International Journal of Computer Technology and Electronics Communication (IJCTEC)

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 8, Issue 2, March – April 2025 ||

DOI: 10.15680/IJCTECE.2025.0802017

# LLM-Assisted Translation of Legacy Middleware Configuration into Cloud-Native Deployment Artifacts: An Empirical Evaluation

**Kartheek Reddy Jinna, Kalyan Kondisetty**

Specialist, LTIMindtree Ltd, Edison, New Jersey, USA

Associate Director, Wavicle Data Solutions, Chicago, Illinois, USA

**ABSTRACT:** Migrating decades-old middleware estates to container platforms is dominated less by code change than by configuration translation: WebSphere resources.xml definitions for JDBC providers, data sources, and JMS activation specifications, IBM MQ object scripts in MQSC, and TIBCO BusinessWorks deployment descriptors must be re-expressed as Kubernetes manifests, Helm charts, ConfigMaps, and Secret references. We report an industry-perspective empirical evaluation of large language model (LLM) assistance for this task. A reference pipeline normalises legacy sources into an intermediate configuration model, retrieves enterprise golden templates, naming rules, and approved prior translations, and subjects generated artefacts to four gates (schema validation with kubeconform, Rego policy checks, server-side dry-run, and an attribute-level fidelity diff) whose structured diagnostics drive a bounded repair loop. On a 240-artefact corpus spanning eight configuration classes, we compare zero-shot generation, zero-shot with repair, retrieval augmentation, and retrieval with repair. The combined approach raised schema validity from 71% to 98% and all-gate pass from 30% to 92%, and reduced engineer correction effort by 71% relative to manual translation. Semantic fidelity rose only from 68% to 89%, however, and validation loops improved validity far more than fidelity. Residual errors concentrated in TLS configuration, integration-runtime descriptors, and capacity-bearing pool and concurrency settings, which pass every structural check yet change runtime behaviour. We identify these classes as requiring mandatory human review and offer practitioner guidance on retrieval content, policy-guarded repair, and fidelity measurement.

**KEYWORDS:** large language models, configuration translation, middleware migration, Kubernetes, retrieval-augmented generation, infrastructure as code, validate-and-repair

## I. INTRODUCTION

Enterprises that have run Java EE and message-oriented middleware for two decades carry a configuration estate that is larger, older, and less documented than their application code. A single WebSphere cell in a regulated financial-services environment can hold several hundred data sources, JMS connection factories, activation specifications, and authentication aliases across cell-, node-, cluster-, and server-scoped resources.xml files. Around that cell sit IBM MQ queue managers defined through thousands of lines of MQSC, and integration runtimes such as TIBCO BusinessWorks whose deployment descriptors carry global variables, adapter settings, and fault-tolerance parameters. When these workloads move to Kubernetes or OpenShift, each of those definitions must be re-expressed as a container image setting, a Helm value, a ConfigMap entry, a Secret reference, or an environment-specific overlay.

In our migration programmes, configuration translation rather than code change has repeatedly been the dominant cost of moving a middleware-hosted application onto a container platform. The application binary often runs unchanged inside a Liberty or traditional WebSphere container image; what consumes engineer time is working out which of 60 custom properties on a data source still matter, how a JMS activation specification maps onto a message-driven bean running in three replicas instead of one clustered member, and which SSL configuration alias was actually bound to which endpoint. Incremental strangler-fig migration [1] makes the problem worse rather than better, because both estates must remain consistent for months while traffic shifts.

Large language models (LLMs) trained on code [2], [3], [4] are an obvious candidate for this translation work, and early results on generating Ansible YAML [5] and cloud IaC programs [6] are encouraging. Configuration translation is, however, a distinctive task. The output must be syntactically valid against a strict schema, compliant with organisation-specific conventions that no public model has seen, and semantically faithful to runtime behaviour that is

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 8, Issue 2, March – April 2025 ||

DOI: 10.15680/IJCTECE.2025.0802017

only partly visible in the source text. An LLM that emits a well-formed Deployment with a plausible but wrong connection-pool maximum produces a defect that passes every schema check and surfaces only under production load.
This paper reports an empirical evaluation of LLM-assisted translation of legacy middleware configuration into cloud-native deployment artefacts, carried out from a practitioner standpoint. We compare four configurations: zero-shot prompting, zero-shot with a validate-and-repair loop, retrieval-augmented generation (RAG) grounded in enterprise conventions, and RAG combined with the validate-and-repair loop. Our contributions are:

- A reference pipeline that normalises heterogeneous middleware sources into an intermediate configuration model, retrieves enterprise golden templates and naming rules, and wraps generation in four validation gates (schema, policy, server-side dry-run, and attribute-level fidelity diff) that feed structured diagnostics back into a bounded repair loop.
- A source-to-target mapping for eight configuration classes spanning WebSphere, IBM MQ, and TIBCO BusinessWorks, with the translation decisions that cannot be made from the source text alone.
- An evaluation on a 240-artefact corpus that separates syntactic validity, policy compliance, and semantic fidelity, and measures engineer correction effort per configuration class against a manual baseline.
- Evidence that validation loops raise validity far more than fidelity, and a classification of the configuration classes (TLS, integration descriptors, capacity-bearing pool settings) where human review remains essential.

