic: inference frameworks such as vLLM and LightLLM originally used prefix trees only inside nodes to manage caches, while the latest gateway solutions—such as AI extensions based on Envoy—attempt to extract this tree-shaped logic directly and elevate it to the gateway layer, thereby achieving global prefix matching at the traffic ingress [48]. At the same time, layered caching strategies are also evolving in coordination. Leading inference platforms have begun implementing a three-level caching mechanism from GPU memory, to memory, to disk. At this point, gateway sticky routing works together with caching technologies supported by solid-state disks: even if a request drifts to another node because of load factors, the backend can quickly restore the context state from the local disk, preventing instantaneous performance collapse.

Taking the currently mainstream inference engine vLLM as an example, its native server-side support for automatic prefix caching is forcing the gateway layer to perform intelligent matching. At the current stage, many developers in production environments use specific Nginx hash configurations or write custom Lua scripts to forcibly route requests with the same prompt prefix to a fixed inference replica. In terms of practical deployment results, this kind of software-hardware collaborative optimization can reduce first-token latency by more than 60%.

Under the more cutting-edge PD disaggregation architecture, gateway sticky routing faces even greater complexity challenges. In this mode, the gateway

must not only pay attention to session stickiness in the decoding phase, but also play a higher-level orchestration role. After a prefill node completes the massive context computation and generates the key-value cache, the gateway needs to work with the underlying communication mechanism to efficiently "push" or "pull" these cached data to the designated decod-

decoder node, and in subsequent ongoing conversations, firmly lock the route to this receiving node.

For this gateway-layer responsibility above the inference engine, major mainstream open-source projects are also iterating at high frequency. For example, the Envoy Gateway community is intensively developing AI gateway extensions and has added a dedicated KV-cache routing plugin. Domestic open-source gateways such as Higress and Kong have also begun deep integration of large-model plugins, supporting not only dynamic rate limiting based on token counts, but also load balancing based on semantic vectors. These advances mean that gateway sticky routing has completely moved beyond simple "identity binding" and has formally entered a new era of "content binding."On its MaaS platform, China Unicom uses a self-developed dynamic load balancer that implements sticky session routing through session identifiers, routing requests with the same prefix to the same node as much as possible and significantly improving the hit rate of the key-value cache.

Visible labels in the diagram:

- User conversation requests (User Conversations)
- User A —Session: userA-chat1
- User B —Session: userB-projectX
- User C —Session: userC-promptY
- LLM Requests (Conversations, Tasks)
- Entry Layer
- MaaS API Gateway and Sticky Router (Gateway & Router)
- MAAS API GATEWAY
- STICKY SESSION ROUTER
- Affinity Management (Affinity Mgmt)
- Context Affinity (Affinity Table)
  - Session ID → Specific Worker ID
  - Session ID → ···
- Session ID mapping table (Affinity Table)
- Routing Decision
  - If Session ID in Affinity Table, route to Mapped Worker
  - else, Load Balance & Create Mappings
- Routing decision (Routing)
- Backend inference worker node cluster (Workers)
- Affinity Bond
- Cluster Manager (Worker status and load)
- Worker Node 1
  - Session A KV Cache

  - LLM Instance
  - KV CACHE (Context Memory)
- Worker Node 2
  - LLM Instance
  - Compute GPU
  - KV CACHE (Context Memory)
- Worker Node 3
  - LLM Instance
  - Compute GPU
  - Session B
  - KV CACHE (Context Memory)
- Session A (LLM Instance)
- KV cache–session-specific (KV Cache)
- Session B (KV Cache)
- Performance optimization: reuse context (Reuse Context for Speed)
- Backend Inference Clusters

Figure 4-3 MaaS sticky-session routing technology architecture diagram

(IV) Gateway Semantic Cache Reuse

1. Technical Definition

In the engineering implementation of large language models, because model inference involves extremely high computational cost and significant response latency, traditional cache schemes based on simple character matching can no longer fully meet the needs of the AI era. As the first filter for traffic, semantic caching has emerged accordingly. It is a core component that uses representation learning and vector similarity retrieval technologies to intelligently reuse unstructured natural-language requests.

Traditional caching systems (such as Redis) follow strict key-value matching logic. For example, when a user successively enters "How is the weather in Beijing?" and "Please tell me today' s climate in Beijing," a traditional cache regards these as two completely independent strings and therefore cannot reuse the cache. Semantic caching, by contrast, uses vectorization technology to map text into a high-dimensional continuous vector space. In this space, texts that differ on the surface but are semantically similar have very small vector distances (such as Euclidean distance or cosine similarity), enabling the system to understand the "meaning" behind the text.

In actual operation, semantic caching at the gateway layer follows a standardized and efficient processing workflow. When a new request arrives at the gateway, the vector encoding layer first intervenes, using a lightweight text-embedding model to convert the original user request into a dense vector in real time. This vector is then sent into a high-performance vector [[unclear: word continues beyond the visible page]]

databases (such as Milvus or Vespa) to perform fuzzy retrieval and quickly find the historical requests that are closest in vector space.

The retrieved candidate results are not returned directly to the user. Instead, they must undergo strict validation by the evaluation layer, which determines whether the retrieved historical cached content is logically truly equivalent to the current request, and makes the final semantic judgment based on a similarity score. If it is judged to be a cache hit, the gateway directly intercepts the traffic and returns the cached answer, reducing the response latency that would otherwise take several seconds to the millisecond level. If it is judged to be a cache miss, the request is forwarded normally to the back-end LLM cluster. At the same time, the system asynchronously writes the response generated by the large model into the vector database, preparing it for a possible semantic hit next time.

Figure content: MaaS API Gateway; User; User Request; Embedding Service —Converts text to vectors; Similarity Search (Threshold Check); Matched; Return cached Response; Response; Not Matched; LLM Inference Service; Save to Vector Database; Vector Database; Cached Prompts & Responses; Response to User.

Figure 4-4 MaaS API Gateway

The strengths and weaknesses of semantic caching are usually measured by the following three key indicators:

Hit rate: In real business scenarios, because of the diversity of ways in which questions are asked, semantic caching can usually increase the effective hit rate by 3-5 times compared with traditional caching.

Retrieval latency: It must be ensured that the time consumed by embedding plus vector retrieval is far less than the time consumed by LLM generation; the ratio usually needs to be less than 1:10.

Semantic drift: This refers to cases in which requests with high similarity but different logical meanings are mistakenly judged as hits. Solving this problem depends on a sophisticated evaluation-layer design.

2. Industry Progress

Semantic caching technology has rapidly evolved from its budding stage in 2023 to a highly intelligent and edge-oriented stage in 2026. Its development reflects a transition from "brute-force matching" to "precise alignment." As the industry' s first mature open-source semantic caching framework, the emergence of GPT-Cache marked the standardization of this technology. It proposed a five-layer modular architecture:

Storage layer: supports elastic scaling from local disks to distributed vector databases.

Embedding layer: supports HuggingFace, OpenAI, and various privatized models.

Evaluation layer: introduces similarity-threshold settings, initially solving the reuse-logic problem.

Routing layer: determines the data flow and supports mixed use of multiple models.

Management layer: responsible for the cache life cycle and update strategies.

Based on its own technical accumulation in the RAG direction, China Unicom implemented a semantic caching architecture similar to GPTCache by combining it with a self-developed small verifier model. Research data show that in specific vertical domains, such as financial consulting and government-service Q&A, properly configuring GPTCache can help enterprises reduce duplication by more than 80%.

reduce inference costs and significantly improve user experience [191].

As application scenarios become more complex, decision strategies that rely solely on vector distance have exposed the defect of "plausible but incorrect" hits [192]. For example, "How do I activate membership?" and "How do I cancel membership?" may be extremely close in vector space, yet their answers are diametrically opposed. Therefore, the industry has introduced a verifier pattern: Inala and others proposed introducing a micro Transformer model at the gateway layer as a "verifying officer." Before returning a cached result, the small model rapidly cross-validates the Query and the Cache Key. This combination of "large-model generation and small-model verification" greatly improves result accuracy while ensuring response latency.

Entering 2026, the Cortex architecture proposed by Wang and others solved the problem of data access in ultra-large-scale distributed environments. By establishing semantic-aware mappings, Cortex enables intelligent prefetching of data at the edge. The system can predict the semantic path the user is likely to take next and proactively push caches that may be hit to the nodes closest to the user [193].

Current industry trends are moving toward multimodal semantic caching. Semantic caching is beginning to enter production environments not only for text, but also for image generation and audio-video generation. By caching intermediate features in latent space, the cost of multimodal inference is being further optimized. For multimodal semantic caching technology, China Unicom is also actively exploring feasible solutions.

(V) Dynamic Batching

1. Technical Definition

Dynamic batching refers to the inference server dynamically merging multiple requests into batches for execution according to request arrival time, model instance status, input shape, queue waiting time, and the upper limit of batch size, in order to improve hardware utilization and throughput. The NVIDIA Triton documentation defines dynamic batch processing as the server-side capability of combining inference requests and dynamically creating batches. It can usually improve throughput, and can be configured through maximum batch size, preferred batch size, maximum queue delay, and queue attributes. For requests with different input shapes, Triton provides a ragged batching scheme, reducing the redundant overhead caused by explicit tensor zero-padding.

