

DOI: <https://doi.org/10.15276/ict.03.2026.44>

UDC 004.415.2:004.75

Architectural and engineering aspects of integrating cloud artificial intelligence services into an e-commerce web application

Khrystyian M. Tyskyi

ORCID: <https://orcid.org/0009-0005-7830-3938>; kristiantyskyi@gmail.com

Lutsk National Technical University, 75, Lvivska Str. Lutsk, 43018, Ukraine

ABSTRACT

Modern electronic commerce increasingly relies on artificial-intelligence capabilities such as visual search and product recommendation, yet for most mid-sized enterprises building and hosting the complete intelligence stack in-house remains impractical. A pragmatic alternative is to compose the required capability from managed cloud services; the engineering cost of such composition, however, is concentrated not in the models themselves but in the integration layer of the application. The paper investigates the architectural and engineering aspects of integrating five heterogeneous cloud services — a managed vector search index, a serverless inference platform, an object storage, a public model hub and a managed relational database — into a production web application for image-based dress search operated by a wedding fashion manufacturer whose catalog contains about twenty-nine thousand items and whose retail network spans more than three hundred sixty stores in seventy-five countries. A three-tier reference architecture is proposed in which every external service is isolated behind a dedicated adapter, authentication is encapsulated in a dedicated facade, and interchangeable embedding providers are implemented with the Strategy design pattern, enabling replacement of the vision-language model without modification of the business logic. The engineering decisions are formalized with analytical models: the availability of the synchronous composition of services, the success probability and worst-case latency overhead of a bounded retry scheme, and a break-even model for choosing between local inference and a per-call cloud interface. Experimental evaluation of the production deployment shows that external interactions dominate the request lifecycle: background removal through the serverless platform accounts for about three-fifths of the mean end-to-end response time of nearly four seconds, whereas the managed vector search contributes only about seventy milliseconds, or less than two percent. The local deployment of the domain-specific encoder surpasses the general-purpose cloud embedding interface in both retrieval quality and latency at the cost of several hundred megabytes of resident memory. During a four-week industrial trial, forty-five consultants executed more than eighteen hundred queries, and more than four-fifths of the queries received positive feedback, which confirms the practical viability of the proposed integration architecture.

Keywords: Cloud artificial intelligence services; software architecture; systems integration; e-commerce; visual search; software engineering

For citation: Tyskyi Kh. M. “Architectural and engineering aspects of integrating cloud artificial intelligence services into an e-commerce web application”. *Informatics. Culture. Engineering*. 2026; Vol 3 № 2 (4): 588–598. DOI: <https://doi.org/10.15276/ict.03.2026.44>

Relevance. Artificial intelligence has become a differentiating capability of electronic commerce. Visual search, in which the query is a photograph rather than a text string, removes the semantic gap between what a customer sees and what a catalog taxonomy can express, and large marketplaces have made such functionality an expected part of the shopping experience. For a mid-sized manufacturer, however, reproducing the in-house platforms of technology corporations is unrealistic: dedicated research teams, GPU clusters and custom serving infrastructure lie far outside a typical engineering budget. A pragmatic alternative is offered by managed cloud services: vector databases offered as a service, serverless inference platforms, public model hubs and managed storage and database offerings. Composing a production feature out of such services shifts the center of gravity of the engineering effort from model development to systems integration — an observation consistent with the argument that machine-learning code constitutes only a small fraction of a real system, while the surrounding infrastructure dominates the total cost of ownership [11], and with surveys of deployment practice that identify integration and operation as the main obstacles between a trained model and a working product [12].

Integrating several independently operated cloud services into one interactive feature raises questions that belong to software architecture rather than machine learning: how to isolate heterogeneous authentication schemes and interface styles behind stable internal contracts; how to keep the latency of a multi-service call chain within an interactive budget; how to remain operational when an external dependency degrades; how to prevent silent behavioral changes when a provider updates a model; and how to retain the ability to replace a provider without rewriting business logic. Although these concerns are individually addressed by classical works on design

patterns [15], microservice architecture [16] and production stability [17], their joint application to the composition of cloud artificial-intelligence services in e-commerce has received little systematic attention.

The paper studies these questions using a production system as the research material: Milla AI Assistant, a web application for image-based wedding dress search developed for the manufacturer Milla Nova, whose catalog contains approximately 29,000 dress models and whose retail network spans 366 stores in 75 countries. The system composes five external cloud services within a single request path bounded by a five-second response-time requirement and has passed a four-week industrial trial in retail stores of five countries. The contribution of the paper is threefold: a reference integration architecture that isolates every cloud service behind a dedicated adapter with centralized authentication and configuration; a set of analytical models that make the reliability, latency and cost consequences of integration decisions explicit; and a quantitative evaluation of the resulting system in production, including a comparison of local and cloud deployment of the embedding model.

