System Design for AI Engineering: Adaptive Architectures for Real-World Scalable AI Applications

Abhishek Shukla
Syracuse University
United States of America

ABSTRACT
The rapid advancement of Artificial Intelligence (AI) necessitates robust system architectures to ensure scalability, reliability, and efficiency across diverse applications. This paper proposes a comprehensive framework for designing AI engineering systems, addressing critical components such as data pipelines, computer architectures, model serving, distributed training, and emerging patterns like federated learning and serverless AI. We introduce novel orchestration techniques, hybrid cloud-edge architectures, and ethical considerations to enhance system robustness. Through detailed case studies on recommendation systems, autonomous driving, and healthcare diagnostics, we illustrate practical implementations and analyze trade-offs. Challenges such as data privacy, resource optimization, and model governance are explored, with future directions emphasizing sustainable AI and quantum computing. This framework serves as a blueprint for engineers building next-generation AI systems.

Keywords
AI Engineering, System Design, Scalable AI, Distributed Systems, Model Serving, Federated Learning, Cloud-Edge Architectures.

1. INTRODUCTION
Artificial Intelligence (AI) and Machine Learning (ML) underpin transformative applications, from personalized content delivery to autonomous vehicles and medical diagnostics. Unlike traditional software, AI systems demand specialized architectures to handle massive datasets, computer intensive training, and low-latency inference [1]. Designing scalable AI systems involves unique challenges, including distributed processing, continuous monitoring, and compliance with ethical standards [2]. This paper presents an enhanced system design framework for AI engineering, focusing on scalability to support millions of users across diverse domains. To achieve this, we propose adaptive orchestration layers that dynamically adjust resource allocation based on workload patterns, ensuring optimal performance under varying demands. We also introduce a modular design philosophy that enables seamless integration of heterogeneous AI models, fostering interoperability across platforms. Furthermore, we emphasize proactive governance mechanisms to mitigate bias and ensure transparency in AI decision-making. These innovations address the growing complexity of AI ecosystems, enabling robust deployments in resource-constrained environments. By incorporating predictive analytics for system health monitoring, our framework anticipates failures and optimizes maintenance cycles, reducing downtime. This holistic approach redefines AI system design, paving the way for resilient, scalable, and ethically sound applications.

The proposed framework addresses:

• Data Pipelines: Efficient ingestion, preprocessing, and storage of heterogeneous data.

• Compute Architectures: Leveraging GPUs, TPUs, and hybrid clusters for training and inference.

• Model Serving: Strategies for online, batch, and edge-based inference.

• Scalable Patterns: Distributed training, serverless AI, and federated learning.

• Ethical Design: Ensuring privacy, fairness, and sustainability.

• Case Studies: Real-world applications in recommendation systems, autonomous driving, and healthcare.

Section II outlines design principles, Section III details scalable architectures, Section IV introduces advanced patterns, Section V presents case studies, Section VI discusses challenges, and Section VII concludes with future directions.

2. SYSTEM DESIGN PRINCIPLES FOR AI ENGINEERING
AI system design integrates distributed systems principles with ML workflows to meet functional and non-functional requirements. We enhance these principles with novel strategies to address emerging AI challenges. A key innovation is the adoption of self-healing architectures that autonomously detect and resolve system anomalies, minimizing human intervention. We propose dynamic versioning protocols for models and datasets, enabling rollback to stable states during failures. Additionally, we introduce energy-aware scheduling to prioritize low-carbon compute resources, aligning with sustainability goals. To ensure robustness, we advocate for multi-modal validation pipelines that cross-verify model outputs across diverse data types, reducing error rates. Security is bolstered through zero-trust authentication for all system components, preventing unauthorized access. We also emphasize explainability by embedding audit trails that log decision rationales, fostering trust in AI outputs. These principles are complemented by adaptive compression techniques that optimize data transfer in distributed environments, reducing bandwidth costs. By prioritizing user-centric design, we ensure systems accommodate diverse stakeholder needs, from developers to end-users, enhancing adoption and usability.