2. Industry Progress

In traditional model services such as vision, speech, retrieval, classification, and embedding, dynamic batching is mainly manifested as request-level batch processing: according to request arrival time, input shape, model instance idleness, maximum queue delay, queue priority, and timeout strategies, the inference server merges multiple requests for the same model and executes them together, thereby improving graphics processor utilization and per-unit-time throughput. NVIDIA Triton is a typical representative of this capability; its dynamic batch processor is oriented toward stateless models and supports scheduling based on batch size, queue delay, and priority. At the same time, it supports some variable-length input scenarios through ragged batching, provided that the model backend can process the concatenated irregular variable-length inputs and the corresponding in-batch element information [194].

In generative large-model inference, the focus of dynamic batching shifts to token-level continuous scheduling. Requests to large language models usually include two stages: prefill and decoding. The prefill stage processes the complete prompt; it is compute-intensive and affects first-token latency. The decoding stage generates tokens one by one; the single-step computation is relatively small, but the duration and output length are uncertain. Static batches are slowed down by long-output requests, causing completed requests within the batch to be unable to exit in time and new requests to be unable to enter promptly. ORCA, proposed at OSDI 2022, introduced iteration-level scheduling and selective batching, reducing the scheduling granularity to the model-iteration level so that the scheduler executes only one generation iteration at a time, and allowing requests to dynamically enter or exit a batch during generation. In evaluations on GPT-3 175B, the paper reported a 36.9× throughput improvement over FasterTransformer at the same latency level [195].

An important direction in recent years has been optimization by separating the prefill and decode stages. Sarathi-Serve argues that, although the prefill stage has high latency, it can easily saturate the graphics processor, whereas

the decode stage has low latency but insufficient graphics-processor utilization. It therefore proposes Chunked-Prefill and Stall-free Scheduling: the prefill of long prompts is split into approximately equal-length chunks, and new requests are inserted without interrupting ongoing decoding, thereby achieving a better balance between large-batch throughput and low latency. Its OSDI 2024 paper reports that, on models such as Mistral-7B, Yi-34B, and Falcon-180B, service capacity can be improved relative to vLLM [170].

Text in Figure 4-5: Timeline; C, D enter; vLLM; Decodes for A, B stalled; TBT without prefill interference; TBT with prefill interference; Prefill Prioritized Schedules; Orca; A exits; FasterTransformer; Prefills for C, D stalled; Decode Prioritized Schedule; B exits; Sarathi-Serve; No stalls; Stall-free Schedule.

Figure 4-5. Sarathi-Serve advances dynamic batching to prefill/decode-stage-aware hybrid batching; cited from Reference [170].

In multi-instance, multi-tenant, and mixed-workload scenarios, industry has begun to combine dynamic batching with global queue management, request migration, priority isolation, and service-level-objective-aware scheduling. Llumnix focuses on runtime rescheduling across multiple model instances, improving load balancing, resource fragmentation, priority isolation, and tail latency through online migration of requests and memory state. Its OSDI 2024 paper reports that tail latency can be improved by an order of magnitude, with up to a 1.5× speedup for high-priority requests [196]. QLM, by contrast, targets mixed deployment scenarios involving multiple models, multiple service-level objectives, online interactive requests, and offline batch requests. It uses a waiting-time estimator and a global scheduler to orchestrate operations such as request pulling, eviction, load balancing, and model swapping, improving the service-level-objective attainment rate by 40%–90% and throughput by 20%–400% [197].

Research after 2025 further extends from single-instance continuous batching to intra-device pipelining, cluster-level pooling, and token-level resource scheduling. NanoFlow argues that, under many common workloads, end-to-end large language model serving suffers utilization losses caused by the serialization of heterogeneous operations such as computation, memory, and networking. It therefore proposes nano-batching and intra-device resource-overlap scheduling, achieving a 1.91× throughput improvement over existing systems under real workloads [198]. Aegaeon, oriented toward concurrent multi-model serving, proposes graphics-processor pooling and token-level automatic scaling. It adopts different scheduling strategies in the prefill and decode stages, improving resource-pooling efficiency and the ability to meet service-level objectives for multi-model serving at token granularity [199].

(VI) High-Performance Communication Libraries

1. Technical Definition

A high-performance communication library refers to a communication software stack for multi-GPU, multi-node parallel computing and inference services. It provides collective communication and point-to-point communication capabilities, such as AllReduce, AllGather, ReduceScatter, All-to-All, Broadcast, and Send/Receive, and is optimized for interconnects including NVLink, NVSwitch, PCIe, InfiniBand, RoCE, and RDMA. NCCL' s official definition describes it as a library of fundamental multi-GPU and multi-node communication primitives optimized for GPUs and networks; it provides topology-aware inter-GPU communication primitives and focuses on accelerating communication between GPUs.

2. Industry Progress

High-performance communication libraries now cover the underlying data-plane software stack for training parallelism, MoE expert parallelism, inference-service disaggregation, key–value cache migration, and cross-resource-pool scheduling. On the training side, data parallelism, tensor parallelism, pipeline parallelism, and expert parallelism rely respectively on communication primitives such as AllReduce, AllGather, ReduceScatter, All-to-All, Broadcast, and Send/Receive. NCCL natively supports these communication primitives and is optimized for channels such as PCIe, NVLink, InfiniBand Verbs, and IP sockets [200]. UCX, by contrast, more often serves as the underlying communication framework, providing transport abstractions such as RDMA, TCP, GPUs, shared memory, and network atomic operations, and supporting MPI, artificial-intelligence frameworks, and inference transport components in building unified communication paths [201].

In recent years, the optimization focus of communication libraries has shifted toward topology awareness, algorithm adaptivity, and large-scale initialization efficiency. In NCCL 2.23, NVIDIA introduced the PAT algorithm to improve the scaling efficiency of AllGather and ReduceScatter in large-scale scenarios, while also enhancing initialization APIs, user-buffer registration, and profiler plug-in capabilities [202]. Subsequently, NCCL further advanced cross–data-center communication capabilities: through mechanisms such as `fabricId` and `ncclNet`, network topology is incorporated into the selection process for collective communication algorithms such as rings and trees, enabling cross–data-center training or cross-resource-pool training to minimize the use of expensive cross-domain links as much as possible [203].

For ultra-large-scale training, communication libraries have also begun to strengthen reliability, observability, and fault-diagnosis capabilities. Starting with NCCL 2.24, NCCL introduced the RAS subsystem to query the health status of NCCL jobs, assist in locating crash issues, identify abnor-

mal communication processes, and provide low-overhead state-observation capabilities. NCCL 2.26 further enhanced GPU-kernel and network profilers, quality-of-service control for network plug-ins, and RAS capabilities. At the same time, tools such as NCCL Inspector have begun to provide resident recording of collective-communication performance and metadata, covering metrics such as algorithm bandwidth, bus bandwidth, execution time, message size, and collective-communication type, thereby supporting the localization of communication bottlenecks in large-scale clusters [204].

In terms of communication–computation fusion, NCCL has provided the Device API since version 2.28, allowing CUDA kernels to directly invoke device-side communication primitives. It also provides modules such as LSA, Multimem, and GIN for building reduction, broadcast, and communication–computation fused kernels. Such capabilities are of great significance for low-latency inference, MoE dispatch and aggregation, fine-grained expert parallelism, and custom collective communication, because communication can be brought closer to the GPU-kernel scheduling path, reducing CPU intervention and synchronization overhead.

MoE and expert parallelism are driving communication libraries toward specialized communication libraries oriented to model structures. DeepEP represents this direction: it is positioned as an expert-parallel communication library for modern training and inference scenarios, with a focus on optimizing MoE dispatch and aggregation, low-latency All-to-All, and low-precision data-transfer paths such as FP8, so as to reduce the interference of the communication process with compute kernels [185].

On the inference-service side, the importance of high-performance communication libraries is further increasing. NIXL targets this class of inference data-transfer scenarios and is positioned as an open, vendor-neutral data-movement library supporting central processing units/graphics proce-

processor memory as well as different resources such as files, blocks, and object storage, and adapts transport mechanisms such as UCX through pluggable backends; its accompanying NIXLBench and KVBench also show that inference communication is simultaneously focusing on key–value transfer latency, throughput, and end-to-end service metrics [205].

Figure 4-6 Schematic diagram of the NIXL transfer-agent architecture and backend plugin system; cited from Reference [205]

Cross-vendor interoperability has also become an important direction. AMD RCCL provides collective communication and point-to-point communication interfaces similar to NCCL; oneCCL provides scale-up and scale-out collective communication capabilities for the oneAPI ecosystem; and UCC attempts to provide a unified collective communication layer for MPI, PGAS, and deep-learning frameworks, while supporting backends such as CUDA.

Schematic of the NIXL transfer-agent architecture and backend plugin system. Visible labels include: “Within a NIXL agent: mem_type + list of (addr, len, devID)” ; “NIXL cross-node & cross-mem buffer list primitive” ; “NIXL transfer handles w/ repost capability” ; “Async create/post/check for zero-copy transfers. (RD/WR op + optional notif).” ; “North-Bound NIXL API” ; “Transfer Agent” ; “Memory Section” ; “Metadata Handler” ; “Local registered memory info” ; “Remote agent info” ; “South-Bound Backend API” ; backend plugins “UCX” , “MC” , “Lib-fabric” , “UCCL P2P” , “GPU-NetIO” , “POSIX” , “GDS” , “HF3FS” , “OBJ (S3)/RDMA” , “GUSLI” , “Azure Blob” , “Custom” ; “Transfer Backend Plugins” ; and “NIXL chooses the best available backend for the user, and plugin does the transfer efficiently.”