Literature review and problem statement. The algorithmic foundation of visual product search is formed by neural image representations. The Vision Transformer treats an image as a sequence of patches and produces a dense embedding vector of fixed dimensionality [1]; contrastive vision-language pre-training aligns image and text embeddings in a shared space [2], and SigLIP improves it by replacing the softmax-based contrastive objective with a pairwise sigmoid loss [3]. For narrow domains, specialized derivatives outperform general-purpose encoders: in the fashion domain, the publicly distributed Marqo FashionSigLIP model is fine-tuned on apparel data and is available through the Hugging Face Hub [4]. Similarity search over embedding collections is served by the Hierarchical Navigable Small World graph algorithm, which provides approximate nearest-neighbor search with logarithmic complexity and high recall [5], and by libraries such as FAISS that extend such search to billion-scale collections [6]; a recent survey of vector database management systems catalogs the rapidly growing landscape of dedicated engines and managed services [14], and commercial offerings such as Google Vertex AI Vector Search package the HNSW index as a fully managed service.

Industrial visual search systems have been described by the engineering teams of eBay [7], Pinterest [8] and Alibaba [9]. These reports confirm the practical value of embedding-based retrieval at the scale of hundreds of millions of images, but they document in-house platforms operated by large dedicated teams; the architectural problem of assembling a comparable capability from rented cloud services, under the constraints of a mid-sized enterprise, is left outside their scope. Salient-object segmentation models such as U²-Net [10] are relevant as a preprocessing step that removes background noise from real-world query photographs. From the software engineering side, the integration of external services rests on the catalog of object-oriented design patterns [15], including Strategy, Adapter and Facade; the microservice architectural style [16]; stability patterns for distributed systems – timeouts, bounded retries, fail-fast validation and bulkheads [17]; and the OAuth 2.0 authorization framework that governs access to most commercial cloud interfaces [19]. Serverless computing, which underlies pay-per-call inference platforms, has been analyzed together with its economic and operational trade-offs in [18]; engineering-oriented studies of machine-learning systems consistently emphasize that integration and operation, not modeling, dominate practical effort [11], [12], [13].

Two trade-offs recur across this literature. The first is managed versus self-hosted operation of the vector index: a self-hosted engine offers full control over parameters and cost, but transfers to the application team the burden of memory provisioning, replication and index rebuilds, which the survey [14] identifies as the dominant operational complexity of vector search; a managed service converts this burden into an interface call at the price of per-query fees and provider dependence. The second is the economics of serverless inference: pay-per-call platforms remove idle-capacity costs and scale transparently, but introduce cold-start latency variance and rate limits that must be absorbed by the calling application [18]. Both trade-offs are resolved quantitatively for the studied system in the results section.

In the author's previous work, the ranking logic of the studied system [20] and a three-level methodology for evaluating the quality of visual recommendations [21] were examined; those publications deliberately leave the integration architecture out of consideration. The review thus reveals a gap: the literature offers mature building blocks — embedding models, managed vector search, serverless inference, integration patterns — but no documented reference architecture that composes several heterogeneous cloud services within one latency-bounded interactive request path of an e-commerce application operated by a mid-sized enterprise. Open questions include the division of responsibilities between local computation and rented services; the encapsulation of provider-specific authentication, configuration and versioning; the containment of partial failures of external dependencies; and the quantitative assessment of what each integrated service contributes to the end-to-end latency, quality and cost of the feature. These questions define the problem addressed in this study.

The aim and objectives of the research. The aim of the research is to develop and experimentally substantiate an integration architecture that composes heterogeneous cloud artificial-intelligence services into a production e-commerce web application under an interactive latency budget.

To achieve this aim, four objectives are addressed:

- 1) to design a reference architecture that isolates each external service behind a dedicated adapter with centralized authentication, configuration and version management;
- 2) to formalize the reliability, latency and cost implications of the integration decisions with analytical models;
- 3) to measure the contribution of every external interaction to the end-to-end behavior of the deployed system;
- 4) to compare local and cloud deployment options for the embedding model and validate the architecture in industrial operation.

General organization of the system. Milla AI Assistant follows a three-tier architecture with a strict separation between the presentation, business-logic and data layers. The presentation tier is an Angular single-page application compiled to a static bundle and served through a content delivery network; it captures or uploads a photograph and displays ranked catalog results with relevance percentages. The business tier is a stateless NestJS (Node.js) server that orchestrates the entire processing of a search request. The data tier combines three PostgreSQL databases (product catalog, materials, and analytics) with a managed vector index. Five external cloud services are attached to this core, each with its own protocol, authentication scheme and role (Fig. 1 and Table 1).