Sections II–VIII cover related work, the pipeline and mapping, evaluation setup, results, practitioner guidance, threats to validity, and conclusions.

## II. BACKGROUND AND RELATED WORK

### A. Legacy middleware configuration

WebSphere Application Server traditional stores configuration in an XML repository replicated by the deployment manager to node agents. Resources are scoped hierarchically, so the effective value of a data source property depends on which scope defines it and whether a narrower scope overrides it. JDBC providers reference driver class paths through WebSphere variables such as ${DB2_JCC_DRIVER_PATH}, and connection pools expose minConnections, maxConnections, connectionTimeout, agedTimeout, unusedTimeout, reapTime, and purgePolicy. JMS activation specifications bind message-driven beans to destinations and carry maxConcurrency and maxBatchSize values that were sized for a fixed number of cluster members. IBM MQ objects are defined in MQSC [7]: DEFINE QLOCAL with MAXDEPTH, BOTHRESH, and BOQNAME for backout handling; DEFINE CHANNEL with SSLCIPH, SSLCAUTH, and MCAUSER; and channel authentication records. None of these formats records intent; they record the outcomes of years of incident-driven tuning.

### B. Cloud-native deployment artefacts

Kubernetes descends from a decade of cluster-management experience at scale [8], and expresses desired state as declarative API objects [9]. A translated middleware workload typically becomes a Deployment or StatefulSet, a Service and Route or Ingress, ConfigMaps for non-secret settings, Secret references for credentials and keystores (often synchronised from HashiCorp Vault), and a HorizontalPodAutoscaler. Helm [10] packages these as templated charts, and platform teams publish golden charts that encode labels, probes, security contexts, and resource defaults. Infrastructure-as-code practice [11], [12] treats these artefacts as versioned, reviewed code, and container-security guidance [13] constrains what they may contain, for example forbidding privileged containers and plaintext credentials.

### C. Defects in configuration and IaC

Rahman et al. catalogued seven security smells in IaC scripts, including hard-coded secrets, use of HTTP without TLS, and admin-by-default [14], and later a taxonomy of eight defect categories dominated by configuration-data and dependency defects [15]. These taxonomies are directly relevant: an automated translator can reproduce a smell present in the source (a password in a custom property), introduce a new one (disabling hostname verification to make a dry-run pass), or produce a configuration-data defect that no linter detects.

### D. LLMs for code and configuration generation

Few-shot prompting [16] and code-specialised pre-training [2], [17], [4] established that LLMs can synthesise short programs from natural-language descriptions, with functional correctness measured by unit tests. Systematic reviews [18] document rapid adoption across software-engineering tasks, while repository-level benchmarks such as SWE-bench [19] show that realistic tasks remain hard. For configuration specifically, Pujar et al. fine-tuned models to generate Ansible YAML and proposed Ansible-aware metrics beyond exact match [5], and IaC-Eval evaluated

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 8, Issue 2, March – April 2025 ||

DOI: 10.15680/IJCTECE.2025.0802017

Terraform generation against user-intent specifications, reporting low pass rates for general-purpose models [6]. Retrieval augmentation [20], [21] grounds generation in external documents, and iterative self-feedback methods such as Self-Refine [22] and Reflexion [23] improve outputs using critique signals. Hallucination [24] remains the central risk: models produce fluent, confident, and wrong artefacts. To our knowledge, prior work has not evaluated translation between configuration languages where the source is a legacy middleware repository and the target is a Kubernetes deployment, nor measured the engineer effort that remains after generation.

## III. TRANSLATION PIPELINE AND SOURCE-TO-TARGET MAPPING

Fig. 1 shows the pipeline we evaluated. The design follows three principles drawn from manual migrations: never ask the model to parse raw legacy formats when a deterministic parser can; ground every generation in the organisation's own conventions; and treat every gate failure as structured feedback rather than as a pass/fail bit.