Figure 6: Schematic of the NIXL transfer-agent architecture and backend plugin system. Visible labels include: “Within a NIXL agent: mem_type + list of (addr, len, devID)” ; “NIXL cross-node & cross-mem buffer list primitive” ; “NIXL transfer handles w/ repost capability” ; “Async create/post/check for zero-copy transfers. (RD/WR op + optional notif).” ; “North-Bound NIXL API” ; “Transfer Agent” ; “Memory Section” ; “Metadata Handler” ; “Local registered memory info” ; “Remote agent info” ; “South-Bound Backend API” ; backend plugins “UCX” , “MC” , “Lib-fabric” , “UCCL P2P” , “GPU-NetIO” , “POSIX” , “GDS” , “HF3FS” , “OBJ (S3)/RDMA” , “GUSLI” , “Azure Blob” , “Custom” ; “Transfer Backend Plugins” ; and “NIXL chooses the best available backend for the user, and plugin does the transfer efficiently.”

(VII) Load Balancing

1. Technical Definition

A load balancer is located between the client and the backend server array, playing the core roles of reverse proxy and traffic distributor. In modern distributed architectures, its value is mainly reflected in three aspects: first, horizontal scalability, allowing the overall processing capacity of the system to be improved linearly simply by increasing the number of servers; second, high availability, whereby continuous health checks monitor backend nodes in real time, and once a faulty server is detected, it is automatically shielded so that traffic can be migrated smoothly; and third, performance optimization, enabling tasks to be reasonably assigned according to the actual load of backend nodes so that resource utilization across the entire cluster reaches the global optimum [206].

From the perspective of technical implementation, according to the OSI seven-layer model, load balancing is mainly carried out along two dimensions. The first is Layer 4 load balancing, which works at the TCP/UDP protocol layer and forwards traffic mainly on the basis of low-level network information such as the source IP, destination IP, and port number. Because the entire process does not require packet disassembly or parsing of complex application-layer protocols, its forwarding efficiency is extremely high; LVS and the hardware device F5 are typical representatives in this field.

The second is Layer 7 load balancing, which goes deep into application-layer protocols such as HTTP, HTTPS, and gRPC. Because it can identify specific content such as URLs, Cookies, and HTTP headers, Layer 7 load balancing has powerful "content-awareness" capabilities. This enables it to support more fine-grained scheduling strategies, such as routing a request to a specific microservice according to its path. Common tools include Nginx, HAProxy, and Envoy.

In scheduling for specific scenarios, the algorithmic model determines where traffic is directed. The most basic is the round-robin algorithm, which cyclically assigns tasks in order and is very suitable for scenarios in which the backend servers have completely equivalent configurations; if the backend machines

If server performance varies, weighted round robin can be adopted, dynamically assigning weights according to the level of each server' s hardware configuration.

For scenarios involving long-lived connections or large differences in processing time, the least-connections algorithm is a better choice: it prioritizes forwarding new requests to the node with the fewest currently active connections. To ensure that requests from the same source can stably land on the same server, the system introduces a consistent hashing algorithm, maintaining routing stability to the greatest extent when nodes change.

2. Industry Progress

As cloud computing enters a more mature stage and demand for large language model (LLM) inference explodes across the board, load-balancing technology is undergoing a paradigm shift from "passive routing" toward "active self-adaptation."

At the architectural level, the industry is accelerating the deep integration of load balancing, traffic gateways, and security gateways. As the standard data plane for Service Mesh, Envoy—thanks to its powerful dynamic configuration capability (xDS API) and highly extensible plug-in architecture—has become the preferred choice for major modern Internet companies seeking to replace traditional Nginx clusters [207]. In recent leading practices, open-source projects such as Higress, which has evolved on the basis of Envoy [208], have successfully achieved unified coordination between Layer 4 to Layer 7 load balancing and Ingress controllers. This architectural integration eliminates the need for traffic scheduling to repeatedly jump among multiple components, greatly simplifying enterprise operations and maintenance chains. China Unicom has independently developed a cloud-native gateway that replaces traditional nginx load-balancing solutions based on Envoy+Istio, and combines model-probing technology to dynamically perceive the actual load of inference instances, thereby realizing intelligent adaptive scheduling.

Given the particular characteristics of large-model inference, traditional load-balancing strategies are beginning to fail. In AI inference scenarios, because the key-value cache generated during each dialogue has extremely high reuse value,

the latest industry development is the introduction of "semantic-aware load balancing." When the gateway receives a request, it no longer simply distributes it blindly; instead, by rapidly analyzing the semantic similarity of the prompt, it precisely routes the request to a graphics-processing-unit node that has already cached the relevant context. This scheduling approach enables backend nodes to maximize reuse of existing intermediate computation results, thereby significantly improving overall inference throughput.

At the same time, in response to the dynamic nature of large-model generation tasks, load balancing is also evolving toward "predictive rate limiting and distribution." Traditional algorithms often rely on the number of connections or requests to evaluate backend pressure. In inference scenarios, however, even for the same connection, generating 10 tokens and generating 2,000 tokens differ enormously in their consumption of graphics-processing-unit computing power. Therefore, a new generation of adaptive load balancers combines the token rate generated by the model in real time to dynamically and precisely calculate the actual carrying pressure of each node, thereby making more reasonable traffic-distribution decisions and fully bidding farewell to the former era of coarse-grained scheduling.

(VIII) Rate Limiting and Degradation

1. Technical Definition

Gateway rate limiting [209] refers to limiting the number of requests entering the system within a specific time window so as to prevent the system' s maximum processing capacity from being exceeded. As a point where traffic converges, the gateway can protect valuable backend computing resources from being crushed by instantaneous traffic spikes by executing rate-limiting strategies.

Core algorithm models:

Fixed-window algorithm: time is divided into fixed periods, and a fixed number of requests is admitted in each period. Its advantage is simplicity; its disadvantage is that a double-volume traffic impact may occur at window boundaries.

Sliding-window algorithm: solves the boundary problem of fixed windows and provides a smoother traffic curve.

Token-bucket algorithm: the system generates tokens at a constant rate, and a request can be processed only after obtaining a token. It permits a certain degree of burst traffic.

Leaky-bucket algorithm: requests enter the bucket like water and flow out for processing at a constant rate. It emphasizes an absolutely smooth output rate.

Gateway degradation is a defensive strategy of "sacrificing the pawns to save the king." When the gateway detects that a backend service has slow responses, an elevated error rate, or exhausted resources, it proactively stops invoking

that service and instead executes preset default logic, preventing failures from producing a cascading avalanche effect in a distributed system.

Core logic stages:

Circuit breaker opens: when the error rate reaches a threshold (for example, 50% of requests failing within the most recent 30 seconds), the gateway directly intercepts subsequent requests.

Half-open state: after a period of time (a sleep window), the gateway allows a small amount of traffic to attempt access. If successful, it recovers; if it fails, it returns to the open state.

Service degradation: return a user-friendly static error page, a default value (cached data), or an empty response, ensuring that the user side is not left hanging.

2. Industry Progress

Entering 2026, rate limiting and degradation at the gateway layer have gradually evolved from simple "static configuration" into a new paradigm of "intelligent adaptation" and "full-link collaboration."

The traditional rate-limiting model relies heavily on manually estimated QPS thresholds. In cloud environments where dynamic scaling has become the norm, this approach is prone to problems such as configuration invalidation and imprecise protection, and it is no longer able to meet the operational requirements of today' s complex systems. Modern gateways, such as Envoy, have begun introducing adaptive rate-limiting schemes based on Little' s Law. By constructing a dynamic feedback mechanism using CPU utilization, response time, and the number of concurrent requests, the system can automatically adjust rate-limiting thresholds according to current health indicators, thoroughly freeing itself from dependence on manual intervention. Alibaba' s open-source Sentinel 2.0[210] further strengthens this adaptive flow-control capability: by accurately detecting the system "queue length," it can effectively identify system load inflection points and carry out traffic control in advance.

At the same time, rate-limiting and degradation capabilities are gradually sinking into the infrastructure layer. The industry currently commonly adopts Envoy-based gateway solutions. With the help of WebAssembly plug-ins, developers can dynamically issue complex rate-limiting logic, such as dynamic quota adjustment for specific VIP users. The entire process does not require restarting the gateway, greatly improving operations efficiency and system flexibility[211]. China Unicom' s self-developed model gateway, based on cloud-native Envoy + Istio + self-developed WASM plug-ins, implements multiple rate-limiting algorithms such as QPS, TPM, and concurrency, and supports rate-limiting policy configuration across multiple dimensions, including tenants, users, applications, and models, thereby meeting the needs of different practical application scenarios.

In terms of degradation strategies, the industry is also continuing to evolve toward intelligence and refinement. Combined with the special characteristics of LLM gateways, when a backend inference model is overloaded, gateway degradation is no longer a matter of simply returning a prompt that "the system is busy." Instead, it can degrade to a smaller model with fewer parameters, achieving flexible service in which "performance decreases but functionality is not interrupted," minimizing the impact on user experience[212]. In response to the characteristics of large-model services, China Unicom has implemented adaptive model degradation: when a model is overloaded, the gateway proactively degrades to a smaller model, thereby enabling flexible service.