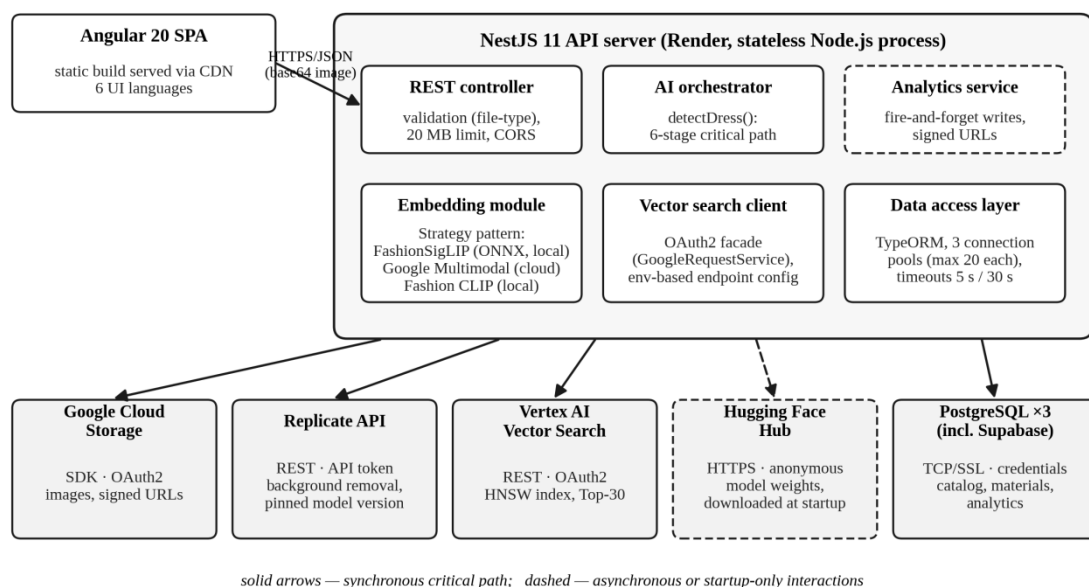


Fig. 1. Integration architecture of the Milla AI Assistant system

Source: compiled by the author

Table 1. External cloud services integrated into the system

Service	Protocol / authentication	Role in the system	Interaction mode	Pinning / limits
Google Vertex AI Vector Search	REST / OAuth2 (service account)	ANN search over 768-D embeddings, Top-30 neighbors	synchronous, critical	endpoint and index ID in env vars
Replicate	REST / API token	background removal (851-labs/background-remover)	synchronous, critical, retry ≤ 3	model version pinned
Google Cloud Storage	SDK / OAuth2	storage of query images; signed URLs	store: sync; URL signing: async	bucket name in env vars
Hugging Face Hub	HTTPS / anonymous	distribution of FashionSigLIP weights	startup only, cached in RAM	model revision fixed
Supabase PostgreSQL	TCP/SSL / credentials	query logging, user feedback	asynchronous, non-critical	pool ≤ 20 ; timeouts 5 s / 30 s

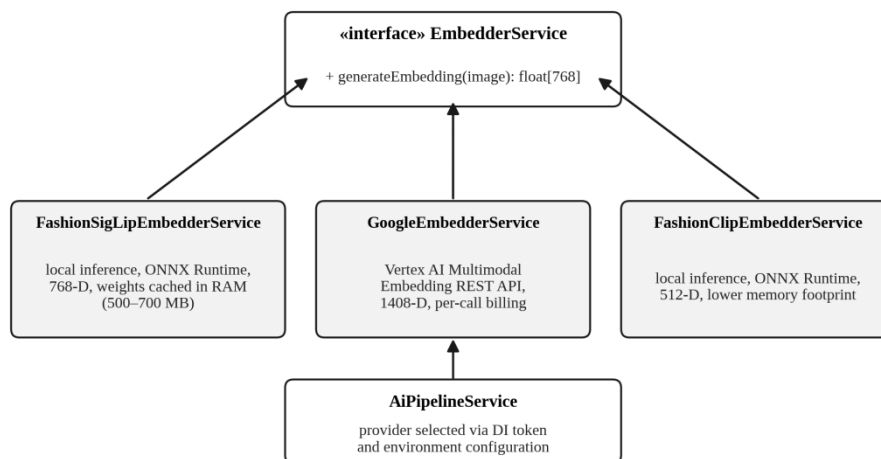
Source: compiled by the author

Isolation and integration patterns. Every external service is accessed exclusively through a dedicated adapter service registered in the dependency-injection container. The adapter owns the wire format, the error taxonomy and the credentials of its service, so a provider change is confined to a single class, and every provider-specific detail – endpoint, credential, wire format, failure taxonomy – occurs in the code base exactly once. Authentication with Google Cloud is centralized in an OAuth2 facade that obtains and refreshes service-account tokens and signs outgoing requests for both the vector index and the object storage; business services never handle raw credentials. Incoming requests are validated at the system boundary in a fail-fast manner: the image format is detected from binary signatures rather than from the declared MIME type, HEIC and WebP inputs are converted to JPEG, and the request body is limited to 20 MB, so malformed traffic is rejected before any paid external call is made.

The embedding subsystem is built around the Strategy pattern (Fig. 2). An EmbedderService interface defines a single contract – the transformation of an image into a numeric vector

$$v = f_{\theta}(I) \in \mathbb{R}^d, \quad (1)$$

where f_{θ} is the encoder with parameters θ and d is the embedding dimensionality – and three interchangeable implementations provide it: local ONNX Runtime inference of Marqo FashionSigLIP ($d = 768$), the cloud Google Multimodal Embedding API ($d = 1408$) and local Fashion CLIP ($d = 512$). The active provider is selected by an environment variable at startup; switching neither touches the orchestration logic nor requires recompilation, which turns a vendor decision into a configuration decision.