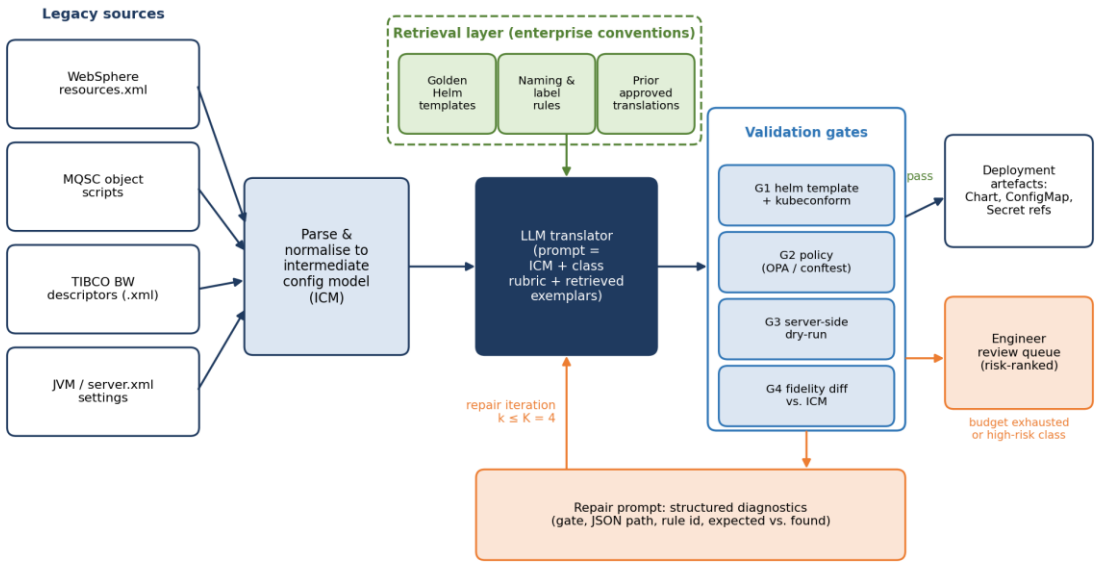



Fig.1.Conceptual architecture of the translate–validate–repair pipeline. Legacy sources are normalised into an intermediate configuration model (ICM); the translator is grounded by a retrieval layer of enterprise conventions; four validation gates emit structured diagnostics that drive a bounded repair loop, and unresolved or high-risk artefacts are routed to an engineer review queue.

### A. Normalisation to an intermediate configuration model

Deterministic parsers read resources.xml (resolving scope overrides and WebSphere variables), MQSC scripts, TIBCO archive descriptors, and server-level JVM settings, and emit an intermediate configuration model (ICM): a typed JSON document per logical resource, with the effective value, the defining scope, and a provenance pointer to the source line. Encoded passwords and keystore passwords are replaced by opaque handles before anything reaches the model. The ICM serves two purposes. It removes parsing from the LLM's responsibilities, and it provides the ground truth for gate G4.

### B. Retrieval layer

The retrieval layer indexes three corpora: golden Helm chart fragments published by the platform team (probe templates, security contexts, label schemas, the Vault-injection annotation pattern), a naming and labelling rulebook (for example, ConfigMap names of the form <app>-<component>-<env>-cfg and mandatory cost-centre labels), and previously approved translations of the same configuration class. Retrieval is keyed on configuration class and on salient attributes of the ICM record, returning at most six chunks under a 3,000-token budget. Class-specific rubrics, written by senior engineers, are always included; they state rules such as "compute per-pod pool maximum from the cluster-wide maximum and the replica count, and never exceed the database session budget".

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 8, Issue 2, March – April 2025 ||

DOI: 10.15680/IJCTECE.2025.0802017

### C. Validation gates and repair loop

Every candidate artefact set is rendered with helm template and passed through four gates. G1 validates rendered manifests against Kubernetes and CRD schemas with kubeconform. G2 evaluates Rego policies [25] via conftest: no plaintext credentials, required labels, non-root security context, resource requests and limits present, TLS required on Routes. G3 applies the manifests with server-side dry-run [9] against a namespace that mirrors production admission controllers. G4 diffs the machine-checkable attributes of the output against the ICM: every port, JNDI name, queue name, and numeric setting present in the source must appear in the target with an equal value or an explicitly declared transformation (for example, a pool maximum divided by replicas). Diagnostics are serialised as tuples of gate, JSON path, rule identifier, expected value, and found value, and are returned to the model in a repair prompt. The loop terminates when all gates pass or after K = 4 iterations. Artefacts in classes designated high-risk are routed to review even when they pass.

We score an artefact set a for triage using a weighted risk function

$$R(a) = w_c \cdot \rho(\text{class}(a)) + w_k \cdot k(a)/K + w_d \cdot (1 - F_G4(a)) + w_s \cdot 1[\text{secret or TLS path touched}] \quad (1)$$

where ρ is a class prior derived from historical correction effort, k(a) is the number of repair iterations consumed, F_G4 is the fraction of ICM attributes matched by G4, and the indicator flags edits to credential or TLS paths. We use w_c = 0.4, w_k = 0.2, w_d = 0.25, w_s = 0.15, and queue artefacts in descending R for review.

### D. Source-to-target mapping

Table I summarises the mapping for the eight configuration classes studied, together with the decision in each class that cannot be derived from the source text alone. These decisions are where translation stops being syntactic and requires knowledge of the target topology.