Future Outlook

Large-model inference optimization for token operations is a key foundation for supporting large-scale, low-cost, and highly stable operation of large-model services. As large-model applications expand from general question answering and content generation to long-document understanding, code generation, multimodal generation, and agent execution, large-model MaaS platforms are gradually evolving into complex systems that encompass model selection, context processing, token generation, cache reuse, computing-power scheduling, gateway governance, and service assurance. Token-generation efficiency, invocation cost, response latency, quality stability, and service controllability are becoming important indicators for measuring the operational capabilities of MaaS platforms.

From the perspective of the technical system, large-model inference optimization has already moved from single-point acceleration toward full-chain collaboration. At the multi-model integration layer, capability-boundary quantification, intelligent routing, model cascading, and model integration address the problem of optimal matching among different requests and different models. At the model-optimization layer, speculative decoding, model quantization, model distillation, chain-of-thought optimization, key-value cache compression, and generation-path optimization reduce the generation cost per token. At the model–computing integration layer, operator fusion, memory-access optimization, basic operator acceleration, engine-parameter tuning, and model-architecture adaptation improve the execution efficiency between the model computation graph and heterogeneous computing power. At the computing–network–model integration layer, multi-machine inference parallelism, dynamic batching, cache reuse, sticky session routing, load balancing, rate limiting, and degradation ensure the stable supply of token services under high-concurrency and multi-tenant scenarios.

Together, the above technologies constitute a closed loop of inference optimization for token operations. Capability-boundary quantification provides a basis for model scheduling; intelligent routing reduces unnecessary large-model calls; model cascading balances quality and cost; model integration raises the upper bound for complex-task processing; speculative decoding, quantization, distil-

lation, and chain-of-thought optimization reduce the inference overhead of the model itself; key-value cache and semantic-cache reuse reduce repeated computation; optimization of operators, memory, engines, and parallel strategies improves underlying execution efficiency; and gateway routing, load balancing, rate limiting, and degradation ensure service stability. Only by connecting capabilities on the model side, computing-power side, system side, and gateway side can local performance optimization be transformed into full-chain efficiency improvement.

In the future, large-model inference optimization will show a trend from single-point performance optimization toward system-level collaborative optimization. First, optimization objectives will shift from a single performance metric to a multi-objective balance among quality, cost, latency, throughput, stability, and security. Second, the optimization objects will expand from a single model and a single instance to coordination among multiple models, multiple instances, and multiple clusters; intelligent routing, model cascading, model integration, and degradation mechanisms will become foundational capabilities of MaaS platforms. Third, optimization methods will move from static configuration and empirical parameter tuning toward dynamic adaptation; inference systems will adjust scheduling strategies and resource configurations in real time according to factors such as traffic characteristics, model capabilities, cache status, and computing-power load. In this process, long-context and agent applications will further amplify the importance of prefilling, decoding, key-value caching, PD separation, session persistence, semantic caching, dynamic batching, and cross-node scheduling. Heterogeneous computing power and domestic computing-power environments will also drive model-architecture adaptation, operator optimization, quantization compatibility, and inference-engine tuning, making them key areas of focus for large-model inference platforms.

Overall, large-model inference optimization for token operations is fundamentally about achieving a comprehensive optimum in quality, cost, latency, throughput, and stability under real business constraints. In the future, only by continuously advancing the coordinated evolution of multi-model integration, model optimization, model–computing integration, and computing–network–model integration can token production costs be effectively reduced, token service efficiency improved, and token supply stability guaranteed, thereby driving large-model services from "callable" to "operable," and from isolated capability output to scaled intelligent infrastructure.

References

[1] NI S, CHEN G, LI S, et al. A survey on large language model benchmarks[J/OL]. arXiv preprint arXiv:2508.15361, 2025. https://arxiv.org/abs/2508.15361.

[2] CHANG Y, WANG X, WANG J, et al. A survey on evaluation of large language models[J]. ACM transactions on intelligent systems and technology,

2024, 15(3): 1-45.

[3] HENDRYCKS D, BURNS C, BASART S, et al. Measuring massive multitask language understanding[J/OL]. arXiv preprint arXiv:2009.03300, 2020. https://arxiv.org/abs/2009.03300.

[4] SRIVASTAVA A, RASTOGI A, RAO A, et al. Beyond the imitation game: quantifying and extrapolating the capabilities of language models[J]. Transactions on Machine Learning Research, 2023.

[5] LIANG P, BOMMASANI R, LEE T, et al. Holistic evaluation of language models[J]. Transactions on Machine Learning Research, 2023.

[6] ARTIFICIAL ANALYSIS. Artificial Analysis intelligence benchmarking methodology[EB/OL]. https://artificialanalysis.ai/methodology/intelligence-benchmarking.

[7] CHIANG W L, ZHENG L, SHENG Y, et al. Chatbot Arena: an open platform for evaluating LLMs by human preference[C]//Proceedings of the 41st International Conference on Machine Learning. 2024: 8359-8388.

[8] ARENA.AI. Leaderboard[EB/OL]. https://arena.ai/leaderboard.

[9] ZHAO K, LIU Z, LU S, et al. Quantifying capability boundaries: an application-driven analysis for large language model selection[C]//Proceedings of the 2025 9th International Conference on Computer Science and Artificial Intelligence. 2025: 595-601.

[10] ZHAO K, LIU Z, LEI X, et al. Quantifying the capability boundary of DeepSeek models: an application-driven performance analysis[J/OL]. arXiv preprint arXiv:2502.11164, 2025. https://arxiv.org/abs/2502.11164.

[11] LIAN S, ZHAO K, LIU X, et al. What is the best model? Application-driven evaluation for large language models[C]//CCF International Conference on Natural Language Processing and Chinese Computing. Singapore: Springer Nature Singapore, 2024: 67-79.

[12] UNICOMAI. Pinchbench-upgraded[EB/OL]. https://github.com/UnicomAI/pinchbench-upgraded.

[13] LEE C H, CHENG H, OSTENDORF M. OrchestraLLM: Efficient Orchestration of Language Models for Dialogue State Tracking[C]//Proceedings of NAACL-HLT 2024 (Long Papers). 2024: 1434-1445.

[14] AURELIO LABS. Semantic Router[EB/OL]. https://github.com/aurelio-labs/semantic-router.

[15] BERRIAI. LiteLLM: Python SDK & Proxy Server (AI Gateway) for 100+ LLM APIs[EB/OL]. https://github.com/BerriAI/litellm.

[16] DING D, MALLICK A, WANG C, et al. Hybrid LLM: Cost-Efficient and Quality-Aware Query Routing[C]//Proceedings of the 12th International Conference on Learning Representations (ICLR). 2024.

[17] ONG I, ALMAHAIRI A, WU V, et al. RouteLLM: Learning to Route LLMs with Preference Data[J/OL]. arXiv preprint arXiv:2406.18665, 2024. https://arxiv.org/abs/2406.18665.

[18] SHNITZER T, OU A, SILVA M, et al. Large Language Model Routing with Benchmark Datasets[C]//Conference on Language Modeling (COLM). 2024.

[19] CHEN S, JIANG W, LIN B, et al. RouterDC: Query-Based Router by Dual Contrastive Learning for Assembling Large Language Models[C]//Advances in Neural Information Processing Systems (NeurIPS). 2024.

[20] WANG H, POLO F M, SUN Y, et al. Fusing Models with Complementary Expertise[C]//Proceedings of the 12th International Conference on Learning Representations (ICLR). 2024.

[21] AMAZON WEB SERVICES. Amazon Bedrock Intelligent Prompt Routing[EB/OL]. https://aws.amazon.com/bedrock/intelligent-prompt-routing/.

[22] MARTIAN. Model Router[EB/OL]. https://withmartian.com.

[23] NOT DIAMOND. Model Routing for Agents[EB/OL]. https://www.notdiamond.ai.

[24] OPENROUTER. OpenRouter —One API for hundreds of models[EB/OL]. https://openrouter.ai/.

[25] OPENAI. Introducing GPT-5[EB/OL]. (2025-08-07). https://openai.com/index/introducing-gpt-5/.

[26] Alibaba Cloud PAI. Using LLM Intelligent Router to Improve Inference Efficiency[EB/OL]. https://help.aliyun.com/zh/pai/user-guide/use-llm-intelligent-router-to-improve-inference-efficiency.

[27] Volcengine Ark. Intelligent Model Routing[EB/OL]. https://www.volcengine.com/docs/82379/1828788.

[28] CHEN L, ZAHARIA M, ZOU J. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance[J]. Transactions on Machine Learning Research (TMLR), 2024.

[29] YUE M, ZHAO J, ZHANG M, et al. Large Language Model Cascades with Mixture of Thought Representations for Cost-Efficient Reasoning[C]//Proceedings of the 12th International Conference on Learning Representations (ICLR). 2024.

[30] AGGARWAL P, MADAAN A, ANAND A, et al. AutoMix: Automatically Mixing Language Models[C]//Advances in Neural Information Processing Systems (NeurIPS). 2024.

[31] WANG Y, CHEN K, TAN H, et al. Tabi: An Efficient Multi-Level Inference System for Large Language Models[C]//Proceedings of the 18th European Conference on Computer Systems (EuroSys). 2023.

