A RAG-enriched LLM to synthesize Architecture Decision Records from microservices deployment and source code

[Authors names anonymized]

***Abstract*—Capturing architectural decisions is key to support a system evolution, yet many projects do not follow this practice, leaving decisions hidden in the source code or other artifacts, and making design knowledge vaporize. Manual recovery of decision from source code and deployment information has been studied, but remains a time-consuming and error-prone task. This article presents an LLM-based approach to recover (or infer) the design decisions made between two consecutive versions of a microservice-based system. It inputs source code and deployment information (Infrastructure-as-Code files, IaC), and generates reviewable Architecture Decision Records (ADRs). The method uses a pipeline with two stages: (1) generate (textual) architecture descriptions from the initial and later system versions' artifacts, analyzing the deployment and source code with a design rules catalog; (2) perform an architecture-aware diff between these two descriptions, synthesizing a set of candidate ADRs. Both phases combine a Retrieval-Augmented Generation (RAG) schema, which provides domain knowledge in the form of few-shot examples and prompts, and the pretrained knowledge of a Large Language Model (LLM). The approach was assessed with several GitHub projects that have source code, IaC files, and ADRs, and the synthesized ADRs were also evaluated by a small set of architecture practitioners. Initial results show that the method recovers most decisions documented in the ADRs, and also provides additional candidate decisions that practitioners perceived as useful and accurate. Decision mining from code and deployment artifacts is a promising direction to mitigate architectural knowledge vaporization in evolving systems.**

***Index Terms*—software architecture recovery; Architecture Decision Records (ADR); Large Language Models (LLM); Retrieval-Augmented Generation (RAG)**

I. Introduction

In industrial projects, the documentation of architecture design decisions is often overlooked, despite the fact that architecture decisions have significant effects on system quality and cost [1]. In particular, popular approaches for capturing decisions, such as Architecture Decision Records (ADRs), are not without adoption challenges [2]. Architects appreciate ADRs, but that manual mining from source code, documentation, and operational assets is costly and complicated. In many projects, ADRs seem to be missing, outdated, or incomplete. When these decisions are not systematically captured, systems start to suffer from knowledge vaporization issues, and performing any kind of architectural reasoning about the system (e.g., an architecture evaluation to determine alternatives for a migration plan) becomes difficult and time-consuming.

In recent years, the widespread adoption of microservices and cloud-native technologies has accelerated architectural evolution, leaving ever more decisions undocumented due to time pressures or other business factors [3]. Many architectural changes end up captured in DevOps and IaC scripts; e.g. a deployment automation, quality, and security study of IaC [4] revealed systemic configuration errors and the need for better tooling. Large language models (LLMs) are being used in CI/CD pipelines to evaluate rules and reason about complex security restrictions on developing infrastructures [5]. The software architecture literature includes lightweight ADR templates (e.g. Markdown ADRs[1], MADR [6]) and decision archaeology [7] to document legacy and to migrate systems, including semi-structured mining of heterogeneous artifacts.

This article investigates whether changes in a system's deployment configuration (as Infrastructure-as-Code, IaC) and source code can be exploited to generate synthetic, evidence-grounded ADRs. It proposes a LLM+RAG pipeline that (i) derives architectural semantics for two versions of the same system; (ii) computes an "architecturally-aware diff" to identify decision candidates; and (iii) generates ADRs for these decisions, using a MADR template.

The rest of this article is structured as follows: Section II surveys related work; Section III describes the proposed approach; Section IV reports the approach evaluation using three open-source systems from the ADR-Study dataset (Abelaa, Serverlessmike, and Chef); Section V addresses threats to validity; and Section VI summarizes and concludes.

II. Related Work

Several approaches have been proposed to recover or generate ADRs for systems already existing or under development, with methodological approaches ranging from semi-automated recovery techniques to fully automated LLM-based generation.

Archium [8] generated ADRs by treating architectural decisions as first-class entities, and integrating them into the development process through an architectural description language (ADL) combined with Java. This tool provided consistency checks and supported tracing requirements to architectural decisions, yielding structured templates that capture context, decision, rationale, consequences, and alternatives. ADDRA [9] employed an iterative five-step process that compared architectural views across releases to identify deltas. It required that the original architect decision-maker be still available.

STREAM-ADD [10] extended its authors' STREAM model-driven process to systematize the documentation of

[1]https://anonymous.4open.science/r/adrtfcodesynth-5513

architectural decisions as they are made. It supported architecture refinement with decisions documented in ADRs, and illustrated the approach with a route-planning system.

TREx [11], [12] extracte design rationale from unstructured text documents (namely, e-mails, meeting notes, and wikis), combining NLP pattern-based information extraction with ontology-based representation. Experimental evaluation showed that tool-supported TREx increased recall (but not precision) compared to manual approaches.

RecovAr/UnArch [13] mined project history artifacts, and mapped version control commits to issues, to identify decisions affecting system architecture. It yielded a decision graph that integrates architectural changes with their associated issues. This approach achieved a recall of 75% and precision of 77% on large open-source systems.

ADeX [14] is a tool and approach that automatically recovers design decisions while deriving further architectural knowledge, including related quality attributes and architectural elements. It maintains its knowledge base updated from several systems used during the software development lifecycle.