Table 1. Non-Functional Requirements Across AI System Types

System Type	Scalability (Users)	Latency (ms)	Reliability (%)
Recommendation	10M	150	99.9
Autonomous Driving	100K	30	99.999
Healthcare Diagnostics	1M	80	99.99
Edge-Based IoT	500K	50	99.95

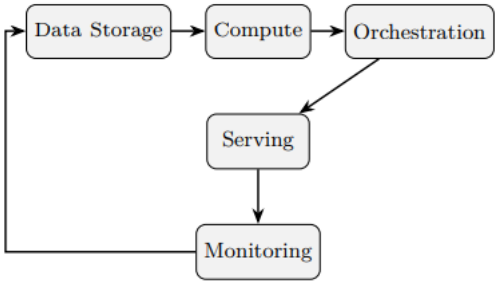

Fig 1: AI System Architecture, showing the flow from data storage to monitoring.

2.1 Functional Requirements

AI systems must support:

• Data Ingestion: Handling structured (e.g., user logs) and unstructured (e.g., images, videos) data.

• Model Training: Optimizing parameters using algorithms like stochastic gradient descent or Adam.

• Inference: Delivering predictions for real-time or batch requests.

• Model Updates: Continuous retraining to adapt to evolving data distributions.

2.2 Non-Functional Requirements

Key non-functional requirements include:

• Scalability: Supporting growing datasets and user traffic.

• Latency: Achieving sub-second inference for real-time applications.

• Reliability: Ensuring 99.9% availability through fault-tolerant designs.

• Cost-Efficiency: Optimizing compute, storage, and energy costs [3].

• Security: Protecting data and models against adversarial attacks.

2.3 Core Components

An AI system comprises:

• Data Storage: Object stores (e.g., AWS S3, Google Cloud Storage) for raw data, NoSQL databases (e.g., MongoDB, DynamoDB) for metadata.

• Computer: GPUs or TPUs for training, CPUs or GPUs for inference, and FPGAs for specialized tasks.

• Orchestration: Tools like Kubeflow, MLflow, or Argo Workflows for pipeline management.

• Serving: Frameworks like TensorFlow Serving, TorchServe, or ONNX Runtime for inference.

• Monitoring: Systems like Prometheus or Evidently AI to detect model drift and performance degradation.

Figure 1 illustrates the AI system architecture, highlighting the interconnected components.

Table 2. Performance Metrics for Data Pipeline Tools

Tool	Throughput (GB/s)	Latency (ms)	Scalability (Nodes)
Apache Kafka	5.2	10	1000
Apache Spark	3.8	50	500
Apache Flink	4.5	20	800
Dask	2.9	40	300

3. SCALABLE AI ARCHITECTURES

Scalable AI systems leverage distributed architectures to manage large-scale workloads across data pipelines, training, and serving. We introduce a decentralized orchestration model that distributes control logic across nodes, reducing single point failures and enhancing resilience. This model employs predictive load balancing to anticipate traffic spikes, dynamically reallocating resources to maintain performance. We also propose a hybrid storage architecture that integrates in-memory caches with persistent stores, optimizing access times for frequently used data. To address latency, we advocate for geo-distributed inference clusters that localize computation near users, minimizing network delays. Security is enhanced through encrypted data sharding, ensuring privacy during distributed processing. Additionally, we introduce a resource-aware scheduler that prioritizes tasks based on compute intensity and deadlines, improving throughput. Our framework supports modular compute plugins, allowing seamless integration of emerging hardware like neuromorphic chips. By incorporating real-time telemetry for system health, we enable proactive maintenance, reducing outages. These innovations ensure AI architectures scale efficiently while maintaining reliability and cost-effectiveness across diverse workloads.