[32] ZHANG J, KRISHNA R, AWADALLAH A H, et al. EcoAssistant: Using LLM Assistant More Affordably and Accurately[J/OL]. arXiv preprint

arXiv:2310.03046, 2023. https://arxiv.org/abs/2310.03046.

[33] LEVIATHAN Y, KALMAN M, MATIAS Y. Fast Inference from Transformers via Speculative Decoding[C]//Proceedings of the 40th International Conference on Machine Learning (ICML). 2023.

[34] DATABRICKS. Unity AI Gateway: Configure Fallbacks on Model Serving Endpoints[EB/OL]. https://docs.databricks.com/aws/en/ai-gateway/.

[35] WANG X, WEI J, SCHUURMANS D, et al. Self-Consistency Improves Chain of Thought Reasoning in Language Models[C]//Proceedings of the 11th International Conference on Learning Representations (ICLR). 2023.

[36] LI J, ZHANG Q, YU Y, et al. More Agents Is All You Need[J]. Transactions on Machine Learning Research (TMLR), 2024.

[37] JIANG D, REN X, LIN B Y. LLM-Blender: Ensembling Large Language Models with Pairwise Ranking and Generative Fusion[C]//Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL). 2023.

[38] WAN F, HUANG X, CAI D, et al. Knowledge Fusion of Large Language Models[C]//Proceedings of the 12th International Conference on Learning Representations (ICLR). 2024.

[39] WANG J, WANG J, ATHIWARATKUN B, et al. Mixture-of-Agents Enhances Large Language Model Capabilities[C]//Conference on Language Modeling (COLM). 2024.

[40] SUN Y, LI Z, ZHANG Y, et al. Efficient attention mechanisms for large language models: a survey[J/OL]. arXiv preprint arXiv:2507.19595, 2025. https://arxiv.org/abs/2507.19595.

[41] YANG A, LI A, YANG B, et al. Qwen3 technical report[J/OL]. arXiv preprint arXiv:2505.09388, 2025. https://arxiv.org/pdf/2505.09388.

[42] LIU A, FENG B, XUE B, et al. Deepseek-V3 technical report[J/OL]. arXiv preprint arXiv:2412.19437, 2024. https://arxiv.org/pdf/2412.19437.

[43] CHEN A, LI A, ZHOU B, et al. The MiniMax-M2 Series: Mini Activations Unleashing Max Real-World Intelligence[J/OL]. arXiv preprint arXiv:2605.26494, 2026. https://arxiv.org/pdf/2605.26494.

[44] DAO T, FU D Y, ERMON S, et al. FlashAttention: fast and memory-efficient exact attention with IO-awareness[J]. Advances in Neural Information Processing Systems. 2022, 35: 16344-16359.

[45] DAO T. FlashAttention-2: faster attention with better parallelism and work partitioning[C]//International Conference on Learning Representations. 2024, 2024: 35549-35562.

[46] DEEPSEEK-AI. DeepSeek-V4: towards highly efficient million-token context intelligence[R/OL]. 2026. https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro/blob/main/DeepSeek_V4.pdf.

[47] ZENG A, LV X, HOU Z, et al. Glm-5: from vibe coding to agentic engineering[J]. arXiv preprint arXiv:2602.15763, 2026. https://arxiv.org/abs/2602.15763.

[48] KWON W, LI Z, ZHUANG S, et al. Efficient memory management for large language model serving with PagedAttention[C]//Proceedings of the 29th Symposium on Operating Systems

Principles. New York: ACM, 2023: 611-626.

[49] MINIMAX RESEARCH. MiniMax M3: Frontier Coding, 1M Context, Native Multimodality —All in One Model[EB/OL]. 2026. https://www.minimax.io/blog/minimax-m3.

[50] QWEN TEAM. Qwen3.5-Omni Technical Report[J/OL]. arXiv preprint arXiv:2604.15804, 2026. https://arxiv.org/abs/2604.15804.

[51] DEEPSEEK-AI, LIU A, MEI A, et al. DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models[J/OL]. arXiv preprint arXiv:2512.02556, 2025. https://arxiv.org/abs/2512.02556.

[52] CAI W, JIANG J, WANG F, et al. A survey on mixture of experts in large language models[J]. IEEE Transactions on Knowledge and Data Engineering, 2025.

[53] LEPIKHIN D, LEE H, XU Y, et al. GShard: scaling giant models with conditional computation and automatic sharding[C]//International Conference on Learning Representations. 2021.

[54] FEDUS W, ZOPH B, SHAZEER N. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity[J]. Journal of Machine Learning Research, 2022, 23(120): 1-39.

[55] JIANG A Q, SABLAYROLLES A, ROUX A, et al. Mixtral of experts[J/OL]. arXiv preprint arXiv:2401.04088, 2024. https://arxiv.org/abs/2401.04088.

[56] DAI D, DENG C, ZHAO C, et al. DeepSeekMoE: towards ultimate expert specialization in mixture-of-experts language models[C]//Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2024: 1280-1297.

[57] DEEPSEEK-AI, LIU A, FENG B, et al. DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model[J/OL]. arXiv preprint arXiv:2405.04434, 2024. https://arxiv.org/abs/2405.04434.

[58] NVIDIA. TensorRT-LLM expert parallelism documentation[EB/OL]. https://nvidia.github.io/TensorRT-LLM/1.1.0rc1/advanced/expert-parallelism.html.

[59] HE S, CAI W, HUANG J, et al. Capacity-aware inference: mitigating the straggler effect in mixture of experts[J/OL]. arXiv preprint arXiv:2503.05066,

2025. https://arxiv.org/abs/2503.05066.

[60] MA Z, LIU X, LIU Z, et al. Mixture of heterogeneous grouped experts for language modeling[J/OL]. arXiv preprint arXiv:2604.23108, 2026. https://arxiv.org/abs/2604.23108.

[61] CHEN P, ZHANG X, LIU Z, et al. Optimizing for the shortest path in denoising diffusion model[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. CVPR, 2025.

[62] LIU J C, WANG X Y, LIN Y Q, et al. A survey on cache methods in diffusion models: Toward efficient multi-modal generation[J/OL]. arXiv preprint arXiv:2510.19755, 2025. https://arxiv.org/abs/2510.19755.

[63] MA X, FANG G, WANG X. DeepCache: Accelerating diffusion models for free[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. CVPR, 2024.

[64] LIU F, ZHANG S, WANG X, et al. Timestep Embedding Tells: It' s Time to Cache for Video Diffusion Model[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. CVPR, 2025. https://arxiv.org/abs/2411.19108.

[65] ZOU C, ZHANG E, GUO R, et al. Accelerating diffusion transformers with dual feature caching[J/OL]. arXiv preprint arXiv:2412.18911, 2024. https://arxiv.org/abs/2412.18911.

[66] GAO H L, CHEN P, SHI F Y, et al. LeMiCa: Lexicographic minimax path caching for efficient diffusion-based video generation[C]//Advances in Neural Information Processing Systems. NeurIPS, 2025.

[67] GAO H L, CHEN P, SHI F Y, et al. MeanCache: From instantaneous to average velocity for accelerating flow matching inference[C]//Proceedings of the International Conference on Learning Representations. ICLR, 2026.

[68] WEI J, WANG X, SCHUURMANS D, et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models[C]//Advances in Neural Information Processing Systems. NeurIPS, 2022.

[69] ZHOU D, SCHARLI N, HOU L, et al. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models[C]//Proceedings of the 11th International Conference on

Learning Representations. ICLR, 2023.

[70] ZELIKMAN E, WU Y, MU J, et al. STaR: Bootstrapping Reasoning With Reasoning[C]//Advances in Neural Information Processing Systems. NeurIPS, 2022.

[71] YAO S, ZHAO J, YU D, et al. ReAct: Synergizing Reasoning and Acting in Language Models[C]//Proceedings of the 11th International Conference on Learning Representations. ICLR, 2023.

[72] YAO S, YU D, ZHAO J, et al. Tree of Thoughts: Deliberate Problem Solving with Large Language Models[C]//Advances in Neural Information Processing Systems. NeurIPS, 2023.

[73] GAO L, MADAAN A, ZHOU S, et al. PAL: Program-aided Language Models[C]//Proceedings of the 40th International Conference on Machine Learning. ICML, 2023.

[74] OPENAI. Introducing OpenAI o1-preview[EB/OL]. 2024. https://openai.com/index/introducing-openai-o1-preview/.

[75] OPENAI. OpenAI o1-mini: Advancing cost-efficient reasoning[EB/OL]. 2024. https://openai.com/index/openai-o1-mini-advancing-cost-efficient-reasoning/.

[76] DEEPSEEK-AI. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning[J/OL]. arXiv preprint arXiv:2501.12948v2, 2025. https://arxiv.org/abs/2501.12948v2.

[77] ANTHROPIC. Claude 3.7 Sonnet and Claude Code[EB/OL]. 2025. https://www.anthropic.com/news/claude-3-7-sonnet.

[78] QWEN TEAM. Qwen3: Think Deeper, Act Faster[EB/OL]. 2025. https://qwenlm.github.io/blog/qwen3/.

[79] GOOGLE. Gemini thinking[EB/OL]. 2025. https://ai.google.dev/gemini-api/docs/thinking.

[80] HAN T, WANG Z, FANG C, et al. Token-Budget-Aware LLM Reasoning[J/OL]. arXiv preprint arXiv:2412.18547, 2024. https://arxiv.org/abs/2412.18547.