Recent work has explored the use of Large Language Models (LLMs) for ADR generation, typically relying on curated datasets of ADRs and design decisions extracted from open-source projects. Dhar et el. [15] used GPT and T5-based models, and found that GPT-4 generates relevant and accurate design decisions in zero-shot settings, achieving a BERTScore of 0.849, quite good but still short of human-level performance. Other models like GPT-3.5 achieve similar results with few-shot prompting, and smaller models like Flan-T5 yield comparable results after fine-tuning.

Finally, an unplublished-yet-available preprint describes DRAFT [16], which combines retrieval-augmented generation (RAG) with LLM fine-tuning. It has two phases (offline fine tuning and online generation), and achieves a ROUGE-1 score of 0.493, BLEU score of 0.221, METEOR score of 0.430, and BERTScore F1 of 0.885. DRAFT claims to outperform zero-shot prompting, RAG alone, and fine-tuning alone in effectiveness, while maintaining efficiency.

These methods approximate ADRs (or decision descriptions) at the sentence or paragraph level, to optimize similarity against existing corpora. Complete MADR-style records based on concrete infrastructure evolution are not their goal.

III. Proposal

The technique synthesizes ADRs from deployment and source code artifacts, as shown in Fig. 1. It involves a pipeline with three phases, each one supported by architectural knowledge and also relying on an LLM.

The ADRs resulting from the pipeline capture the decisions that took a system between two versions v_A and v_B. This article focused on the common case where two snapshots of the same system (e.g., a minor and a major Terraform evolution) are available, and both can be treated as an evolution step $v_A = t_i \rightarrow v_B = t_j$ with $i < j$ of the same system. A key assumption is that the delta between v_A and v_B encodes a non-trivial set of architectural decisions (e.g., migration steps, deployment changes); implications of this assumption are addressed in Sect. V.

Other configurations (e.g., single-snapshot analysis against an initial baseline t_0) are conceptually supported by the pipeline, but left for future work.

A. Pipeline overview

The method pipeline (see Fig. 1) takes two inputs:

- Infrastructure-as-Code (IaC) versions of a system, typically Terraform configuration files for v_A and v_B; and
- Source code fragments for the same two system versions.

and executes three phases:

1) **Version-level analysis**: obtain an architectural description for each version by examining the IaC and code using an LLM and an IaC rule catalog;
2) **Pattern- and quality-aware enrichment**: refine each description by injecting microservice patterns and quality-attribute dimensions derived from IaC and code;
3) **Architecture-aware diff and ADR synthesis**: contrast the enriched descriptions for v_A and v_B, cluster the resulting deltas into decision candidates, and instruct an LLM to generate a MADR-file for each cluster.

The final output is a collection of synthetic ADRs in JSON and markdown (MADR) format for each (v_A, v_B) pair.

The three phases (see Fig. 1) are detailed below:

a) Phase 1: LLM-based architecture analysis of two deployment artifacts, corresponding to two consecutive versions v_A and v_B. If $v_A = t_0$, then only a single analysis on v_B is performed. The choice of deployment (IaC) artifacts as inputs is motivated by previous research suggesting that IaC (e.g., *Terraform* scripts) is a primary source for cloud architectures, in which defects and low-quality scripts often lead to outages, security vulnerabilities, and configuration drift [17], [18]. Furthermore, architectural knowledge management studies suggest that many decisions and their quality consequences are implicit in code and deployment artifacts [16], favoring IaC as a relevant artifact for decision mining.

Two examples of IaC versions (*Terraform* scripts) given as pipeline inputs are shown in Fig. 2. The analysis of the IaC artifacts uses a RAG-based strategy, where knowledge about architectural decisions and design rationale, relevant quality attributes, and microservice decomposition patterns is provided as an external knowledge base [19], [20]. The results of the RAG analysis for the running examples are summarized in Fig. 3, using word clouds for presentation purposes. Note that the actual analyses are textual descriptions in the pipeline.

b) Phase 2: Augments the previous analyses with information about microservices-related patterns, which are codified as LLM prompts. The source code of the input versions is provided as an additional information source. The patterns are expressed as rules and scoring heuristics that map IaC elements (i.e., *Terraform/OpenTofu* resources) and source code fragments to architectural dimensions (e.g., modularity, independent services, asynchronous communication, and distributed deployment). The improved analyses for the examples