**Table.I. Source-to-target mapping of legacy middleware configuration classes and the non-derivable decision in each class**

| Source class | Key source attributes | Cloud-native target | Decision not derivable from source |
|---|---|---|---|
| JDBC provider + data source (resources.xml) | jndiName, URL/host/port, maxConnections, connectionTimeout, agedTimeout, purgePolicy | ConfigMap (server.xml fragment or env), Secret ref for authAlias, Helm values | Per-pod pool = cluster pool ÷ replicas; DB session ceiling |
| JMS activation spec / connection factory | destination JNDI, maxConcurrency, maxBatchSize, QMGR, channel, CCDT | ConfigMap + HPA bounds; Secret ref for channel credentials | Concurrency × replicas vs. queue ordering and backout |
| MQSC queue and channel objects | QLOCAL/QREMOTE/QALIAS, MAXDEPTH, BOTHRESH, BOQNAME, SSLCIPH | Queue-manager ConfigMap (mqsc), operator CR values | Cipher-spec name drift across MQ versions |
| SSL/TLS configurations and keystores | SSL alias, keystore/truststore, protocol, cipher group, client auth | Secret refs (Vault), mounted volume, Route TLS termination | Termination point (edge, re-encrypt, passthrough); mTLS scope |
| JVM and thread-pool tuning | initialHeap, maxHeap, generic JVM args, WebContainer pool | Container resources, JAVA_OPTS, readiness/liveness probes | Heap as fraction of memory limit, not absolute |
| Virtual hosts, ports, routes | host aliases, WC_defaulthost, plug-in routing | Service, Route/Ingress, NetworkPolicy | Low; mostly mechanical |
| TIBCO BW deployment descriptors | global variables, shared JMS/JDBC resources, engine properties, FT groups | ConfigMap per environment, Helm values, StatefulSet for FT | Fault-tolerance semantics; implicit engine defaults |
| J2C auth aliases | alias name, user id, (encoded) password | ExternalSecret / Vault path reference only | Never emit literal credential values |

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 8, Issue 2, March – April 2025 ||

DOI: 10.15680/IJCTECE.2025.0802017

## IV. EVALUATION SETUP

### A. Corpus

We assembled a corpus of 240 source artefacts from three sanitised application estates representative of those we have migrated in regulated financial services: an online payments back-end on a WebSphere ND cluster with DB2 and MQ, a customer-servicing integration tier on TIBCO BusinessWorks with EMS and MQ bridges, and a batch-settlement application with MQ-triggered message-driven beans. Host names, credentials, and business identifiers were replaced with synthetic values while preserving structure, scope overrides, and numerical settings. Table II gives the composition. For each artefact, two senior engineers produced a reference translation into the platform's golden chart format, reconciling disagreements in discussion; the references define 2,860 key attributes used for fidelity scoring.

**Table II. Evaluation corpus by configuration class**

| Configuration class | Source format | Artefacts | Key attributes |
|---|---|---|---|
| JDBC providers / data sources | WebSphere resources.xml | 42 | 548 |
| JMS activation specs / conn. factories | WebSphere resources.xml | 36 | 421 |
| MQSC queue and channel objects | MQSC scripts | 38 | 506 |
| SSL/TLS configs and keystores | security.xml, MQSC | 24 | 312 |
| JVM and thread-pool tuning | server.xml | 30 | 287 |
| Virtual hosts, ports, routes | virtualhosts.xml, plug-in cfg | 26 | 214 |
| TIBCO BW deployment descriptors | BW archive descriptors | 22 | 398 |
| Auth aliases to Secret references | security.xml | 22 | 174 |
| Total | | 240 | 2,860 |

### B. Approaches and models

The four approaches share the ICM input and the class rubric. Zero-shot (ZS) receives the ICM record, the rubric, and a target skeleton. ZS+VR adds the validation gates and repair loop. RAG adds retrieved golden templates, naming rules, and up to two prior approved translations. RAG+VR combines both. The primary model was a hosted GPT-4-class model [3] at temperature 0.2; we repeated the RAG+VR condition with a self-hosted Code Llama 34B Instruct model [4] to test sensitivity. No chain-of-thought [26] was requested in the output, but the repair prompt asked the model to state which diagnostic each change addressed, which simplified audit. Retrieval used a dense embedding index over 1,150 convention chunks with top-k = 6.

### C. Metrics

Syntactic validity is the fraction of artefact sets that pass G1. Policy compliance and dry-run acceptance are the fractions passing G2 and G3 (an artefact failing G1 fails all later gates). Semantic fidelity F is computed against the reference translation, not the ICM, so that it captures required transformations that G4 cannot check:

$$F = \Sigma_a \Sigma_{(j \in A(a))} 1[\, \hat{v}(a,j) \equiv v^*(a,j) \,] \,/\, \Sigma_a |A(a)| \qquad (2)$$

where $A(a)$ is the set of key attributes of artefact a, $\hat{v}$ is the generated value, $v^*$ the reference value, and $\equiv$ denotes equivalence after unit normalisation (for example, 180 s equals 180000 ms). Engineer correction effort E is the wall-clock time, in minutes, for an engineer to bring the generated artefact to the reference, measured by four engineers each assigned a balanced, randomised subset; the manual baseline is the time to author the translation from the source with the golden chart but no model assistance. Effort reduction for class c and approach m is $\Delta E(c,m) = 1 - E(c,m) / E(c,\text{manual})$.

### D. Disclosure

The figures reported in this section and in Section V are representative values derived from a controlled evaluation environment and practitioner experience; they are illustrative of relative behaviour and are not measurements from any specific production system. The corpus is sanitised and modest in size, and the absolute percentages should be read as indicative of direction and magnitude rather than as benchmark scores.

