State of the Art for Online ML Pipelines (2026)

This research outlines the **State of the Art (SOTA)** for online ML pipelines in 2026, structured to align with your design for **Constellation**.

By 2026, the industry has shifted from **"models as APIs"** to **"Compound AI Systems"**—where inference is a complex DAG involving retrieval (RAG), guardrails, and real-time feature engineering.

---

1. Industry State of the Art (2026)

The primary goal of modern inference pipelines is eliminating **Training-Serving Skew** while maintaining **sub-100ms latency**.

Trend	SOTA Requirement
Compound Systems	Moving beyond a single model to a DAG of multiple models (e.g., `Router -> Embedder -> Ranker`).
Real-time Enrichment	Fetching features from an **Online Store** (Redis/Aerospike) using a `userId` during the request flow.
Batch-Inference Hybrid	Handling single requests and `Candidates<T>` (mini-batches) through the same logic to optimize GPU throughput.
Guardrails & Alignment	Real-time validation of inputs/outputs for safety and schema compliance (Pydantic-like validation).

---

2. Constellation-Lang Level: Tools for Data Scientists

Goal: High-level DSL abstractions for common *Inference Path* operations.

Data scientists need to manipulate the data flow without worrying about memory management or JVM types.

A. Feature Lookup (Online Store Integration)

The DSL should support a native **Enrichment** operator to pull data from external feature stores.

```
# DSL Concept
in userId: Int
userFeatures = lookup("user-profile-store", userId)
# Automatically merges into the pipeline context
```

B. Logical Guardrails (Predicates)

Tools to *drop* or *redirect* candidates based on business logic before they reach expensive models.

```
# DSL Concept
valid_candidates = communications.filter(c => c.content.length > 0)
```

C. Standard Transformers

Out-of-the-box functions for common inference-time tasks:

- **Scaling/Normalization:** `z-score(value, mean, std)` (constants from training).
- **Categorical Encoding:** `one-hot(category, vocabulary_list)`.
- **Embedding Projection:** `project(vector, matrix)` for dimensionality reduction.

---

3. Scala Module Level: Tools for ML Engineers

Goal: Lower-level control over performance, IO, and type-safe data movement.

Engineers must bridge the gap between the Constellation IR and the high-performance JVM ecosystem.

A. Zero-Copy Record Merging

Since `+` (merge) is a core Constellation-Lang feature, the Scala module must implement this using **structural sharing** or **specialized value classes** to avoid excessive object allocation during high-frequency inference.

B. Specialized `Candidates<T>` Runtimes

Industry SOTA includes **Adaptive Batching**. The Scala module should provide:

- **Batcher Module:** Collects individual requests into `Candidates<T>` up to a `max_latency` or `max_batch_size`.
- **DJL / ONNX Integration:** Scala wrappers for **Deep Java Library (DJL)** or **ONNX Runtime** to execute models in the DAG.

C. Observability & Lineage

Engineers need visibility into why predictions occur.

- **Traceable Records:** Attach a `traceId` to every record in `Candidates<T>`.
- **Schema Validation:** Strict record validation at DAG boundaries to ensure incoming JSON matches the Constellation-Lang type definition.

---

4. Prioritizing Work Opportunities

Based on current industry gaps, here is the suggested roadmap for **Constellation**:

Phase 1: The Type-Safe Core *(Priority: High)*

- **Scala:** Implement the high-performance type algebra (`merge`, `project`) to ensure the DSL's `+` operator does not become a bottleneck.
- **DSL:** Finalize `Candidates<T>` element-wise operation semantics.

Phase 2: The Enrichment Bridge *(Priority: Medium)*

- **DSL:** Add a `lookup` keyword for online feature stores.
- **Scala:** Create a `FeatureStoreConnector` trait that engineers can implement for Redis, Feast, or DynamoDB.

Phase 3: Operational Guardrails *(Priority: Medium)*

- **DSL:** Implement `validate` and `filter` blocks.
- **Scala:** Add automatic metrics collection (latency per node, batch size distribution).