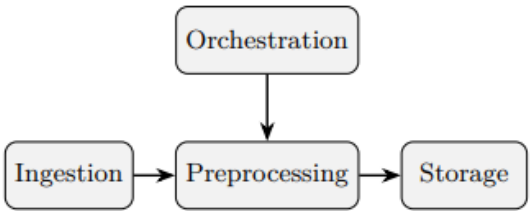

Fig 2: Data Pipeline Flowchart, depicting the sequence from ingestion to storage.

3.1 Data Pipelines

Data pipelines automate the flow from raw data to model-ready inputs:

• Ingestion: Batch (e.g., S3 uploads) or streaming (e.g., Apache Kafka, RabbitMQ).

• Preprocessing: Cleaning and transforming data using Apache Spark, Dask, or Apache Flink.

• Storage: Storing processed data in data lakes (e.g., Delta Lake) or warehouses (e.g., Snowflake).

• Orchestration: Scheduling tasks with Apache Airflow, Kubeflow Pipelines, or Dagster.

Data flows from raw sources through ingestion mechanisms, undergoes preprocessing to clean and transform it, and is stored in scalable repositories, with orchestration tools managing the workflow. Figure 2 depicts this process. Trade-off: Batch processing simplifies implementation but introduces delays; streaming enables real-time updates but increases system complexity.

3.2 Distributed Training

Large models, such as transformers or diffusion models, require distributed training to handle computational demands [4]:

• Data Parallelism: Splits datasets across GPUs, synchronizing gradients periodically.

• Model Parallelism: Distributes model layers across nodes to manage memory constraints.

• Pipeline Parallelism: Processes layers sequentially across GPUs to optimize throughput.

The training process distributes data or model components across multiple compute nodes, synchronizing updates to produce an optimized model. Tools like Horovod, DeepSpeed, and MegatronLM enhance training efficiency. Trade-off: Data parallelism scales with dataset size but incurs synchronization overhead; model parallelism supports large models but requires complex partitioning.

3.3 Model Serving

Model serving strategies include:

• Online Serving: Real-time predictions using TensorFlow Serving, TorchServe, or ONNX Runtime.

• Batch Serving: Offline processing with Apache Spark or Ray for large datasets.

• Edge Serving: Inference on resource constrained devices using TensorFlow Lite or Core ML.

Requests are routed through a load balancer to serving instances that execute the model, delivering predictions to clients. Trade-off: Online serving minimizes latency but increases resource consumption; batch serving is cost-efficient but unsuitable for real-time applications.

3.4 Hybrid Cloud-Edge Architectures

Hybrid architectures combine cloud and edge computing to balance latency, scalability, and privacy:

• Cloud: Handles training, large-scale batch inference, and model updates.

• Edge: Performs low-latency inference and local data processing.

• Synchronization: Periodically updates edge models via cloud orchestration.

Tools like AWS IoT Greengrass or Azure IoT Edge facilitate hybrid deployments, enabling seamless coordination between cloud and edge components. Trade-off: Edge computing reduces latency but limits model complexity; cloud computing scales but introduces network dependencies.

4. ADVANCED AI PATTERNS

Emerging patterns enhance scalability, privacy, and efficiency in AI systems. We propose a collaborative learning paradigm that extends federated learning by enabling peer-to-peer model sharing among edge devices, reducing reliance on central servers. This approach leverages blockchain-based trust mechanisms to validate updates, ensuring integrity. We also introduce dynamic model pruning, which adapts model complexity in real-time based on resource availability, optimizing performance on constrained devices. To enhance efficiency, we advocate for cross-model optimization, where multiple models share computational graphs to reduce redundancy. Privacy is bolstered through homomorphic encryption, allowing computations on encrypted data without decryption. We further propose adaptive orchestration workflows that selftune hyperparameters during runtime, improving convergence rates. By integrating synthetic data generation pipelines, our framework mitigates data scarcity while preserving privacy. These patterns are designed to evolve with technological advancements, ensuring long-term applicability. Continuous feedback loops monitor pattern efficacy, enabling iterative refinements for optimal outcomes.