**Fig. 2. Interchangeable embedding providers implemented with the Strategy pattern**

Source: compiled by the author

Reliability model of the synchronous composition. A request succeeds only if every service on its synchronous critical path succeeds. Assuming independent failures, the availability of a serial composition of m services is the product

$$A_{path} = A_1 \cdot A_2 \cdot \dots \cdot A_m, \quad (2)$$

which decreases monotonically with every added dependency: four external calls of 99.5 % availability each already bound the path at about 98 %. This multiplicative penalty motivates the first architectural rule of the system: an external interaction is placed on the critical path only if the response cannot be formed without it. Analytics writes and signed-URL generation are therefore executed in a fire-and-forget manner: their failures are logged but never propagate to the user.

For the unstable serverless dependency (background removal), a bounded retry scheme with a fixed delay is applied. If a single attempt succeeds with probability a , the probability of success within n attempts is

$$P_n = 1 - (1 - a)^n, \quad (3)$$

so even a degraded service with $a = 0.9$ yields $P_3 = 0.999$. The price is a bounded worst-case latency overhead

$$T_{add}^{max} = (n - 1) \cdot (t_o + d), \quad (4)$$

where t_o is the per-attempt timeout and d is the inter-attempt delay; with the production configuration $n = 3$ and $d = 3$ s the scheme remains within the interactive tolerance of the application.

The expected latency overhead is far smaller than the bound (4), since only the fraction of calls that fail on the first attempt pays the retry cost:

$$E[T] = \bar{t} + (1 - a) \cdot (t_o + d), \quad (5)$$

where $\bar{t}$ is the mean single-attempt duration; for availability close to unity the expected overhead is negligible, although the worst case (4) must still be budgeted when the response-time requirement is stated for the 95th percentile. Version pinning complements retries: the exact revision of the external model is fixed in configuration, which excludes silent behavioral drift when the provider updates the default model.

Local versus cloud inference. The embedding step admits two deployment forms with different cost structures. A per-call cloud interface converts every query into marginal cost, whereas local inference converts the same workload into a fixed infrastructure cost – chiefly the 500–700 MB of resident memory occupied by the FashionSigLIP weights and the ONNX Runtime session that are loaded once at application startup and cached for the lifetime of the process.

Denoting by c the provider fee per call and by C the monthly increment of infrastructure cost attributable to local hosting, the break-even monthly query volume is

$$N^* = \frac{C}{c}; \quad (6)$$

above N^* local inference is strictly cheaper, and it additionally removes a network round trip and an external failure mode from the critical path. This model, together with the quality comparison reported below, justified the choice of local inference for the production system, while the Strategy pattern keeps the cloud option available as a fallback configuration.

Index maintenance workflow. The maintenance of the vector index as the catalog evolves and as the embedding model changes is deliberately decoupled from the interactive path. A dedicated command-line utility walks the catalog databases, generates an embedding for every product image through the same EmbedderService abstraction that serves user queries, and emits a JSONL file in the import format of the vector service; the file is then imported into the index as an offline batch operation. Because the identifier of the deployed index is a configuration value, a re-indexing triggered by a model replacement follows a blue-green scheme: the new index is built and validated next to the old one, and the switch is a single environment-variable change with instant rollback.

Treating the index as an immutable, versioned artifact produced by a reproducible batch job removes the hardest consistency problems of vector-database integration from the runtime entirely.

Security of the integration layer. Attaching five external services multiplies the attack and leakage surface of the application, so the integration layer enforces a uniform security discipline. Credentials never appear in the repository or in the build artifact: all tokens and service-account keys are injected through environment variables, each service uses its own credential, and the Google service account is restricted to the minimal scopes required by the vector index and the storage bucket, so a compromise of one integration does not grant access to the others. All channels are encrypted, and the OAuth2 token lifecycle is confined to the facade, so access tokens never leak into business code or logs. On the inbound side, the server accepts requests only from the front-end origins enumerated in the CORS whitelist, limits the request body to 20 MB and validates the image format against binary signatures before any processing; query images are never exposed through public bucket permissions – a time-limited signed URL is generated instead. All environment-specific values are externalized to environment variables, so the same stateless build artifact runs unchanged in staging and production.

Measurement methodology. All measurements were performed on the production deployment of the system – the stateless server process with a warm model cache, connected to the live vector index containing embeddings of the full catalog of about 29,000 dress models. Latency was profiled over a sample of 100 search requests, recording the mean and the 95th percentile (P95) for every interaction of the request lifecycle. Retrieval quality was evaluated on a labeled set of 200 query images whose composition models the real usage scenarios of the system: 60 official catalog images, 80 real consultant photographs, 30 competitor brand images and 30 synthetic modifications testing robustness to rotation, cropping and brightness changes. The ground truth was defined at the level of the product rather than the individual image, and the standard information-retrieval metrics Precision@K, Hit Rate@K and mean average precision (MAP@30) were computed for each embedding provider under an otherwise identical pipeline [21].

Measured characteristics of external interactions. Fig. 3 shows the sequence of interactions of the server with the external services while processing one search request, and Table 2 summarizes the latency measurements.