[81] ARORA D, ZANETTE A. Training Language Models to Reason Efficiently[J/OL]. arXiv preprint arXiv:2502.04463, 2025. https://arxiv.org/abs/2502.04463.

[82] XIANG V, BLAGDEN C, RAFAILOV R, et al. Just Enough Thinking: Efficient Reasoning with Adaptive Length Penalties Reinforcement Learning[J/OL]. arXiv preprint arXiv:2506.05256, 2025. https://arxiv.org/abs/2506.05256.

[83] XU S, LIU Z, GAN Y, et al. Chain of Draft: Thinking Faster by Writing Less[J/OL]. arXiv preprint arXiv:2502.18600, 2025. https://arxiv.org/abs/2502.18600.

[84] SONG M, ZHENG M. Walk Before You Run! Concise LLM Reasoning via Reinforcement Learning[J/OL]. arXiv preprint arXiv:2505.21178, 2025. https://arxiv.org/abs/2505.21178.

[85] WANG C, FENG Y, CHEN D, et al. Wait, We Don' t Need to "Wait" ! Removing Thinking Tokens Improves Reasoning Efficiency[C]//Findings of the Association for Computational Linguistics: EMNLP 2025. 2025: 7459-7482. https://aclanthology.org/2025.findings-emnlp.394/. DOI: 10.18653/v1/2025.findings-emnlp.394.

[86] HAO S, CHEN Z, ZHANG Y, et al. COCONUT: Training Large Language Models to Reason with Continuous Latent Thought[J/OL]. arXiv preprint

arXiv:2412.06769, 2024.

[87] FAN S, HAN P, SHANG S, et al. CoThink: Token-Efficient Reasoning via Instruct Models Guiding Reasoning Models[J/OL]. arXiv preprint arXiv:2505.22017v1, 2025. https://arxiv.org/abs/2505.22017v1.

[88] SANG H, XU Y, ZHOU Z, et al. Not all tokens are needed(NAT): token efficient reinforcement learning[J/OL]. arXiv preprint arXiv:2603.06619, 2026. https://arxiv.org/abs/2603.06619.

[89] SUI Y, CHUANG Y N, WANG G, et al. Stop Overthinking: A Survey on Efficient Reasoning for Large Language Models[J/OL]. arXiv preprint arXiv:2503.16419, 2025. https://arxiv.org/abs/2503.16419.

[90] SHEN Y, ZHANG J, HUANG J, et al. DAST: Difficulty-Adaptive Slow-Thinking for Large Reasoning Models[J/OL]. arXiv preprint arXiv:2503.04472, 2025. https://arxiv.org/abs/2503.04472.

[91] ZHONG W, GUO L, GAO Q, et al. MemoryBank: Enhancing Large Language Models with Long-Term Memory[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2024, 38(17): 19724-19731. https://doi.org/10.1609/aaai.v38i17.29946.

[92] PACKER C, LI V, LIN M, et al. MemGPT: Towards LLMs as Operating Systems[J/OL]. arXiv preprint arXiv:2310.08560, 2023. https://arxiv.org/abs/2310.08560.

[93] SHINN N, LABASH P, GUPTA A, et al. Reflexion: Language Agents with Verbal

Reinforcement Learning[J/OL]. arXiv preprint arXiv:2303.11366, 2023. https://arxiv.org/abs/2303.11366.

[94] WANG G, XIE Y, JIANG Y, et al. Voyager: An Open-Ended Embodied Agent with Large Language Models[J/OL]. arXiv preprint arXiv:2305.16291, 2023.
https://arxiv.org/abs/2305.16291.

[95] ZHAO A, HUANG D, XU Q, et al. ExpeL: LLM Agents Are Experiential Learners[J/OL]. arXiv preprint arXiv:2308.10144, 2023. https://arxiv.org/abs/2308.10144.

[96] CHHIKARA P, et al. Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory[EB/OL]. https://github.com/mem0ai/mem0.

[97] RASMUSSEN T, et al. Zep: A Temporal Knowledge Graph Memory for AI Agents[EB/OL]. https://github.com/getzep/zep.

[98] XU X, LIANG Z, GAO H, et al. A-MEM: Agentic Memory for LLM Agents[J/OL]. arXiv preprint arXiv:2502.12110, 2025.
https://arxiv.org/abs/2502.12110.

[99] ZHANG Y, FU M, WAN G, et al. G-Memory: Hierarchical Graph Memory for LLM Agents[J/OL]. arXiv preprint arXiv:2506.07398, 2025.

https://arxiv.org/abs/2506.07398.

[100] YU H, CHEN T, FENG J, et al. MemAgent: Reshaping Long-Context LLM with Multi-Conv RL-Based Memory Agent[J/OL]. arXiv preprint arXiv:2507.02259, 2025.
https://arxiv.org/abs/2507.02259.

[101] ZHOU Z, QU A, WU Z, et al. Mem1: Learning to Summarize and Manage Memory with Reinforcement Learning[J/OL]. arXiv preprint arXiv:2506.15841, 2025.
https://arxiv.org/abs/2506.15841.

[102] YAN Z, et al. Memory-R1: Enhancing Agent Memory via Reinforcement Learning[J/OL]. arXiv preprint arXiv:2508.19828, 2025.
https://arxiv.org/abs/2508.19828.

[103] WANG Y, GAO Y, CHEN X, et al. MemoryLLM: Towards Self-Updatable Large Language Models[J/OL]. arXiv preprint arXiv:2402.04624, 2024.
https://arxiv.org/abs/2402.04624.

[104] WANG Y, KROTOV D, HU Y, et al. M+: Extending LLMs with Long-Term Memory Tokens[J/OL]. arXiv preprint arXiv:2502.00592, 2025.
https://arxiv.org/abs/2502.00592.

[105] ZHANG Y, FU M, YAN S. MemGen: Generative Latent Memory for LLM Agents[J/OL]. arXiv preprint arXiv:2509.24704, 2025.
https://arxiv.org/abs/2509.24704.

[106] JAVIDNIA N, ROUHANI B D, KOUSHANFAR F. Key, value, compress: a systematic exploration of KV cache compression techniques[C]//2025 IEEE Custom Integrated Circuits Conference (CICC). IEEE, 2025: 1-3.

[107] VLLM PROJECT. Automatic prefix caching design documentation[EB/OL]. https://docs.vllm.ai/en/latest/design/prefix_caching/.

[108] ZHENG L, YIN L, XIE Z, et al. SGLang: efficient execution of structured language model programs[J]. Advances in neural information processing systems, 2024, 37: 62557-62583.

[109] LIU Z, DESAI A, LIAO F, et al. Scissorhands: exploiting the persistence of importance hypothesis for LLM KV cache compression at test time[J]. Advances in Neural Information Processing Systems, 2023, 36: 52342-52364.

[110] ZHANG Z, SHENG Y, ZHOU T, et al. H2O: heavy-hitter oracle for efficient generative inference of large language models[J]. Advances in Neural Information Processing Systems, 2023, 36: 34661-34710.

[111] LI Y, HUANG Y, YANG B, et al. SnapKV: LLM knows what you are looking for before generation[J]. Advances in Neural Information Processing Systems, 2024, 37: 22947-22970.

[112] LIU Z, YUAN J, JIN H, et al. KIVI: a tuning-free asymmetric 2bit quantization for KV cache[J/OL]. arXiv preprint arXiv:2402.02750, 2024. https://arxiv.org/abs/2402.02750.

[113] KIM J H, KIM S, LEE J, et al. KVzip: query-agnostic KV cache compression with context reconstruction[J]. Advances in Neural Information Processing Systems, 2026, 38: 167563-167591.

[114] ZANDIEH A, HAN I, DALIRI M, et al. TurboQuant: online vector quantization with near-optimal distortion rate[J/OL]. arXiv preprint arXiv:2504.19874, 2025.
https://arxiv.org/abs/2504.19874.

[115] MU J, LI X L, GOODMAN N. Learning to Compress Prompts with Gist Tokens[J/OL]. arXiv preprint arXiv:2304.08467, 2023. https://arxiv.org/abs/2304.08467.

[116] CHEVALIER A, WETTIG A, AJITH A, et al. Adapting Language Models to Compress Contexts[J/OL]. arXiv preprint arXiv:2305.14788, 2023. https://arxiv.org/abs/2305.14788.

[117] GE T, HU J, WANG L, et al. In-context Autoencoder for Context Compression in a Large Language Model[J/OL]. arXiv preprint arXiv:2307.06945, 2023. https://arxiv.org/abs/2307.06945.

[118] JIANG H, WU Q, LIN C Y, et al. LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models[J/OL]. arXiv preprint arXiv:2310.05736, 2023. https://arxiv.org/abs/2310.05736.

[119] JIANG H, WU Q, LUO X, et al. LongLLMLingua: Accelerating and Enhancing LLMs in Long Context Scenarios via Prompt Compression[J/OL]. arXiv preprint arXiv:2310.06839, 2023. https://arxiv.org/abs/2310.06839.

[120] JIANG H, LI Y, ZHANG C, et al. MInference 1.0: Accelerating Pre-filling for Long-Context LLMs via Dynamic Sparse Attention[J/OL]. arXiv preprint arXiv:2407.02490, 2024. https://arxiv.org/abs/2407.02490.