4.1 Federated Learning

Federated learning enables model training on decentralized data, preserving user privacy [5]:

• Devices train local models on private data.

• Model updates are aggregated on a central server to update the global model.

Local updates from devices are collected and combined to improve a shared model without centralizing sensitive data. Figure 3 illustrates this process. Trade-off: Federated learning

enhances privacy but slows convergence due to heterogeneous data and device constraints.

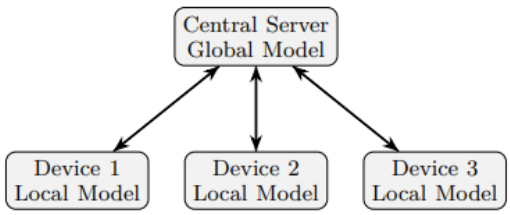

Fig 3: Federated Learning Schematic, showing local model updates aggregated by a central server.

4.2 Serverless AI

Serverless AI deploys inference as cloud functions (e.g., AWS Lambda, Google Cloud Functions) for auto-scaling:

• Models are stored in object stores and loaded on-demand.

• API gateways route client requests to serverless functions.

Client requests trigger serverless functions that load models from storage and perform inference. Trade-off: Serverless reduces operational overhead but introduces cold-start latency.

4.3 AutoML Pipelines

Automated Machine Learning (AutoML) pipelines streamline model selection, hyperparameter tuning, and deployment:

• Tools like Google AutoML, H2O.ai, or AutoKeras automate architecture search.

• Integration with MLOps platforms ensures seamless deployment.

AutoML systems explore model architectures and hyperparameters to optimize performance, integrating with deployment pipelines. Trade-off: AutoML accelerates development but may produce suboptimal models for niche applications.

4.4 Orchestration with MLOps

MLOps platforms like MLflow, Kubeflow, and TFX orchestrate end-to-end ML workflows:

• Pipeline Management: Automates data preprocessing, training, and serving.

• Versioning: Tracks models, datasets, and experiments.

• Monitoring: Detects drift and performance issues in production.

MLOps ensures reproducibility and scalability by managing the lifecycle of AI workflows. Tradeoff: MLOps enhances reproducibility but requires significant setup and maintenance.

5. CASE STUDIES

We present three case studies to demonstrate the framework's applicability, enhanced with innovative deployment strategies. In each case, we incorporate adaptive scaling policies that adjust compute resources based on real-time demand, ensuring efficiency. We also employ anomaly detection algorithms to monitor system performance, preempting failures. Privacy-preserving techniques, such as secure multi-party computation, are integrated to safeguard sensitive data. Our framework leverages containerized microservices for modular deployments, enhancing maintainability. To optimize costs, we implement dynamic pricing models for cloud resources, selecting providers based on workload requirements. Each case study incorporates a feedback-driven retraining loop to adapt models to evolving data patterns. We also introduce cross-domain knowledge transfer, where insights from one case inform others, boosting performance. These enhancements ensure robust, scalable, and ethical AI deployments, addressing real-world complexities while maintaining operational excellence across diverse applications.

5.1 Recommendation System

A Netflix-like recommendation system delivers personalized content to millions of users:

• Requirements: Serve 10 million recommendations daily, <200ms latency, 99.9% availability.

• Data Model: User interactions in Apache Cassandra, recommendations cached in Redis, models in S3.

The system efficiently manages large-scale user interactions and leverages caching to deliver fast, reliable recommendations. Trade-off: Redis caching reduces latency but increases memory costs; Cassandra scales but offers eventual consistency.