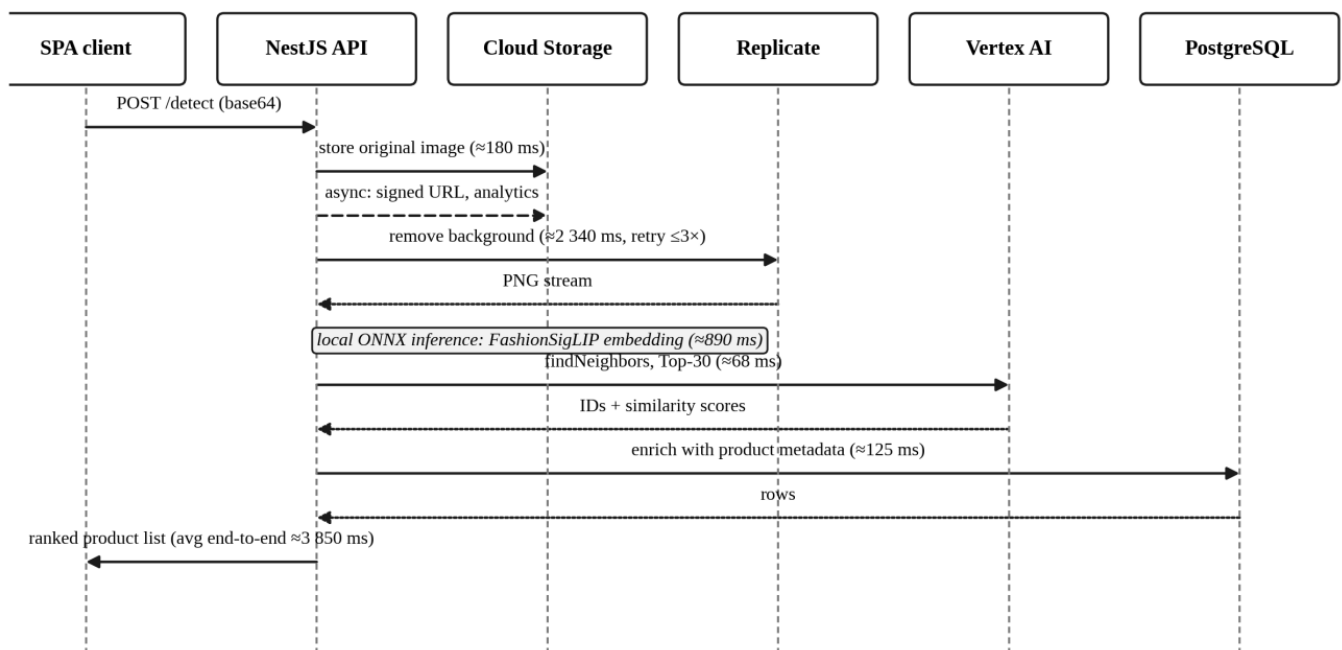


Fig. 3. Interactions of the API server with external services while processing one search request

Source: compiled by the author

Table 2. Measured characteristics of external interactions (100 production requests)

Interaction	Mean, ms	P95, ms	Share of time, %	Failure handling
Google Cloud Storage: store query image	180	310	4.7	non-blocking side effects isolated
Replicate: background removal	2 340	3 870	60.8	retry $\leq 3 \times 3$ s; pinned version
Vertex AI Vector Search: Top-30 query	68	95	1.8	OAuth2 facade; env-switchable index
PostgreSQL: metadata enrichment	125	210	3.2	pooling; connect/idle timeouts
Local processing (validation, embedding, serialization)	1 137	—	29.5	in-process; warm model cache
End-to-end request	3 850	5 420	100.0	—

Source: compiled by the author

The measurements confirm the architectural expectations. The managed vector search — the only service on the critical path that is queried at catalog scale — costs a mere 68 ms on average, demonstrating that a rented HNSW index imposes no practical latency penalty. The dominant contributor is the serverless background-removal call, which accounts for 60.8 % of the mean end-to-end time of 3,850 ms; its P95 of 3,870 ms is also the main driver of the end-to-end P95 of 5,420 ms, which slightly exceeds the five-second budget in the worst tail. The asynchronous side operations contribute nothing to the response time, validating the fire-and-forget separation, and during the measurement campaign the bounded retry scheme absorbed transient failures of the serverless platform without a single user-visible error.

Comparison of embedding providers. The Strategy-based embedding module makes provider comparison an inexpensive experiment: each provider was activated by configuration, a separate vector index was built with the same catalog images, and the labeled query set was executed against each index. Table 3 combines the retrieval quality metrics with the operational characteristics of the three providers.

Table 3. Embedding providers: retrieval quality and operational profile

Provider	Deployment	Dim.	P@1	P@5	MAP@30	Cost profile
Marqo FashionSigLIP	local, ONNX Runtime	768	0.875	0.794	0.762	fixed: 500–700 MB RAM
Google Multimodal Embedding	cloud API (Vertex AI)	1408	0.810	0.738	0.698	per-call billing + RTT
Fashion CLIP	local, ONNX Runtime	512	0.745	0.672	0.621	fixed: lower RAM

Source: compiled by the author

The domain-specific local model dominates on every quality metric: Precision@1 of 0.875 against 0.810 for the general-purpose cloud encoder and 0.745 for the lighter local alternative. The result carries an integration lesson: the highest-quality option in this case is also the one that removes an external dependency from the critical path, and the architecture made this comparison – and any future revision of it – a configuration-level experiment rather than a redevelopment effort.