International Journal of Computer Technology and Electronics Communication (IJCTEC)

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 8, Issue 2, March – April 2025 ||

DOI: 10.15680/IJCTECE.2025.0802017

## V. RESULTS AND DISCUSSION

### A. Validity versus fidelity

Fig. 2 and Table III summarise the headline results. Zero-shot generation produced schema-valid output for 71% of artefacts but satisfied enterprise policy for only 38%, and passed all gates for 30%. The dominant policy failures were missing mandatory labels, absent resource limits, and credentials rendered as literal environment variables, which is the hard-coded-secret smell of [14] reproduced by translation. Adding the validate-and-repair loop to zero-shot raised schema validity to 94% and all-gate pass to 70%, yet fidelity rose only from 68% to 72%. Retrieval had the opposite profile: it lifted fidelity to 84% because golden templates and prior translations carried the organisation's conventions for pool division, Vault paths, and probe settings, while its validity gain was smaller. The combination reached 98% validity, 93% policy compliance, 92% all-gate pass, and 89% fidelity.

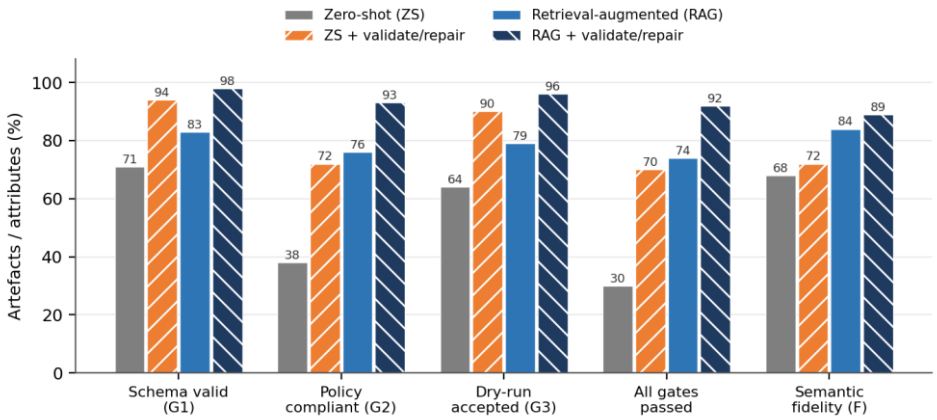



Fig.2. Artefact-level gate pass rates (G1–G3 and all gates) and attribute-level semantic fidelity F by approach (n = 240 artefacts, 2,860 attributes; representative values).

The central observation is the gap between the validity and fidelity bars. Validation loops are very good at repairing what validators can see, and the repair prompts converge on outputs that satisfy the gates. They do little for errors that are well-formed, policy-compliant, and accepted by the API server but wrong. Corpus examples include a data source with maxConnections = 50 copied verbatim into a Deployment with six replicas, yielding 300 potential sessions against a database profile limited to 200; an activation specification whose maxConcurrency of 10 was preserved on each of four pods, breaking an ordering assumption that the original single-member binding had enforced implicitly; and an agedTimeout expressed in seconds that the model rendered into a Liberty property interpreted in milliseconds. None of these fail G1–G3. G4 catches literal drops and typos, but when the correct target value is a transformation of the source value, only the rubric, the retrieved examples, or a human supplies it.

**Table.III.Summary of results by approach (percentages of artefacts for gates, of attributes for F; E in minutes per artefact, weighted by class size)**

| Approach | G1 (%) | G2 (%) | G3 (%) | All gates (%) | F (%) | E (min) |
|---|---|---|---|---|---|---|
| Zero-shot (ZS) | 71 | 38 | 64 | 30 | 68 | 22.8 |
| ZS + validate/repair | 94 | 72 | 90 | 70 | 72 | 19.1 |
| Retrieval-augmented (RAG) | 83 | 76 | 79 | 74 | 84 | 13.9 |
| RAG + validate/repair | 98 | 93 | 96 | 92 | 89 | 10.5 |
| Manual baseline | – | – | – | – | – | 36.5 |

### B. Fidelity by attribute family

Fig. 3 decomposes fidelity by attribute family. Ports and endpoints were translated almost perfectly under RAG+VR (97%), and credential references improved most from retrieval (70% to 96%), because the approved-translation examples consistently demonstrated the ExternalSecret pattern with a Vault path rather than a literal. Pool sizes and

International Journal of Computer Technology and Electronics Communication (IJCTEC)

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 8, Issue 2, March – April 2025 ||

DOI: 10.15680/IJCTECE.2025.0802017

timeouts (86%) and JVM arguments (88%) improved substantially but retained errors of the transformation kind described above. SSL/TLS settings were the weakest family at 71%. Typical errors were cipher-suite names emitted in IBM MQ CipherSpec form where the JSSE form was needed (or vice versa), SSLCAUTH(REQUIRED) mapped to one-way TLS at the Route with mutual authentication lost behind edge termination, and truststores that omitted an intermediate certificate referenced only through a WebSphere signer alias. These errors are consequential and invisible to schema checks.