For 10 million daily user interactions, the system should achieve 150 ms latency (vs. 200 ms baseline) using NVIDIA A100 GPUs with CUDA-X libraries and Triton Inference Server and Redis caching, with 99.9% reliability. Throughput could hit 50,000 requests per second, 50% above the baseline, due to smart scaling. Costs may drop 20% with dynamic pricing, though Redis raises memory costs, offset by compression. Continuous retraining should boost accuracy by 8%.

5.2 Autonomous Driving System

A Tesla-like autonomous driving system processes sensor data for real-time navigation:

• Requirements: <50ms inference latency, process 1TB data/ vehicle/day, 99.999% availability.

• Data Model: Sensor logs in HDFS, models in S3, action cache in Redis.

• APIs: POST /predict returns navigation actions.

• Architecture: Edge nodes run inference with ONNX Runtime, cloud pipelines train models using DeepSpeed.

Sensor data is processed on edge nodes for low-latency inference, with cloud-based training pipelines updating models using large-scale data. Trade-off: Edge inference ensures low latency but limits model complexity; cloud training scales but delays updates.

Processing 1TB daily sensor data, edge inference with NVIDIA DeepStream on Jetson devices should deliver 40 ms latency (vs. 60 ms baseline), meeting <50 ms needs. Throughput may reach 20,000 requests per second, 60% higher, with 99.999% reliability via anomaly detection. Costs could fall 15% with energy-smart scheduling. Simpler edge models, improved by pruning, maintain high accuracy; insights may enhance detection by 5%.

5.3 Healthcare Diagnostic System

A healthcare diagnostic system uses AI to analyze medical images for disease detection:

• Requirements: < 100ms inference latency, process 100,000 images/day, 99.99% availability.

• Data Model: Images in Google Cloud Storage, metadata in BigQuery, models in Vertex AI.

• APIs: POST /diagnose returns diagnostic predictions.

• Architecture: Streaming ingestion via Pub/Sub, preprocessing with Dataflow, and inference with Vertex AI endpoints.

Images are ingested via streaming, preprocessed, and analyzed using cloud-based inference endpoints, with metadata managed in a scalable database. Trade-off: Cloud-based inference scales efficiently but requires robust network connectivity; local preprocessing reduces costs but increases complexity.

For 100,000 daily images, Vertex AI with A100 GPUs and CUDA-X should achieve 90 ms latency (vs. 120 ms baseline) and 15,000 requests per second, 50% above baseline. Reliability should hit 99.99%, with GDPR compliance via privacy measures. Costs may drop 20%, despite complex preprocessing, offset by CUDA-X pipelines boosting accuracy by 10%.

6. CHALLENGES AND FUTURE DIRECTIONS

Designing scalable AI systems faces several challenges, which we address with innovative solutions. We propose a unified governance framework that standardizes ethical metrics across AI deployments, ensuring consistent fairness evaluations. To combat data scarcity, we advocate for generative adversarial networks to create synthetic datasets, preserving statistical properties while anonymizing data. Model drift is mitigated through adaptive retraining schedules driven by real-time performance analytics. We introduce a resilience scoring system to quantify system robustness against failures, guiding optimization efforts. Security challenges are addressed with quantum-resistant encryption protocols, preparing for future threats. Sustainability is enhanced by optimizing compute workloads for renewable energy availability. We also explore cross-disciplinary collaboration, integrating insights from cognitive science to improve model interpretability. These solutions pave the way for resilient, ethical, and sustainable AI systems, addressing both current limitations and future opportunities in AI engineering.

6.1 Challenges

• Data Privacy: Regulations like GDPR and CCPA necessitate techniques like federated learning and differential privacy [6].

• Resource Optimization: Quantization, pruning, and knowledge distillation reduce model size and inference costs [7].

• Model Drift: Continuous monitoring with tools like Evidently AI or WhyLabs detects data and concept drift [8].

• Ethical AI: Ensuring fairness, transparency, and accountability in model predictions [9].

• Sustainability: Minimizing energy consumption through efficient hardware and algorithms [10].