Industrial validation. The integrated system passed a four-week industrial trial in 12 retail stores across five countries. Forty-five consultants executed 1,847 search queries through the production deployment; 83.1 % of the queries received a positive rating through the built-in feedback mechanism, 8.9 % a negative one, and 42 of the 45 consultants assessed the system as useful for daily work. Throughout the trial the degradations of non-critical services remained invisible to users, and no incident required manual intervention in the integration layer. The analysis of the negatively rated queries showed that the dominant causes were queries for styles absent from the catalog, images of very low quality or extreme zoom, and searches for a specific detail rather

than an overall silhouette; none of the negative cases was attributable to a failure or timeout of an integrated service, which indicates that the residual quality risk of the feature lies in the model and data domain rather than in the integration layer [21].

Discussion of the results. The measurements substantiate an integration doctrine that can be transferred to other mid-scale e-commerce systems. Renting the most infrastructure-heavy component – the ANN index – is essentially free in latency terms (1.8 % of the request time) and eliminates the operational burden of index hosting, whereas renting a per-request GPU computation (background removal) is the principal latency and availability liability; when a cloud service sits on an interactive critical path, its latency profile matters more than its accuracy alone. The local-versus-cloud decision for the embedding model is not a dichotomy but a configuration governed by the break-even volume (6) and by quality: in this system the domain-specific local model wins on both axes, but the Strategy pattern preserves the cloud option for scenarios with low traffic or constrained memory. The multiplicative availability model (2) provides a simple design discipline — every candidate integration must justify its presence on the synchronous path — which in practice moved two of the five services entirely off that path.

Several limitations should be acknowledged. The latency figures were collected from one geographic deployment over a bounded observation window, so absolute values will differ in other configurations, although the structural conclusions are expected to transfer. The quality comparison relies on a 200-query labeled set constructed with the participation of domain experts, and the study rests on a single industrial case; the analytical models deliberately use independence assumptions and worst-case bounds rather than full queueing analysis. The retry scheme uses a fixed delay and no circuit breaker, so stability patterns such as circuit breakers and bulkheads [17] are a natural next step. The identified bottleneck also defines the optimization route: replacing the external background-removal call with a local segmentation model of the U²-Net family [10] is projected to reduce the mean end-to-end time to about 1.5 s while removing the least available dependency; the integration architecture localizes this change to one adapter.

It is instructive to contrast the resulting composition with the in-house platforms documented by eBay, Pinterest and Alibaba [7], [8], [9]. Those systems achieve lower absolute latency by owning every component, but at the price of dedicated infrastructure teams that a mid-sized enterprise cannot sustain. The architecture presented here reaches interactive latency and production-grade quality with a two-component footprint — a statically served front end and one stateless server process — by renting exactly the components whose self-hosting would dominate operational cost, and by keeping the exit door open: the embedding weights are consumed in the standard ONNX format, the index is populated from a portable JSONL export, and every provider binding is a configuration value. The lock-in risk of building on rented services is thereby reduced to the cost of rewriting single adapters.

Conclusions. The paper has examined the integration of heterogeneous cloud artificial-intelligence services into a production e-commerce web application from the standpoint of software architecture and engineering, using the Milla AI Assistant visual search system as the research material. The following results were obtained.

1) A reference integration architecture has been proposed in which five cloud services are attached to a stateless three-tier core through dedicated adapters, authentication is concentrated in an OAuth2 facade, non-critical interactions are moved off the synchronous path, and interchangeable embedding providers are realized with the Strategy pattern, so that provider selection becomes a configuration decision.

2) Analytical models of the integration decisions — the availability of a serial service composition, the success probability and worst-case latency overhead of bounded retries, and the break-even volume between per-call cloud billing and fixed-cost local inference — have been formulated and applied to justify the production configuration.

3) Production measurements have quantified the contribution of every external interaction: the managed HNSW vector search adds only 68 ms (1.8 %) to the mean 3,850 ms request, whereas

serverless background removal dominates with 60.8 % and defines the P95 tail of 5,420 ms; the local domain-specific FashionSigLIP encoder surpasses the cloud embedding interface in quality (Precision@1 of 0.875 versus 0.810) while eliminating an external dependency at the cost of 500–700 MB of memory.

4) A four-week industrial trial (45 consultants, 1,847 queries, 83.1 % positive ratings) has confirmed the operational viability of the architecture, with no user-visible incidents caused by the integrated services.

5) The proposed architecture is transferable beyond the studied case: any interactive e-commerce feature composed of rented artificial-intelligence capabilities faces the same set of decisions – critical-path composition, retry budgets, provider interchangeability, index maintenance, secrets management – and the presented models make these decisions explicit and checkable. Future work includes replacing the dominant external dependency with a local segmentation model, extending the integration layer with circuit-breaker and bulkhead patterns, and continuous monitoring of model drift in operation.