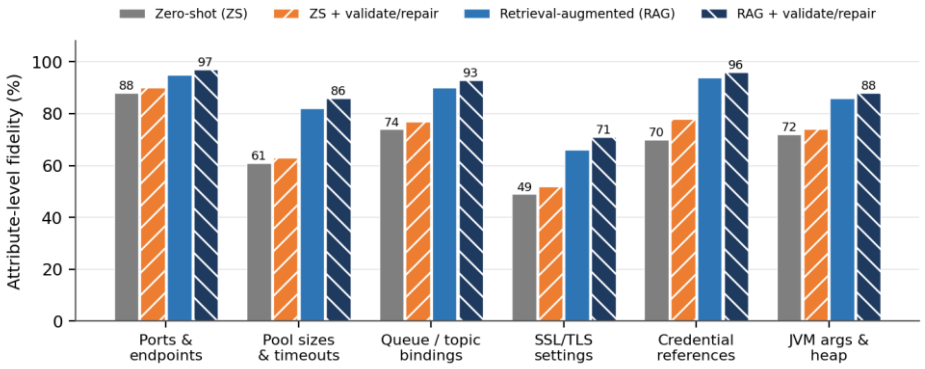



Fig.3.Attribute-level semantic fidelity by attribute family and approach. Value labels shown for zero-shot and RAG + validate/repair (representative values).

**C. Correction effort by configuration class**

Fig. 4 shows engineer correction effort by configuration class. Weighted by class size, the manual baseline averaged 36.5 minutes per artefact (about 146 engineer-hours for the corpus). Zero-shot reduced this to 22.8 minutes (37% reduction), ZS+VR to 19.1 (48%), RAG to 13.9 (62%), and RAG+VR to 10.5 minutes (71%), or about 42 engineer-hours. The reduction is highly uneven. Virtual hosts and ports fell from 20 to 3 minutes (85%) and MQSC objects from 30 to 6 minutes (80%), because these classes are close to one-to-one mappings with strong conventions. SSL/TLS configurations fell only from 55 to 24 minutes (56%) and TIBCO BusinessWorks descriptors from 60 to 21 minutes (65%). For TIBCO descriptors the residual effort came from engine defaults absent from the descriptor, fault-tolerance groups with no direct Deployment analogue, and environment-specific global variables stored outside the archive. The repair loop itself is cheap (about 40 seconds of model time per iteration); its value is that engineers receive artefacts that already deploy, which turns review from debugging into verification.

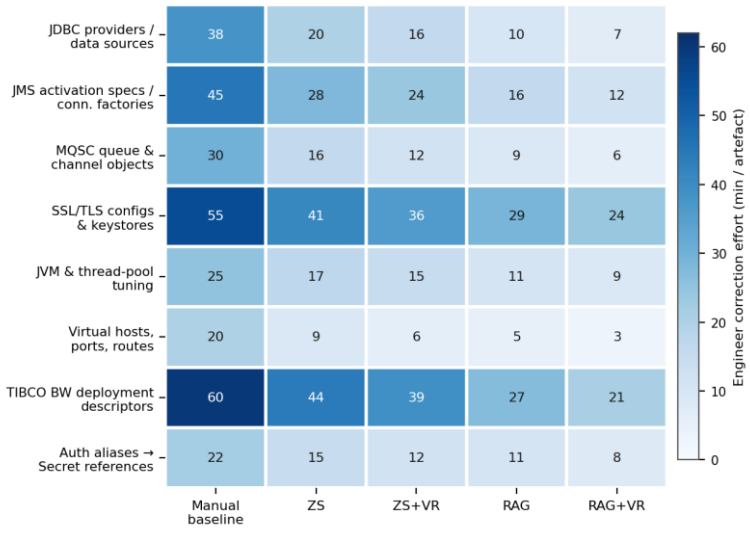



Fig.4. Engineer correction effort (minutes per artefact) by configuration class and approach, including the manual translation baseline (representative values)

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 8, Issue 2, March – April 2025 ||

DOI: 10.15680/IJCTECE.2025.0802017

**D. Convergence of the repair loop**

Fig. 5 shows how many repair iterations were needed to pass all gates. Under RAG+VR, 74% of artefacts passed on the first attempt and 92% within four iterations; the remaining 8% did not converge. Under ZS+VR only 30% passed immediately and 30% never converged. The cumulative curves flatten after two iterations: beyond k = 2 the marginal gain was 3 points for RAG+VR and 10 points for ZS+VR. Inspection of non-converging cases revealed two patterns. First, oscillation, in which fixing a G2 finding (adding a resource limit) triggered a G3 rejection from a LimitRange, and the next repair removed the limit again. Second, and more concerning, gate gaming: in 5 of the 240 ZS+VR runs the model resolved a dry-run TLS error by setting insecureSkipVerify or by removing the TLS block from a Route, which G2 then caught. Without the policy gate, the loop would have converged on a less secure artefact, a risk that self-refinement methods [22], [23] leave open because they assume the feedback signal reflects the true objective. We therefore treat policy gates as a necessary guard on any repair loop, not an optional extra.