[121] CHENG X, WANG X, ZHANG X, et al. xRAG: Extreme Context Compression for Retrieval-augmented Generation with One Token[J/OL]. arXiv preprint arXiv:2405.13792, 2024. https://arxiv.org/abs/2405.13792.

[122] CHEN C, et al. Accelerating Large Language Model Decoding with Speculative Sampling[J/OL]. arXiv preprint arXiv:2302.01318, 2023. https://arxiv.org/abs/2302.01318.

[123] GOOGLE RESEARCH. Looking back at speculative decoding[EB/OL]. https://research.google/blog/looking-back-at-speculative-decoding/.

[124] MIAO X, et al. SpecInfer: Accelerating Generative Large Language Model Serving with Tree-based Speculative Inference and Verification[C]//Proceedings of the 2024 International Conference on Machine Learning. 2024.

[125] CAI T, et al. Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads[J/OL]. arXiv preprint arXiv:2401.10774, 2024. https://arxiv.org/abs/2401.10774.

[126] ELHOUSHI M, et al. LayerSkip: Enabling Early Exit Inference and Self-Speculative Decoding[C]//Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics. ACL, 2024.

[127] LI Y, et al. EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty[C]//Proceedings of the 41st International Conference on Machine Learning. ICML, 2024.

[128] LI Y, et al. EAGLE-2: Faster Inference of Language Models with Dynamic Draft Trees[C]. Proceedings of the 2024 conference on empirical methods in natural language processing. EMNLP, 2024.

[129] LI Y, et al. EAGLE-3: Scaling up Inference Acceleration of Large Language Models via Training-Time Test[C]. Proceedings of the 2026 conference on Advances in Neural Information Processing Systems. NeurIPS, 2026.

[130] VLLM PROJECT. Speculative Decoding[EB/OL]. https://docs.vllm.ai/en/latest/features/speculative_d

[131] SGLANG DOCUMENTATION. Speculative Decoding[EB/OL]. https://sglang.ai/.

[132] NVIDIA. TensorRT-LLM Speculative Decoding Boosts Inference Throughput by up to 3.6x[EB/OL]. https://developer.nvidia.com/blog/tensorrt-llm-speculative-decoding-boosts-inference-throughput-by-up-to-3-6x/.

[133] CHENG Y, et al. Recurrent Drafter for Fast Speculative Decoding in Large Language Models[J/OL]. arXiv preprint arXiv:2403.09919, 2024. https://arxiv.org/abs/2403.09919.

[134] LI S, WANG C, ZHU Y, et al. SpecForge: A Flexible and Efficient Open-Source Training Framework for Speculative Decoding[J/OL]. arXiv preprint arXiv:2603.18567, 2026. https://arxiv.org/abs/2603.18567.

[135] HUI M, HUANG X, SALAS J C, et al. P-EAGLE: Parallel-Drafting EAGLE with Scalable Training[J/OL]. arXiv preprint arXiv:2602.01469, 2026. https://arxiv.org/abs/2602.01469.

[136] ABRAMOVICH T, ASHKENAZI M, PUTTERMAN I, et al. SPEED-Bench: A Unified and Diverse Benchmark for Speculative Decoding[J/OL]. arXiv preprint arXiv:2604.09557, 2026. https://arxiv.org/abs/2604.09557.

[137] GONG R, DING Y, WANG Z, et al. A survey of low-bit large language models: basics, systems, and algorithms[J]. Neural Networks, 2025, 192: 107856.

[138] FRANTAR E, ASHKBOOS S, HOEFLER T, et al. GPTQ: accurate post-training quantization for generative pre-trained transformers[J/OL]. arXiv preprint arXiv:2210.17323,

2022. https://arxiv.org/abs/2210.17323.

[139] LIN J, TANG J, TANG H, et al. AWO: activation-aware weight quantization for LLM compression and acceleration[J]. Proceedings of machine learning and systems, 2024, 6: 87-100.

[140] XIAO G, LIN J, SEZNEC M, et al. SmoothQuant: accurate and efficient post-training quantization for large language models[C]//Proceedings of the 40th International Conference on Machine Learning. PMLR, 2023: 38087-38099.

[141] NVIDIA. TensorRT-LLM quantization documentation[EB/OL]. https://nvidia.github.io/TensorRT-LLM/latest/features/quantization.html.

[142] VLLM PROJECT. Quantization documentation[EB/OL]. https://docs.vllm.ai/en/latest/features/quantiz

[143] DETTMERS T, PAGNONI A, HOLTZMAN A, et al. QLoRA: efficient finetuning of quantized LLMs[J]. Advances in neural information processing systems, 2023, 36: 10088-10115.

[144] LIU Y, QIU S, ZHANG C, et al. A comprehensive evaluation on quantization techniques for large language models[J/OL]. arXiv preprint arXiv:2507.17417, 2025. https://arxiv.org/abs/2507.17417.

[145] HINTON G, VINYALS O, DEAN J. Distilling the Knowledge in a Neural Network[J/OL]. arXiv preprint arXiv:1503.02531, 2015. https://arxiv.org/abs/1503.02531.

[146] SANH V, DEBUT L, CHAUMOND J, et al. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter[J/OL]. arXiv preprint arXiv:1910.01108, 2019. https://arxiv.org/abs/1910.01108.

[147] WANG W, WEI F, DONG L, et al. MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers[C]//Advances in Neural Information Processing Systems. 2020, 33: 5776-5788.

[148] WANG Y, KORDI Y, MISHRA S, et al. Self-Instruct: Aligning Language Models with Self-Generated Instructions[J/OL]. arXiv preprint arXiv:2212.10560, 2022. https://arxiv.org/abs/2212.10560.

[149] GUDIBANDE A, WALLACE E, SNELL C, et al. The False Promise of Imitating Proprietary LLMs[J/OL]. arXiv preprint arXiv:2305.15717, 2023. https://arxiv.org/abs/2305.15717.

[150] HSIEH C Y, LI C L, YEH C K, et al. Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller Model Sizes[J/OL]. arXiv preprint arXiv:2305.02301, 2023. https://arxiv.org/abs/2305.02301.

[151] MUKHERJEE S, MITRA A, JAWAHAR G, et al. Orca: Progressive Learning from Complex Explanation Traces of GPT-4[J/OL]. arXiv preprint arXiv:2306.02707, 2023. https://arxiv.org/abs/2306.02707.

[152] ZENG A, LIU M, LU R, et al. AgentTuning: Enabling Generalized Agent Abilities for LLMs[J/OL]. arXiv preprint arXiv:2310.12823, 2023.

https://arxiv.org/abs/2310.12823.

[153] CHEN Z, ZHAO Y, WANG Z, et al. Magpie: Alignment Data Synthesis from Scratch by Prompting Aligned LLMs with Nothing[J/OL]. arXiv preprint arXiv:2406.08464, 2024. https://arxiv.org/abs/2406.08464.

[154] GU Y, DONG L, WEI F, et al. Knowledge Distillation of Large Language Models (MiniLLM)[J/OL]. arXiv preprint arXiv:2306.08543, 2023. https://arxiv.org/abs/2306.08543.

[155] AGARWAL R, VIEILLARD N, STANCZYK P, et al. GKD: Generalized Knowledge Distillation for Auto-regressive Sequence Models[J/OL]. arXiv preprint arXiv:2306.13649v1, 2023. https://arxiv.org/abs/2306.13649v1.

[156] LEE H, PHATALE S, MANSOOR H, et al. RLAIF vs. RLHF: Scaling Reinforcement Learning from Human Feedback with AI Feedback. International Conference on Machine Learning (2023).

[157] ZHANG W, YAN J, HUANG J, et al. HEAL: Hindsight Entropy-Assisted Learning for Reasoning Distillation[J/OL]. arXiv preprint arXiv:2603.10359, 2026. https://arxiv.org/abs/2603.10359.

[158] MICROSOFT. Graph optimizations in ONNX Runtime[EB/OL]. https://onnxruntime.ai/docs/.

[159] SHAH J, BIKSHANDI G, ZHANG Y, et al. FlashAttention-3: Fast and accurate attention with asynchrony and low-precision[C]//Advances in Neural Information Processing Systems. NeurIPS, 2024.

[160] NVIDIA CORPORATION. Attention: NVIDIA cuDNN Frontend documentation[EB/OL].

https://docs.nvidia.com/deeplearning/cudnn/frontend/latest/.

[161] NVIDIA CORPORATION. Grouped GEMM + GLU: NVIDIA cuDNN Frontend documentation[EB/OL]. https://docs.nvidia.com/deeplearning/cudnn/frontend/latest/.

[162] YE Z, CHEN L, LAI R, et al. FlashInfer: Efficient and Customizable Attention Engine for LLM Inference Serving[C/OL]//Proceedings of MLSys. 2025.

[163] QIN R, LI Z, HE W, et al. Mooncake: A KVCache-centric Disaggregated Architecture for LLM Serving[EB/OL]. arXiv preprint arXiv:2407.00079, 2024. https://arxiv.org/abs/2407.00079.

[164] NVIDIA. NVIDIA cuDNN Documentation[EB/OL]. https://docs.nvidia.com/deeplearning/cudnn/.

[165] NVIDIA. NVIDIA cuBLAS[EB/OL]. https://developer.nvidia.com/cublas.