Conflict of interests: the author declares that he has no conflict of interest, including financial, personal, authorship or any other, that could affect the research and the results published in this article

Funding: the research was conducted without financial support

Data availability: all data are available in digital or graphical form in the main text of the manuscript.

Use of artificial intelligence: during the preparation of the manuscript the author used generative artificial-intelligence tools for language editing, formatting of the text and preparation of the illustrations; after using these tools, the author reviewed and edited the content as needed and takes full responsibility for the content of this publication

REFERENCES

1. Dosovitskiy, A., Beyer, L., Kolesnikov, A., et al. “An image is worth 16×16 words: Transformers for image recognition at scale”. *Proceedings of the 9th International Conference on Learning Representations (ICLR)*. 2021. DOI: <https://doi.org/10.48550/arXiv.2010.11929>.
2. Radford, A., Kim, J. W., Hallacy, C., et al. “Learning transferable visual models from natural language supervision”. *Proceedings of Machine Learning Research*. 2021; 139: 8748–8763. DOI: <https://doi.org/10.48550/arXiv.2103.00020>.
3. Zhai, X., Mustafa, B., Kolesnikov, A. & Beyer, L. “Sigmoid loss for language image pre-training”. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2023. p. 11975–11986. DOI: <https://doi.org/10.1109/ICCV51070.2023.01100>.
4. “Marqo-FashionSigLIP: model card”. *Hugging Face*. 2024. – Available from: <https://huggingface.co/Marqo/marqo-fashionSigLIP>. – [Accessed: Aug. 23, 2026].
5. Malkov, Y. A. & Yashunin, D. A. “Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2020; 42 (4): 824–836. DOI: <https://doi.org/10.1109/TPAMI.2018.2889473>.
6. Johnson, J., Douze, M. & Jégou, H. “Billion-scale similarity search with GPUs”. *IEEE Transactions on Big Data*. 2021; 7 (3): 535–547. DOI: <https://doi.org/10.1109/TBDATA.2019.2921572>.
7. Yang, F., Kale, A., Bubnov, Y., et al. “Visual search at eBay”. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2017. p. 2101–2110. DOI: <https://doi.org/10.1145/3097983.3098162>.
8. Jing, Y., Liu, D., Kislyuk, D., et al. “Visual search at Pinterest”. *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2015. p. 1889–1898. DOI: <https://doi.org/10.1145/2783258.2788621>.
9. Zhang, Y., Pan, P., Zheng, Y., et al. “Visual search at Alibaba”. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018. p. 993–1001. DOI: <https://doi.org/10.1145/3219819.3219820>.
10. Qin, X., Zhang, Z., Huang, C., et al. “U²-Net: Going deeper with nested U-structure for salient object detection”. *Pattern Recognition*. 2020; 106: 107404. DOI: <https://doi.org/10.1016/j.patcog.2020.107404>.
11. Sculley, D., Holt, G., Golovin, D., et al. “Hidden technical debt in machine learning systems”. *Advances in Neural Information Processing Systems 28 (NIPS 2015)*. 2015: 2503–2511. – Available from: https://proceedings.neurips.cc/paper_files/paper/2015/hash/86df7dcfd896fcdf2674f757a2463eba-Abstract.html. – [Accessed: Aug. 23, 2026].

12. Paleyes, A., Urma, R.-G. & Lawrence, N. D. “Challenges in deploying machine learning: A survey of case studies”. *ACM Computing Surveys*. 2022; 55 (6): 1–29. DOI: <https://doi.org/10.1145/3533378>.
13. Kreuzberger, D., Kühl, N. & Hirschl, S. “Machine learning operations (MLOps): Overview, definition, and architecture”. *IEEE Access*. 2023; 11: 31866–31879. DOI: <https://doi.org/10.1109/ACCESS.2023.3262138>.
14. Pan, J. J., Wang, J. & Li, G. “Survey of vector database management systems”. *The VLDB Journal*. 2024; 33: 1591–1615. DOI: <https://doi.org/10.1007/s00778-024-00864-x>.
15. Washizaki, H., Khomh, F., Guéhéneuc, Y.-G., Takeuchi, H., Natori, N., Doi, T. & Okuda, S. “Software-engineering design patterns for machine learning applications”. *Computer*. 2022; 55 (3): 30–39, <https://scopus.com/pages/publications/85127556940>. DOI: <https://doi.org/10.1109/MC.2021.3137227>.
16. Dragoni, N., Giallorenzo, S., Lluch Lafuente, A., Mazzara, M., Montesi, F., Mustafin, R. & Safina, L. “Microservices: Yesterday, today, and tomorrow”. *Present and Ulterior Software Engineering*. Cham: Springer. 2017. p. 195–216, <https://scopus.com/pages/publications/85054938535>. DOI: https://doi.org/10.1007/978-3-319-67425-4_12.
17. Mendonça, N. C., Aderaldo, C. M., Cámara, J. & Garlan, D. “Model-based analysis of microservice resiliency patterns”. *Proceedings of the IEEE International Conference on Software Architecture (ICSA)*. 2020. p. 114–124. DOI: <https://doi.org/10.1109/ICSA47634.2020.00019>.
18. Jonas, E., Schleier-Smith, J., Sreekanti, V. et al. “Cloud programming simplified: A Berkeley view on serverless computing”. *arXiv*. 2019. DOI: <https://doi.org/10.48550/arXiv.1902.03383>.
19. Hardt, D. “The OAuth 2.0 authorization framework”. *RFC 6749, IETF*. 2012. DOI: <https://doi.org/10.17487/RFC6749>.
20. Tyskyi, K. M. “Re-ranking of visual product search results based on a combined score of visual similarity and match count” (in Ukrainian). *Proceedings of the 1st International Scientific and Practical Conference “Modern Information Technologies: from Theory to Practice”*. Lutsk: Ukraine. 2026. – Available from: <https://mintech-conf.lntu.edu.ua/f/MInTech2026-theses.pdf>. – [Accessed: Sept. 23, 2026].
21. Tyskyi, K. M. “Quality assessment of visual recommendations: metrics of similarity, relevance and user experience” (in Ukrainian). *Science and Technology Today*. 2026. – Available from: <https://perspectives.pp.ua/index.php/nts/issue/view/472/575>. – [Accessed: Sept. 23, 2026].