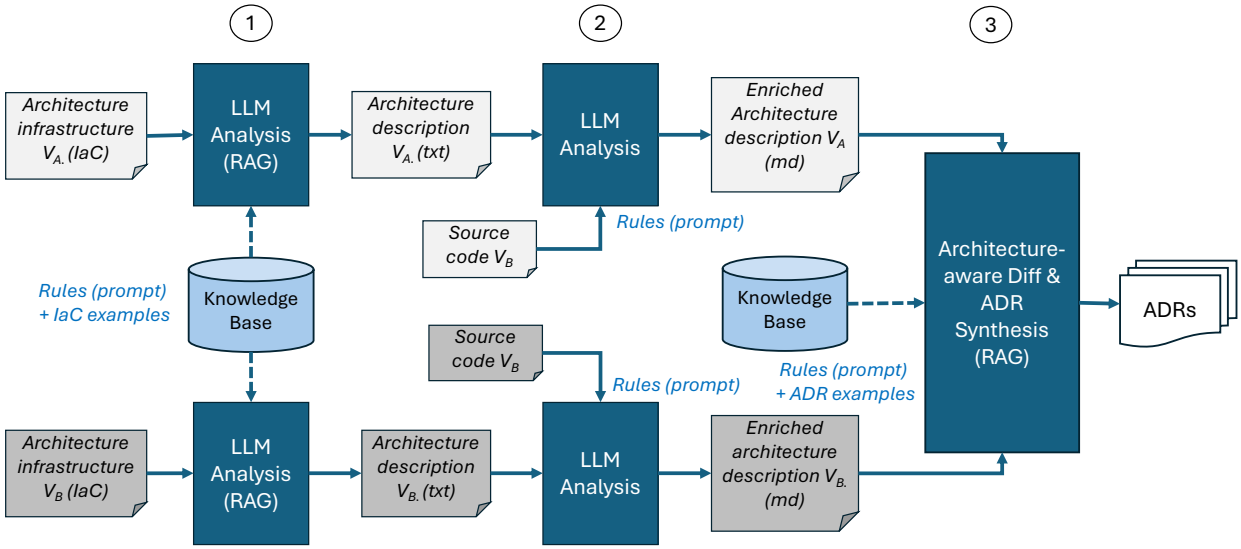
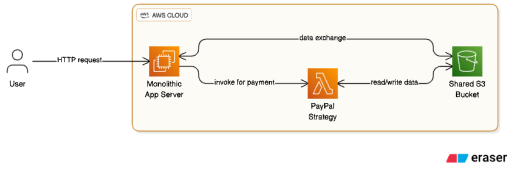
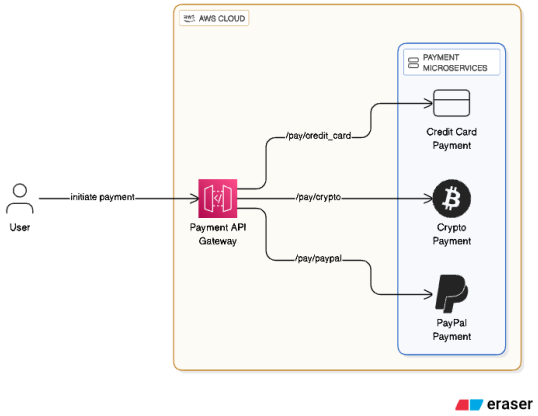


Fig. 1: Overview of the ADR-synthesis pipeline from IaC and source code artifacts.



(a) Monolith IaC (v_A)



(b) Microservices IaC (v_B)

Fig. 2: Example of IaC input artifacts (phase 1).

(see Fig. 4) shows that the first cloud is basically an inventory of infrastructure components detected in *Terraform*; while the second one includes code and pattern rules, "levels up" and begins to address domain (payments), migration from monolith to microservices, and design strategies and patterns. This is precisely the difference between seeing only the deployment picture vs. examining the architectural decisions behind it. Pattern matches and quality attribute trade-offs identified in

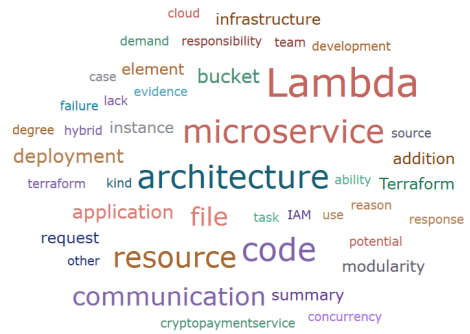


Fig. 3: Word clouds summarizing the RAG-based analyses of the input artifacts (phases 1 and 2).

this phase are useful downstream to ground the synthesized ADRs on them.

c) Phase 3: Performs an *architecture-aware diff* between the two system version models built in the previous phases. This is the core of the approach: instruct an LLM to generate architecture decisions, exploiting the architecture differences between the two system versions.

Besides a specialized prompt, the synthesis of architecture decisions is aided by examples (few-shots) of ADRs from the literature. The generated ADRs are the main output of the pipeline and adhere to the MADR template[2]. Specifically, for each IaC/code evolution pair, the pipeline produces a collection of ADRs in both JSON and MADR, with fields TITLE, STATUS, MOTIVATION, DECISION DRIVERS, MAIN DECISION, ALTERNATIVES, PROS, CONS, CONSEQUENCES, VALIDATION and ADDITIONAL INFORMATION. These ADRs are complete documents (see for example Fig.6) human-readable line by line, rather than intermediate machine-readable representations.

[2]https://adr.github.io/madr/

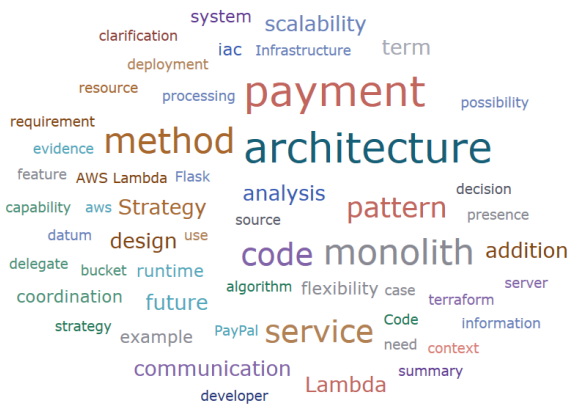


Fig. 4: Based on the project's IaC and source-code artifacts, each panel presents the most common architectural elements.