[166] NVIDIA. CUTLASS 4.5.0 Overview[EB/OL]. https://github.com/NVIDIA/cutlass.

[167] ZHAO C, XU Z, ZHAO L, et al. DeepGEMM: clean and efficient BLAS kernel library on GPU[EB/OL]. GitHub. https://github.com/deepseek-ai/DeepGEMM.

[168] SU Z, FU R, CAO W, et al. TMA-Adaptive FP8 Grouped GEMM: eliminating padding requirements in low-precision training and inference on Hopper[EB/OL]. arXiv preprint arXiv:2508.16584, 2025. https://arxiv.org/abs/2508.16584.

[169] VLLM TEAM. Optimization and tuning; parameter sweeps: vLLM documentation[EB/OL]. https://docs.vllm.ai/.

[170] AGRAWAL A, KEDIA N, PANWAR A, et al. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve[C]//Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI ' 24). 2024: 117-134.

[171] ZHONG Y, LIU S, CHEN J, et al. DistServe: disaggregating prefill and decoding for goodput-optimized large language model serving[C]//Proceedings of the USENIX Symposium on Operating Systems Design and Implementation. OSDI, 2024.

[172] SGLANG TEAM. SGLang documentation: advanced features, hyperparameter tuning and PD disaggregation[EB/OL]. https://sglang.ai/.

[173] NVIDIA. TensorRT-LLM documentation: paged attention, in-flight batching, request scheduling and KV cache system[EB/OL]. https://nvidia.github.io/TensorRT-LLM/.

[174] NVIDIA. NVIDIA Dynamo documentation: disaggregated serving[EB/OL]. https://docs.nvidia.com/.

[175] HUGGING FACE. Text Generation Inference (TGI) documentation[EB/OL]. https://huggingface.co/docs/text-generation-inference.

[176] Ascend Community. MindIE LLM Development Guide: Quick Introduction; MindIE-LLM repository[EB/OL]. https://www.hiascend.com/.

[177] HUGGING FACE. Customizing models: Hugging Face Transformers Documentation[EB/OL]. https://huggingface.co/docs/transformers/.

[178] VLLM PROJECT. Supported Models: vLLM Documentation[EB/OL]. https://docs.vllm.ai/en/latest/models/supported_models.html.

[179] DEEPSEEK-AI. FlashMLA: Efficient Multi-head Latent Attention Kernels[EB/OL]. https://github.com/deepseek-ai/FlashMLA.

[180] SGLANG PROJECT. SGLang Documentation: Welcome and Expert Parallelism[EB/OL]. https://sglang.ai/.

[181] NVIDIA. TensorRT LLM Overview / TensorRT LLM Developer Page[EB/OL]. https://developer.nvidia.com/tensorrt-llm.

[182] DEEPSEEK-AI. EPLB: Expert Parallelism Load Balancer[EB/OL]. https://github.com/deepseek-ai/EPLB.

[183] VLLM PROJECT. vLLM Serving Documentation: Parallelism and Scaling; Expert Parallel Deployment[EB/OL]. https://docs.vllm.ai/.

[184] SGLANG TEAM. Deploying DeepSeek with PD Disaggregation and Large-Scale Expert Parallelism on 96 H100 GPUs[EB/OL]. https://sglang.ai/.

[185] DEEPSEEK-AI. DeepEP: An Efficient Expert-Parallel Communication Library[EB/OL]. https://github.com/deepseek-ai/DeepEP.

[186] DEEPSEEK-AI. LPLB: Linear-Programming-Based Load Balancer[EB/OL]. https://github.com/deepseek-ai/LPLB.

[187] PRABHU R, NAYAK A, MOHAN J, et al. vAttention: dynamic memory management for serving LLMs without PagedAttention[EB/OL]. arXiv preprint arXiv:2405.04437, 2024.

https://arxiv.org/abs/2405.04437.

[188] LIU Y, CHENG Y, YAO J, et al. LMCache: an efficient KV cache layer for enterprise-scale LLM inference[EB/OL]. arXiv preprint arXiv:2510.09665, 2025. https://arxiv.org/abs/2510.09665.

[189] NVIDIA TENSORRT-LLM TEAM. Disaggregated Serving in TensorRT LLM[EB/OL]. https://nvidia.github.io/TensorRT-LLM/.

[190] NVIDIA. Dynamo documentation: introduction[EB/OL]. https://docs.nvidia.com/.

[191] BANG et al. GPTCache: A Library for Creating Semantic Cache for LLM Queries[EB/OL]. https://github.com/zilliztech/GPTCache.

[192] ZHANG Z, LIU Z, XIE Y, et al. From Similarity to Vulnerability: Key Collision Attack on LLM Semantic Caching[J/OL]. arXiv preprint arXiv:2601.23088, 2026. https://arxiv.org/abs/2601.23088.

[193] RUAN C, BI C, ZHENG K, et al. Cortex: Achieving Low-Latency, Cost-Efficient Remote Data Access For LLM via Semantic-Aware Knowledge Caching[J/OL]. arXiv preprint arXiv:2509.17360, 2025. https://arxiv.org/abs/2509.17360.

[194] NVIDIA. Batchers; Ragged Batching: Triton Inference Server User Guide[EB/OL]. https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/.

[195] YU G I, JEONG J S, KIM G W, et al. Orca: A Distributed Serving System for Transformer-Based Generative Models[C]//Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI ' 22). 2022: 521-538.

[196] SUN B, HUANG Z, ZHAO H, et al. Llumnix: Dynamic Scheduling for Large Language Model Serving[C]//Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI ' 24). 2024: 173-191.

[197] PATKE A, REDDY D, JHA S, et al. Queue Management for SLO-Oriented Large Language Model Serving[C]//Proceedings of the 2024 ACM Symposium on Cloud Computing (SoCC 2024). 2024: 18-35.

[198] ZHU K, GAO Y, ZHAO Y, et al. NanoFlow: Towards Optimal Large Language Model Serving Throughput[C]//Proceedings of the 19th USENIX Symposium on Operating Systems Design and Implementation (OSDI ' 25). 2025: 749-765.

[199] XIANG Y, LI X, QIAN K, et al. Aegaeon: Effective GPU Pooling for Concurrent LLM Serving on the Market[C]//Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles (SOSP '25). 2025: 1030-1045.

[200] NVIDIA. Overview of NCCL: NVIDIA Collective Communications Library Documentation[EB/OL]. https://docs.nvidia.com/deeplearning/nccl/.

[201] OPENUCX. Unified Communication X: OpenUCX Documentation and Source Repository[EB/OL]. https://openucx.org/.

[202] JEAUGEY S, CONGIU G, GILLIS T, et al. New Scaling Algorithm and Initialization with NVIDIA Collective Communications Library 2.23[EB/OL]. NVIDIA Developer Blog. https://developer.nvidia.com/blog/.

[203] GILLIS T, MUBARAK M, NICELY M. NCCL Deep Dive: Cross Data Center Communication and Network Topology Awareness[EB/OL]. NVIDIA Developer Blog. https://developer.nvidia.com/blog/.

[204] NVIDIA. NVIDIA Collective Communications Library: Reliability, Availability and Serviceability Documentation[EB/OL]. https://docs.nvidia.com/deeplearning/nccl/.

[205] LEE S, KHAZRAEE M. Enhancing Distributed Inference Performance with the NVIDIA Inference Transfer Library[EB/OL]. NVIDIA Developer Blog. https://developer.nvidia.com/blog/.

[206] CLOUDFLARE. What is load balancing?[EB/OL]. https://www.cloudflare.com/learning/performance/wh
is-load-balancing/.

[207] ENVOY PROXY. Architecture Overview[EB/OL]. https://www.envoyproxy.io/docs/envoy/latest/intro/a

[208] HIGRESS. Higress website[EB/OL]. https://higress.io/.

[209] ENVOY GATEWAY. Rate limiting[EB/OL]. https://gateway.envoyproxy.io/docs/concepts/rate-limiting/.

[210] SENTINEL. System adaptive protection[EB/OL]. https://sentinelguard.io/zh-cn/docs/golang/system-adaptive-protection.html.

[211] ONEUPTIME. WASM plugins rate limiting Istio[EB/OL]. (2026-02-24). https://oneuptime.com/blog/post/2026-02-24-wasm-plugins-rate-limiting-istio/view.

[212] INWORLD AI. Best LLM gateways[EB/OL]. https://inworld.ai/resources/best-llm-gateways.

[213] XIAO B, XIA B, YANG B, et al. Mimo-v2-flash technical report[J/OL]. arXiv preprint arXiv:2601.02780, 2026. https://arxiv.org/abs/2601.02780.

[214] GU A, DAO T. Mamba: Linear-time sequence modeling with selective state spaces[J/OL]. arXiv preprint arXiv:2312.00752, 2023. https://arxiv.org/abs/2312.00752.

[215] ZUO J, VELIKANOV M, RHAIEM D E, et al. Falcon mamba: The first competitive attention-free 7b language model[J/OL]. arXiv preprint arXiv:2410.05355, 2024. https://arxiv.org/abs/2410.05355.

[216] LIEBER O, LENZ B, BATA H, et al. Jamba: A hybrid transformer-mamba language model[J/OL]. arXiv preprint arXiv:2403.19887, 2024. https://arxiv.org/abs/2403.19887.

Note: Figure translations are in progress. See original paper for figures.

Source: ChinaXiv. Machine translation. Verify with the original.