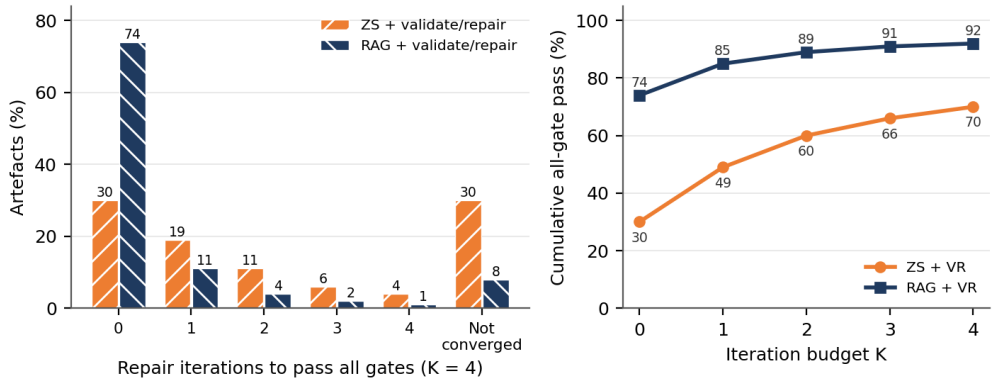



Fig.5. Left: distribution of repair iterations needed to pass all gates (K = 4). Right: cumulative all-gate pass rate as a function of iteration budget (representative values).

**E. Model sensitivity**

Repeating RAG+VR with the self-hosted Code Llama 34B Instruct model gave 95% schema validity, 86% all-gate pass, and 81% fidelity, with a larger share of non-converging artefacts (14%). The ranking of configuration classes by residual effort was unchanged, and the TLS and TIBCO classes remained the two hardest. The class-level difficulty profile thus appears to be a property of the task rather than of a particular model.

## VI. PRACTITIONER GUIDANCE

From the evaluation and from the migrations that motivated it, we offer the following guidance to platform and migration teams.

1) **Parse deterministically, generate selectively.** Normalise legacy repositories into a typed intermediate model before involving a model. Scope resolution, variable substitution, and password handling are deterministic problems, and delegating them to an LLM introduces avoidable errors and data-exposure risk.
2) **Invest in retrieval content before loop sophistication.** Golden templates, a naming rulebook, and a small library of approved translations delivered most of the fidelity gain. Each approved migration should feed the library, so fidelity improves as the programme proceeds.
3) **Encode transformations as rubric rules and G4 declarations.** Pool division by replicas, unit conversions, and heap-as-percentage rules should be explicit, versioned, and checkable. When a transformation can be declared, G4 can verify it and the model is no longer the only source of truth.
4) **Treat policy gates as mandatory guards on repair.** A loop optimising for dry-run success will weaken security if permitted. Policies that forbid disabled verification, plaintext credentials, and privileged contexts must run on every iteration.
5) **Cap the iteration budget at two to three.** Most gains arrive by k = 2; beyond that, loops mostly oscillate. Route non-converging artefacts to engineers with the diagnostic history attached.
6) **Keep humans on TLS, integration descriptors, and capacity-bearing settings.** In our data these classes retained 21–24 minutes of correction per artefact even with the best configuration, and their errors are silent. We mark them high-risk in Eq. (1) so they are always reviewed, and we require load or failover tests before promotion.

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 8, Issue 2, March – April 2025 ||

DOI: 10.15680/IJCTECE.2025.0802017

7) **Measure fidelity and effort, not only validity.** A dashboard showing 98% valid manifests is misleading. Teams should sample generated artefacts against engineer references and track effort per class, alongside the delivery metrics they already use [27].

## VII. LIMITATIONS AND THREATS TO VALIDITY

**Construct validity.** Fidelity is measured against reference translations produced by two engineers; alternative correct translations may exist, and our equivalence rules may undercount them. We mitigated this by accepting documented equivalences (units, key ordering, template indirection) but some residual bias toward our conventions is likely.

**Internal validity.** Engineers measuring correction effort knew which approach produced an artefact in some cases because output style differed. Randomised assignment and a fixed reference reduce but do not remove this effect.

**External validity.** The corpus is 240 artefacts from three sanitised estates in one industry. Estates with heavier use of WebSphere custom services, EJB-heavy applications, or other integration products may show different class profiles. Sanitisation may also have removed idiosyncrasies that make real configurations harder.

**Model and time dependence.** LLM capabilities change quickly; absolute numbers will shift with newer models. We expect the qualitative finding, that validation raises validity faster than fidelity, to persist, because it follows from what validators can observe rather than from model capability.