The pipeline is implemented with a Python toolchain. *Terraform* files are parsed into resource graphs, enriched with architectural dimensions using the IAC.TXT rule catalog, and then embedded for retrieval. For each (v_A, v_B) pair, the tool computes dimension-level deltas, groups them into decision clusters, and invokes the LLM with a MADR-centric prompt. The resulting ADRs are <REPO>_ADR_COLLECTION.JSON and Markdown MADR files, and can be analyzed with the metrics described in Section IV-C.

IV. Evaluation

The approach was evaluated with three research questions and three case studies:

- **RQ1** - *Decision coverage*: To what extent do synthetic ADRs, generated from IaC and source code changes, can recover the architectural decisions documented in the original ADRs of a system?
- **RQ2** - *ADR quality*: How does the structural quality of synthetic ADRs (section coverage and alternatives) compare to that of the original ADRs?
- **RQ3** - *Degree of grounding:* How consistent are synthetic ADRs with the actual evolution of the underlying infrastructure and code?

A. Subjects and Dataset

The ADR-Study dataset[3] contains 921 open-source repositories that reportedly include architectural decision records. Using mining scripts, the repositories were classified into three groups : (i) with ADRs but no *Terraform* files (194 repos), (ii) with *Terraform* files but no ADRs (58 repos), and (iii) with both ADRs and *Terraform* files (14 repos).

This study focused on the third group (i.e. with ADRs and *Terraform*) to be able to compare human-written ADRs with synthetic ADRs generated using IaC evolution data. Each repository was assessed regarding (i) non-trivial *Terraform* evolution, (ii) ADR files in markdown format (MADR), and (iii) different architectural styles (e.g. serverless, container-based, and cluster-based deployments). These inclusion criteria led to selecting three projects, namely: *Abelaa*, *Serverlessmike*, and *Chef*. The term cloud in Fig. 3 shows the most common architectural themes found by the LLM for each repository: *Abelaa* focuses on API gateway and serverless issues, *Serverlessmike* on container and messaging, and *Chef* on container orchestration and deployment automation, which matches the architectural philosophies.

For each repository, two IaC versions were identified (labeled as *minor* and *major*) in the evolution in modular projects.

B. Generated ADR artifacts

For each repository and IaC evolution pair minor/major, the pipeline outputs a set of ADRs in JSON and MADR formats.

For each case study, the pipeline generated five[4] synthetic ADRs. The core sections Motivation, Main Decision, Alternatives and Consequences of each synthetic ADR contain on average $L \approx 148$ tokens of text (Abelaa and Serverlessmike: 162.2 tokens on average; Chef: 118.8 tokens each). This length is comparable to the human-written ADRs found in ADR.MD, and confirms that the generated artifacts are full-fledged documents rather than short hints or labels.

Fig. 6 shows a synthetic ADR generated for the *Abelaa* repository: a migration decision at the architectural level, explicitly stating the motivation, main choice, alternatives, and expected consequences. This is indeed the target level of granularity: documents that can be read, reviewed, and stored alongside human-written ADRs as part of architecture knowledge base.

The number of tokens per section is a surrogate for expressed logic. Fig. 5 shows the token distribution in six sections (Motivation, Main Decision, Alternatives, Consequences, Pros, and Cons) for all 15 synthetic ADRs, and it makes explicit two architecture-relevant aspects: (1) in all fundamental portions, the pipeline creates non-trivial language, not labels or single-sentence "decisions"; and (2) later ADRs are fundamentally weaker, as seen in real projects where initial choices are well documented but later ones becomes thinner. Our rightmost bars (after ADR index) are shorter and feature fewer alternatives and consequences contributions.

The pipeline generates complete MADR-style documents: all ADRs contain non-trivial text in the core sections, and most of them also populate the Pros and Cons sections with additional rationale.

a) Synthetic ADRs: The pipeline takes the *Terraform* minor and major versions, plus the corresponding source code, and generates a collection of synthetic ADRs in JSON format (<REPO>_ADR_COLLECTION.JSON). Each synthetic ADR follows an extended MADR-template with sections Title, Status, Motivation, Decision Drivers, Main Decision, Alternatives, Pros, Cons, Consequences, Validation and Additional Information. This study

[3]https://github.com/software-competence-center-hagenberg/ADR-Study-Dataset/tree/main

[4]This was a mere coincidence; the decisions per project are not mandated.

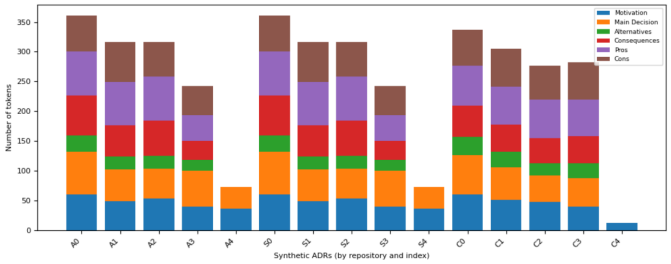


Fig. 5: Tokens per section (Motivation, Main Decision, Alternatives, Consequences, Pros, Cons) for the 15 synthetic ADRs.

TABLE I: Field mapping between MADR template and ADRs.