Received 03.09.2026

Received after revision 11.10.2026

Accepted 14.10.2026

DOI: <https://doi.org/10.15276/ict.03.2026.44>

УДК 004.415.2:004.75

Архітектурні та інженерні аспекти інтеграції хмарних сервісів штучного інтелекту у вебзастосунок електронної комерції

Тиский Християн Михайлович¹⁾

ORCID: <https://orcid.org/0009-0005-7830-3938>; kristiantyskyi@gmail.com

¹⁾ Луцький національний технічний університет, вул. Львівська, 75. Луцьк, 43018, Україна

АНОТАЦІЯ

Сучасна електронна комерція дедалі ширше спирається на можливості штучного інтелекту, зокрема візуальний пошук та рекомендації товарів, проте для більшості середніх підприємств самостійна побудова та супровід повного стека штучного інтелекту залишаються практично недосяжними. Прагматичною альтернативою є композиція потрібної функціональності з керованих хмарних сервісів; інженерна складність такої композиції зосереджена не в моделях, а в інтеграційному шарі застосунок. У статті досліджено архітектурні та інженерні аспекти інтеграції п'яти різноманітних хмарних сервісів — керованого векторного індексу, безсерверної інференс-платформи, об'єктного сховища, публічного хмарного моделювання та керованої реляційної бази даних — у промисловий вебзастосунок пошуку весільних суконь за зображенням, що експлуатується виробником весільної моди, каталог якого налічує близько двадцяти дев'яти тисяч моделей, а торговельна мережа охоплює понад триста шістьдесят салонів у сімдесяти п'яти країнах. Запропоновано еталонну трирівневу архітектуру, в якій кожен зовнішній сервіс ізольовано за окремим адаптером, автентифікацію інкапсульовано в окремому фасаді, а взаємозамінні постачальники векторних подань зображень реалізовані за патерном проектування Стратегія, що дає змогу замінювати модель без модифікації бізнес-логіки. Інженерні рішення формалізовано аналітичними моделями доступності послідовної композиції сервісів, обмеженої схеми повторних спроб та точки беззбитковості між локальним інференсом і хмарним інтерфейсом з поштучною тарифікацією. Експериментальна оцінка промислового розгортання показала, що

зовнішні взаємодії домінують у життєвому циклі запиту: видалення фону через безсерверну платформу становить близько трьох п'ятих середнього наскрізного часу відповіді, що не перевищує чотирьох секунд, тоді як керований векторний пошук додає лише близько сімдесяти мілісекунд, тобто менше ніж два відсотки. Локальне розгортання доменно-спеціалізованого енкодера перевершує універсальний хмарний сервіс векторизації як за якістю пошуку, так і за затримкою. Упродовж чотиритижневої промислової апробації сорок п'ять консультантів виконали понад тисячу вісімсот запитів, і понад чотири п'ятих запитів отримали позитивну оцінку, що підтверджує практичну придатність запропонованої інтеграційної архітектури.

Ключові слова: хмарні сервіси штучного інтелекту; архітектура програмного забезпечення; інтеграція систем; електронна комерція; візуальний пошук; інженерія програмного забезпечення

ІНФОРМАЦІЯ ПРО АВТОРА



Khrystyian M. Tyskyi - PhD student of the Department of Computer Science and Information Technologies. Lutsk National Technical University, Lutsk, Ukraine.

ORCID: <https://orcid.org/0009-0005-7830-3938>; kristiantyskyi@gmail.com

Research field: intelligent visual search; engineering of machine-learning systems

Тиский Християн Михайлович - аспірант кафедри Комп'ютерних наук та інформаційних технологій. Луцький національний технічний університет, вул. Львівська, 75. Луцьк, 43018, Україна.

Наукові інтереси: інтелектуальний візуальний пошук; інженерія програмного забезпечення систем машинного навчання