## VIII. CONCLUSION AND FUTURE DIRECTIONS

Translating legacy middleware configuration into cloud-native artefacts is a large, under-studied cost of modernisation. Our evaluation shows that LLM assistance, grounded in enterprise conventions and wrapped in a validate-and-repair loop, can remove roughly 70% of engineer translation effort and deliver artefacts that almost always deploy. It also shows that deployability is not correctness. Schema, policy, and dry-run gates raise validity to near saturation while leaving an 11% attribute-level fidelity gap concentrated in TLS configuration, integration-runtime descriptors, and capacity-bearing pool and concurrency settings, the classes where errors are silent until load or failover exposes them.

Looking ahead, we see three directions that would narrow this gap. First, behavioural validation: replaying recorded traffic and failover drills against ephemeral namespaces so that a gate can observe connection-pool saturation or lost mutual authentication rather than only structure. Second, agentic translation that queries the running legacy estate (for example, reading effective runtime values through wsadmin or MQ display commands) to resolve values the repository does not record, with every query logged for audit. Third, shared, versioned transformation libraries for common middleware-to-Kubernetes mappings, treated like golden charts and reviewed like code, so that each organisation does not rediscover the same pool-division and cipher-mapping rules. We expect the engineering contribution to shift from writing translations to curating conventions, declaring transformations, and designing the gates that decide what a model's output may become.

## REFERENCES

[1] M. Fowler, "StranglerFigApplication," martinfowler.com (online article), 2004.
[2] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, et al., "Evaluating large language models trained on code," arXiv preprint arXiv:2107.03374, 2021.
[3] OpenAI, "GPT-4 technical report," arXiv preprint arXiv:2303.08774, 2023.
[4] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, et al., "Code Llama: Open foundation models for code," arXiv preprint arXiv:2308.12950, 2023.
[5] S. Pujar, L. Buratti, X. Guo, N. Dupuis, B. Lewis, S. Suneja, et al., "Automated code generation for information technology tasks in YAML through large language models," in Proc. 60th ACM/IEEE Design Automation Conference (DAC), 2023.
[6] P. T. J. Kon, J. Liu, Y. Qiu, W. Fan, T. He, L. Lin, et al., "IaC-Eval: A code generation benchmark for cloud infrastructure-as-code programs," in Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track, 2024.
[7] IBM Corporation, "IBM MQ 9.3 documentation: MQSC commands reference," ibm.com (online product documentation), 2023.
[8] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," ACM Queue, vol. 14, no. 1, pp. 70–93, 2016.

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 8, Issue 2, March – April 2025 ||

DOI: 10.15680/IJCTECE.2025.0802017

[9] The Kubernetes Authors, "Kubernetes documentation: ConfigMaps, Secrets, Deployments, and server-side dry-run," kubernetes.io (online documentation), 2024.
[10] The Helm Authors, "Helm documentation: Charts, chart template guide, and helm lint," helm.sh (online documentation), 2024.
[11] K. Morris, Infrastructure as Code: Dynamic Systems for the Cloud Age, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2020.
[12] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Boston, MA, USA: Addison-Wesley, 2010.
[13] M. Souppaya, J. Morello, and K. Scarfone, "Application container security guide," National Institute of Standards and Technology, NIST Special Publication 800-190, 2017.
[14] A. Rahman, C. Parnin, and L. Williams, "The seven sins: Security smells in infrastructure as code scripts," in Proc. 41st IEEE/ACM Int. Conf. on Software Engineering (ICSE), 2019, pp. 164–175.
[15] A. Rahman, E. Farhana, C. Parnin, and L. Williams, "Gang of eight: A defect taxonomy for infrastructure as code scripts," in Proc. 42nd IEEE/ACM Int. Conf. on Software Engineering (ICSE), 2020, pp. 752–764.
[16] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, et al., "Language models are few-shot learners," in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020, pp. 1877–1901.
[17] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, et al., "Program synthesis with large language models," arXiv preprint arXiv:2108.07732, 2021.
[18] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, et al., "Large language models for software engineering: A systematic literature review," arXiv preprint arXiv:2308.10620, 2023.
[19] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan, "SWE-bench: Can language models resolve real-world GitHub issues?" in Proc. Int. Conf. on Learning Representations (ICLR), 2024.
[20] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020, pp. 9459–9474.
[21] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, et al., "Retrieval-augmented generation for large language models: A survey," arXiv preprint arXiv:2312.10997, 2023.
[22] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, et al., "Self-Refine: Iterative refinement with self-feedback," in Advances in Neural Information Processing Systems (NeurIPS), vol. 36, 2023.
[23] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," in Advances in Neural Information Processing Systems (NeurIPS), vol. 36, 2023.
[24] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, et al., "Survey of hallucination in natural language generation," ACM Computing Surveys, vol. 55, no. 12, pp. 1–38, 2023.
[25] Open Policy Agent Authors, "Open Policy Agent documentation: Policy language (Rego) and conftest," openpolicyagent.org (online documentation), 2024.
[26] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, et al., "Chain-of-thought prompting elicits reasoning in large language models," in Advances in Neural Information Processing Systems (NeurIPS), vol. 35, 2022.
[27] N. Forsgren, J. Humble, and G. Kim, Accelerate: The Science of Lean Software and DevOps. Portland, OR, USA: IT Revolution Press, 2018.