MADR field	Synthetic ADR field(s)
Title	TITLE
Status	STATUS
Context / Problem	MOTIVATION
Decision	MAIN DECISION
Consequences	CONSEQUENCES
Considered options	ALTERNATIVES
Pros / Cons per option	PROS, CONS
Validation / Notes	VALIDATION, ADD. INFORMATION

did a quantitative analysis on the core sections MOTIVATION, MAIN DECISION, ALTERNATIVES and CONSEQUENCES.

b) Pipeline configuration: The experiments used the OpenAI GPT-4O conversational model as LLM for all pipeline phases. Phases 1 and 2 called it with a low temperature ($T = 0.2$) and a maximum of 1 200 output tokens to promote consistent, low-variance architecture descriptions. Phase 3 utilized a slightly higher temperature ($T = 0.3$) and 2 000 output tokens for richer reasoning and control of variability. To determine IaC semantics, the IAC.TXT rule catalogue was applied, which maps *Terraform* resources to higher-level architectural aspects like modularity, messaging, and redundancy. OpenAI's TEXT-EMBEDDING-3-LARGE model embeds code and IaC fragments, which are then employed in the analysis and alignment scripts for decision coverage and grounding calculations. *This pipeline generated all synthetic ADRs in this study, with no LLM fine-tuning nor manual post-editing.*

C. Quantitative Evaluation of Address RQs

Instead of using sentences or keyword lists, the generated ADRs were evaluated using metrics over whole documents (original and synthetic), using MADR sections. Four metrics were considered to measure human and synthetic ADRs:

- **MADR Section Coverage (MSC)**: fraction of core sections {MOTIVATION, MAIN DECISION, ALTERNATIVES, CONSEQUENCES} that are non-empty and have at least N tokens of substantive text (the experiments used $N = 20$).
- **Alternatives Presence & Richness (APR)**: number of distinct alternatives (capped at 2) in the ALTERNATIVES section, normalised to $[0, 1]$.
- **Dimension Consistency ($\mathbf{DC_{avg}}$)**: average consistency between ADR claims and the evolution observed in IaC and code, over a set of architectural dimensions (e.g., modularity, redundancy, backup strategy).

```
# ADR: Deployment Model with AWS Lambda
## Title
Deployment Model with AWS Lambda
## Status
Accepted
## Motivation
The existing deployment model is centralized
and lacks flexibility, limiting the ability
to scale and adapt to changing demands.
Adopting a serverless deployment model with
AWS Lambda will enable more efficient resource
utilization and scalability.
## Decision Drivers
- Need for scalable and cost-effective deployment.
- Requirement for flexible resource allocation based on
    demand.
- Desire to reduce operational overhead and complexity.
## Main Decision
The decision is to deploy specific microservices as AWS
    Lambda functions, leveraging the serverless model for
    event-driven processing and independent scaling. This
    approach will reduce infrastructure management
    overhead and allow services to scale automatically in
    response to demand, improving cost efficiency and
    performance.
## Alternatives
- Traditional VM-based deployment on EC2 instances.
- Containerized deployment using ECS or Kubernetes.
## Pros
- Main decision:
  - Pros:
    - Reduces operational overhead with automatic scaling.
    - Enhances cost efficiency by paying only for used
        resources.
    - Simplifies deployment and management with serverless
        infrastructure.
- Traditional VM-based:
  - Pros:
    - Provides full control over the environment and
        resources.
    - Familiar and well-understood deployment model.
- Containerized deployment:
  - Pros:
    - Offers flexibility in resource allocation and scaling
        .
    - Supports consistent environments across development
        and production.
## Cons
- Main decision:
  - Cons:
    - Limited control over execution environment.
    - Potential cold start latency for infrequently used
        functions.
- Traditional VM-based:
  - Cons:
    - Higher operational overhead and complexity.
    - Limited scalability and flexibility.
- Containerized deployment:
  - Cons:
    - Requires container orchestration and management.
    - Potentially higher complexity in setup and operation.
## Consequences
Deploying services as AWS Lambda functions will improve
    scalability and cost efficiency while reducing
    operational complexity. However, it may introduce
    challenges related to execution environment control
    and cold start latency, which need to be managed
    effectively.
## Validation
Validation to be defined in future iterations.
## Additional Information
- Related ADRs: Transition to Microservices Architecture,
    Communication Style.
```

Fig. 6: Example of generated ADR.

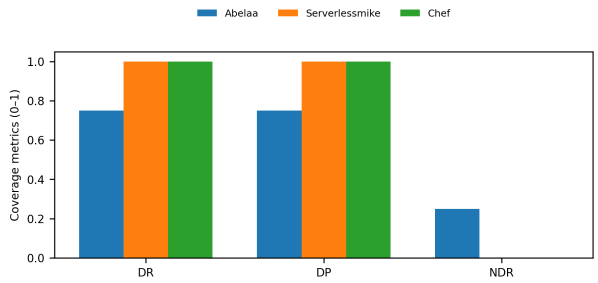


Fig. 7: Decision Recall (DR), Decision Precision (DP) and New Decision Rate (NDR) for synthetic ADRs in the cases.

- **Evidence Link Ratio (ELR)**: ratio between ADR claims that can be explicitly grounded in IaC or code and the total number of claims in the ADR.

D. RQ1: Decision Coverage vs. Original ADRs

To assess coverage, the original and synthetic ADRs are compared at the *decision* level. Sentence embeddings are computed over ADR titles and decision sections, yielding top-k candidate matches for each original ADR. This alignment allows to compute:

- **ADR Recall**: proportion of original ADRs that have at least one equivalent synthetic ADR.
- **ADR Precision**: proportion of synthetic ADRs that correctly correspond to some original ADR.