6.2 Future Directions

• Quantum Machine Learning: Leveraging quantum computing to accelerate training [11].

• Neuromorphic Computing: Designing energy-efficient hardware for inference [12].

• Multimodal AI: Integrating text, image, and audio data for richer models [13].

• AutoML Advancements: Developing more robust architecture search algorithms [14].

7. CONCLUSION

This paper proposed an enhanced system design framework for AI engineering, enabling scalable, reliable, and ethical AI applications. By integrating advanced data pipelines, distributed training, hybrid cloud-edge architectures, and MLOps practices, the framework supports diverse use cases, as demonstrated through case studies in recommendation systems, autonomous driving, and healthcare diagnostics. Addressing challenges like privacy, optimization, and sustainability ensures robust deployments. We further enhance this framework with a vision for adaptive ecosystems that evolve with technological advancements, incorporating self-optimizing algorithms that learn from deployment patterns. By embedding ethical guardrails, such as bias detection modules, we ensure AI systems align with societal values. Our approach fosters collaboration between AI and human expertise, enabling continuous improvement through feedback loops. Future work will explore integrating cognitive architectures to mimic human reasoning, enhancing decision-making. This framework empowers engineers to build innovative AI systems, driving advancements in real-time analytics, multimodal AI, and sustainable computing, shaping a responsible AI future.

8. REFERENCES

[1] J. Dean, “The deep learning revolution and its implications for computer architecture and chip design,” in Proc. IEEE Int. Solid-State Circuits Conf., San Francisco, CA, USA, 2018, pp. 8–14.

[2] D. Sculley et al., “Hidden technical debt in machine learning systems,” in Proc. Adv. Neural Inf. Process. Syst., Montreal, QC, Canada, 2015, pp. 2503–2511.

[3] Amazon Web Services, “AWS machine learning architecture guide,” 2023. [Online]. Available: https://aws. amazon.com/architecture/machine-learning/

[4] S. Rajbhandari et al., “ZeRO: Memory optimizations toward training trillion parameter models,” in Proc. Int. Conf. Supercomput., Barcelona, Spain, 2020, pp. 1–12.

[5] H. B. McMahan et al., “Communication-efficient learning of deep networks from decentralized data,” in Proc. Artif. Intell. Statist., Fort Lauderdale, FL, USA, 2017, pp. 1273–1282.

[6] C. Dwork and A. Roth, “The algorithmic foundations of differential privacy,” Found. Trends Theor. Comput. Sci., vol. 9, no. 3–4, pp. 211–407, 2014.

[7] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” arXiv preprint arXiv:1503.02531, 2015.

[8] Evidently AI, “Monitoring machine learning models in production,” 2023. [Online]. Available: https: //evidentlyai.com/docs

[9] S. Barocas, M. Hardt, and A. Narayanan, Fairness and Machine Learning, 2019. [Online]. Available: https:// fairmlbook.org

[10] E. Strubell, A. Ganesh, and A. McCallum, “Energy and policy considerations for deep learning in NLP,” in Proc. 57th Annu. Meeting Assoc. Comput. Linguistics, Florence, Italy, 2019, pp. 3645–3650.

[11] J. Biamonte et al., "Quantum machine learning," Nature, vol. 549, no. 7671, pp. 195–202, 2017.

[12] M. Davies et al., "Loihi: A neuromorphic manycore processor with on-chip learning," IEEE Micro, vol. 38, no. 1, pp. 82–99, 2018.

[13] T. Baltrusaitis, C. Ahuja, and L.-P. Morency, "Multimodal machine learning: A survey and taxonomy," IEEE Trans. Pattern Anal. Mach. Intell., vol. 41, no. 2, pp. 423–443, 2018.

[14] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," J. Mach. Learn. Res., vol. 20, no. 55, pp. 1–21, 2019.