Synthetic ADRs with no equivalent in the ground truth are also counted as *new decision candidates*.

Expert annotation shows good 2-system coverage and partial 1-system coverage. *Serverlessmike* and *Chef* achieved flawless Decision Recall (DR = 1.0), Decision Precision (DP = 1.0), and New Decision Rate (0.0), indicating that all synthetic ADRs were matched to human ADRs and no new decisions were made. *Abelaa*, in turn, had strong but imperfect agreement (DR and DP 0.75, NDR 0.25), indicating that most synthetic ADRs mirror human choices while a small percentage are new. In some systems, the pipeline can reliably recover human-documented decisions, but coverage depends on decision granularity and equivalence operationalization, especially in *Abelaa*, where synthetic ADRs may capture broader or differently scoped architectural interpretations.

E. RQ2: ADR Structural Quality (MSC and APR)

Fig. 6 shows an actual, typical ADR generated by the pipeline. Synthetic ADRs had MSC values comparable to human-written ADRs in all three cases (see Fig. 8). The average MSC for human ADRs is *38%*, while synthetic ADRs reached *75%* with similar variance. This suggest that the pipeline can populate the core parts (Motivation, Main Decision, Alternatives, Consequences) with a structural completeness comparable to project teams. Higher coverage does not necessarily mean deeper or more accurate design reasoning, because MSC is a structural metric.

A word cloud was generated for each case study, with core ADR components (TITLE, MOTIVATION, PRIMARY CHOICE, ALTERNATIVES, and REPERCUSSIONS). The key terms include MIGRATION TO MICROSERVICES, COMMUNICATION STYLE, DEPLOYMENT STRATEGY, and DATA MANAGEMENT, confirming that the generated ADRs focus on architecturally relevant decisions rather than on low-level setup details.

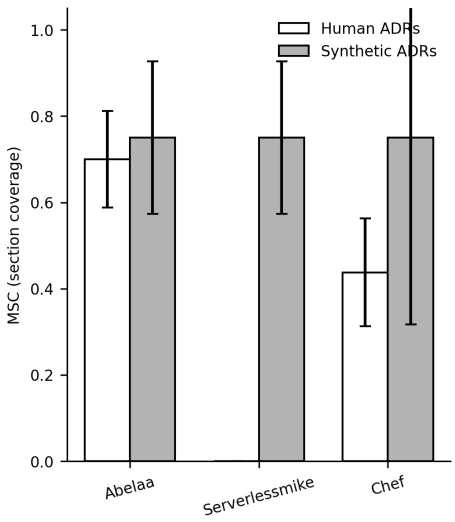


Fig. 8: Average MADR Section Coverage (MSC) for in-scope human and synthetic ADRs in the three case studies. Error bars show standard deviation.

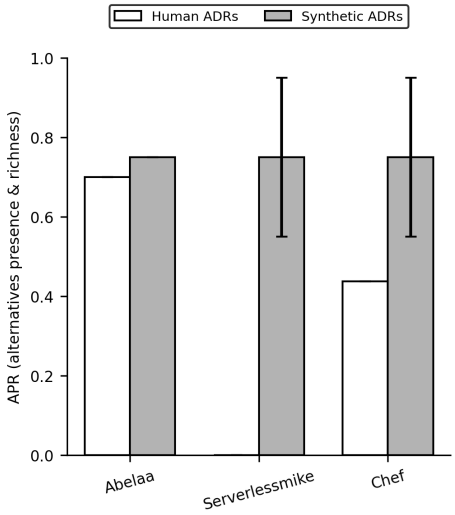


Fig. 9: Alternatives Presence & Richness (APR) for in-scope human and synthetic ADRs. Error bars show standard dev.

Alternatives Presence & Richness (APR) was calculated for each ADR (see Fig. 9), capped at two alternatives (MAX_ALTS_FOR_APR = 2), with APR = 0, 0.5, and 1.0 indicating 0, 1, and at least 2 explicit alternatives, respectively.

Synthetic ADRs had a larger APR than human ADRs in *Abelaa* and *Serverlessmike* (e.g., 0.75 vs. 0.50 on average), but for *Chef* it was identical. This suggests that the pipeline makes alternatives explicit more systematically than the original

TABLE II: Dimensions used to calculate DC_avg.

Dimension	Interpretation
load_balancer	Load balancer / HTTP ingress
api_gateway	API gateway and management
serverless_functions	Serverless compute (Lambda/FaaS)
containers	Container platform (ECS/EKS/Docker)
compute_instances	VM/instance-based compute (EC2)
vpc_networking	VPC, subnets, and routing
dns_routing	DNS and traffic routing (Route53)
messaging_queues_events	Messaging and eventing (SQS/SNS/EventBridge)
caching_cdn	Caching and CDN (CloudFront/ElastiCache)
monitoring_logging	Monitoring and logging (CloudWatch/observability)
postgresql_cluster	Relational database cluster (RDS/PostgreSQL)
nosql_store	NoSQL store (DynamoDB)
opensearch_cluster	Search/analytics cluster (OpenSearch/Elasticsearch)
s3_backups	Object storage and backup strategy (S3/snapshots)
terraform_state_manage..	Terraform state/backend management
multi_region_environ..	Multi-region / disaster recovery environment
security_iam	IAM security (roles/policies)
security_network	Network security (security groups/rules)
security_ops_tooling	Security ops tooling (SSM/Inspector)

documentation. However, it does not prove that all generated alternatives are equally grounded or were examined by the team. APR captures the recorded decision space, but not the plausibility or evidence of each alternative, which we handle independently using grounding metrics and expert judgment.

Each ADR (human-written and synthetic) was measured with two metrics:

- **MSC** (MADR Section Coverage): fraction of the core sections {MOTIVATION, MAIN DECISION, ALTERNATIVES, CONSEQUENCES} that are non-empty and have at least N tokens of substantive text.
- **APR** (Alternatives Presence & Richness): number of distinct alternatives (capped at 2) in the ALTERNATIVES section, normalised to $[0, 1]$.

Some auxiliary sections (PROS, CONS, VALIDATION and ADDITIONAL INFORMATION) were excluded from MSC and APR to avoid rewarding generic or hallucinated boilerplate.

F. RQ3: Grounding degree in IaC and Code (DC_{avg} and ELR)

Evaluating whether ADR statements are grounded in the actual infrastructure and source code was done with two steps:

1) Identify claims that mention architectural properties or components (e.g., load balancers, OpenSearch cluster, PostgreSQL cluster, backup mechanisms).
2) Check whether these claims are consistent with the evolution from <REPO>_CLOUD_EVOLUCION_MENOR.TF to <REPO>_CLOUD_EVOLUCION_MAYOR.TF and with the corresponding source code.

DC_{avg} is the average normalized consistency across the architectural dimensions extracted from IaC (see Table II), and ELR is the ratio between grounded claims and all claims in the ADR (see Fig.10).

Grounding profiles of synthetic ADRs differ among the three cases (see Fig. 10). *Abelaa* has low grounding across all measures (DC_claim_avg $\approx$ 0.10, ELR_any $\approx$ 0.10, ELR_all $\approx$ 0.10), indicating that only a limited portion

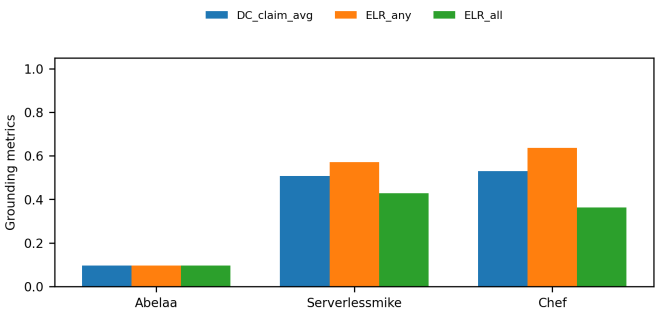


Fig. 10: Claim-level grounding metrics (DC_claim_avg, ELR_any y ELR_all) for synthetic ADRs.

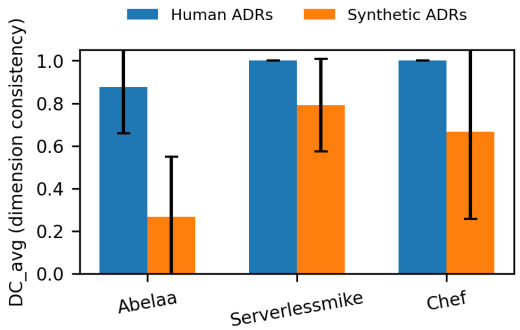


Fig. 11: Average DC_avg for in-scope human and synthetic ADRs. Error bars show standard deviation.

of synthetic claims can be linked to Terraform evolution's architectural dimensions. This suggests that generated ADRs may prioritize idealized microservice/serverless patterns (e.g., API Gateway, Lambda-related decisions) over IaC infrastructure. *Serverlessmike* has moderate grounding (DC_claim_avg $\approx$ 0.51, ELR_any $\approx$ 0.57, ELR_all $\approx$ 0.43), with almost half of claims supported in IaC dimensions including messaging, security, and compute. *Chef* has a moderate profile (DC_claim_avg $\approx$ 0.53, ELR_any $\approx$ 0.63), but a lower ELR_all ($\approx$ 0.36), showing that IaC evidence partially supports many claims but fewer are firmly grounded across all dimensions. These results imply that the pipeline can yield plausibly grounded ADRs when Terraform directly reflects architectural decisions, but may still drift toward higher-level or idealized patterns when the IaC evolution provides insufficient or inconsistent claim set evidence.

Human-built ADRs show (see Fig. 11) near-perfect grounding (DC_avg $\approx$ 0.9–1.0) in all three cases, but synthetic ADRs show greater fluctuation. Synthetic ADRs in *Abelaa* are less consistent with Terraform progression (DC_avg $\approx$ 0.15), but in *Serverlessmike* and *Chef*, they capture most implemented dimensions (DC_avg 0.67 to 0.89). The pipeline can yield well-grounded ADRs when IaC evolution is regular and linked with decisions, but it may falter in projects using higher-level or implicit architectural language. DC_avg enhances MSC and APR by identifying instances where infrastructure changes

weakly support structurally rich ADRs.

V. Threats to Validity

Some threats to study validity were identified and mitigated.