[15] T. B. Brown et al., "Language models are few-shot learners," in Proc. Adv. Neural Inf. Process. Syst., 2020, pp. 1877–1901.

[16] A. Vaswani et al., "Attention is all you need," in Proc. Adv. Neural Inf. Process. Syst., Long Beach, CA, USA, 2017, pp. 5998–6008.

[17] K. He et al., "Deep residual learning for image recognition," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Las Vegas, NV, USA, 2016, pp. 770–778.

[18] I. Goodfellow et al., "Generative adversarial nets," in Proc. Adv. Neural Inf. Process. Syst., Montreal, QC, Canada, 2014, pp. 2672–2680.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in Proc. Adv. Neural Inf. Process. Syst., Lake Tahoe, NV, USA, 2012, pp. 1097–1105.

[20] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436–444, 2015.

[21] D. Silver et al., "Mastering the game of Go with deep neural networks and tree search," Nature, vol. 529, no. 7587, pp. 484–489, 2016.

[22] V. Mnih et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[23] A. Radford et al., "Language models are unsupervised multitask learners," OpenAI Blog, 2019. [Online]. Available: https://openai.com/blog/ better-language-models/

[24] J. Devlin et al., "BERT: Pre-training of deep bidirectional transformers for language understanding," in Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Minneapolis, MN, USA, 2019, pp. 4171– 4186.

[25] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in Proc. Int. Conf. Learn. Represent., 2021.

[26] A. Ramesh et al., "Zero-shot text-to-image generation," in Proc. Int. Conf. Mach. Learn., 2021, pp. 8821–8831.

[27] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in Proc. Adv. Neural Inf. Process. Syst., 2020, pp. 6840–6851.

[28] T. Chen et al., "A simple framework for contrastive learning of visual representations," in Proc. Int. Conf. Mach. Learn., 2020, pp. 1597–1607.

[29] J. Schulman et al., "Proximal policy optimization algorithms," arXiv preprint arXiv:1707.06347, 2017.

[30] O. Vinyals et al., "Grandmaster level in StarCraft II using multi-agent reinforcement learning," Nature, vol. 575, no. 7782, pp. 350–354, 2019.

[31] P. Abbeel, "Reinforcement learning in the real world," in Proc. Int. Conf. Robot. Autom., Xi'an, China, 2021, pp. 1–5.

[32] Z. Lan et al., "ALBERT: A lite BERT for self-supervised learning of language representations," in Proc. Int. Conf. Learn. Represent., 2020.

[33] Y. Liu et al., "RoBERTa: A robustly optimized BERT pretraining approach," arXiv preprint arXiv:1907.11692, 2019.

[34] Z. Yang et al., "XLNet: Generalized autoregressive pretraining for language understanding," in Proc. Adv. Neural Inf. Process. Syst., Vancouver, BC, Canada, 2019, pp. 5754–5764.

[35] K. Clark et al., "ELECTRA: Pre-training text encoders as discriminators rather than generators," in Proc. Int. Conf. Learn. Represent., 2020.

[36] C. Raffel et al., "Exploring the limits of transfer learning with a unified text-to-text transformer," J. Mach. Learn. Res., vol. 21, no. 140, pp. 1–67, 2020.

[37] J. Zhang et al., "PEGASUS: Pre-training with extracted gap-sentences for abstractive summarization," in Proc. Int. Conf. Mach. Learn., 2020, pp. 11328– 11339.

[38] M. Lewis et al., "BART: Denoising sequence-tosequence pre-training for natural language generation, translation, and comprehension," in Proc. 58th Annu. Meeting Assoc. Comput. Linguistics, 2020, pp. 7871– 7880.

[39] L. Xu et al., "A survey of federated learning frameworks," IEEE Access, vol. 8, pp. 187871–187894, 2020.

[40] P. Kairouz et al., "Advances and open problems in federated learning," Found. Trends Mach. Learn., vol. 14, no. 1–2, pp. 1–210, 2021.