A. Internal validity

Internal validity concerns whether the pipeline's results are caused by it or by confounding circumstances.

The original ADRs in ADR.MD was the "ground truth" for documented architectural decisions, but they might be incomplete, outdated, or biased toward team decisions. Coverage metrics are sensitive to human ADR granularity and how they operationalize "equivalence" across decision layers. The updated coverage findings show that Serverlessmike and Chef have perfect agreement within the scope (DR = 1.0, DP = 1.0, NDR = 0.0), while Abelaa has high but partial alignment (DR = 0.75, DP = 0.75, NDR = 0.25). Hence, the pipeline can successfully recover documented decisions in some systems, but may introduce a narrower set of higher-level microservice/cloud interpretations in others.

Establishing equivalence between human and synthetic ADRs has two steps: embedding-based retrieval of top-k candidate pairs, and hand annotation. The embedding model and similarity thresholds may miss semantically-related decisions (e.g., "on-prem vs. SaaS" vs. "migration to microservices"). Annotators may use different granularity when deciding if two ADRs represent the same decision. Conflicts were resolved by conversation, but inter-rater agreement was not recorded.

IaC-based analyses and metrics use custom rules and dimension mappings (e.g., DIMENSIONS). The updated grounding profiles show that Abelaa has low claim grounding across metrics, while Serverlessmike and Chef have moderate grounding with a significant drop in ELR_all, indicating partial evidence alignment rather than full multi-dimension support. ADR claims based on Terraform alone without architectural dimensions were considered ungrounded, even if they mirrored the infrastructure.

B. External vality

External validity concerns whether the findings are generalizable to other systems, technologies, and locations.

Three open-source systems from the ADR-Study dataset were chosen for empirical evaluation, given their non-trivial Terraform evolution and diverse deployment styles (serverless, container-based, and cluster-based). However, this sample may not represent industrial systems with heterogeneous IaC, sophisticated ADR processes, or tougher compliance requirements for architectural decisions.

Further, the study considers a single IaC technology (Terraform) and a specific source code co-evolution method. Implementing the rule catalog (IAC.TXT) in cloud settings controlled by other tools or mixed on-prem/cloud configurations may require significant change due to differing architecture signals. The pipeline has been tested on microservice-oriented and cloud-native systems, but monolithic or embedded systems with limited IaC support might differ.

This study had a fixed LLM configuration (model family, temperature, prompt templates, few-shot samples). Other models, languages, or organizational contexts (e.g., highly regulated areas with greater privacy constraints) might yield different coverage, structural quality, and grounding.

C. Construct validity

Construct validity concerns whether metrics and instruments accurately measure the studied notions.

ADR structural quality was measured using MADR Section Coverage (MSC) and Alternatives Presence & Richness (APR). ADRs with non-empty core sections and at least N tokens of text are rewarded by MSC, but this does not guarantee valid, precise, or architecturally deep content. Verbose but superficial explanations may outperform concise but well-argued decisions. APR counts different alternatives up to a cap, without distinguishing between realistic IaC/code possibilities and speculative or generic ones. This is partially addressed by analyzing grounding independently (DC_{avg}, DC_{claim_avg}, ELR), but MSC and APR alone do not indicate quality in terms of superior design rationale.

DR and DP use candidate pairs from the embedding model and the human idea of "equivalent decision" for decision coverage. Based on the updated results, this operationalization can generate near-perfect alignment in Serverlessmike and Chef and partial alignment in Abelaa. Some practitioners see "on-prem vs. SaaS" and "migration to microservices" as distinct strategic issues. Thus, the operationalization of equivalence is imprecise, and depends on decision scope, documentation style, and domain framing.

Dimensions were defined using keyword heuristics on ADR text and IaC resources, and a claim is grounded if at least one dimension it mentions is in IaC evolution. New grounding results indicate that the instruments capture partial evidence alignment (ELR_any) better than complete multi-dimension grounding (ELR_all), which may underestimate correctness when decisions span multiple IaC signals or overestimate it when keywords align superficially.

VI. Conclusions

The LLM-based approach discussed in this paper is able to recover architectural decisions from source code and deployment IaC information for two consecutive versions of a microservice-based system. The method generates reviewable ADRs from source code and IaC deployment data. The proposed pipeline integrates domain knowledge with pretrained knowledge in two stages using RAG and LLM. These stages involve: (1) generation of TEXTUAL architecture descriptions from initial and subsequent system artifacts, analyzing deployment and source code with a design rules catalog, and (2) architecture-aware diff between these two descriptions, synthesizing possible ADRs.

The approach was tested with three existing GitHub projects that contained source code, IaC files and ADRs. Our quantitative evaluation of synthesized ADRs did not include statistical testing or power analysis, therefore differences between human

and synthetic ADRs or systems should be considered trends rather than statistically validated effects.

The generated ADRs represent architectural migration decisions, including motive, main choice, alternatives, and expected consequences with a reasonable accuracy. We believe that these ADRs can be useful for software architects needing artifacts that can be read, approved, and saved alongside human-built ADRs in a project knowledge base.

Initial results indicate that the approach recovers most ADR choices and produces additional candidate judgments that practitioners found useful and accurate. The human-written and synthesized ADRs are similarity formatted in JSON and Markdown MADR, making the process clear. To reduce architectural knowledge vaporization in changing systems, our approach for decision mining from code and deployment artifacts seems promising.

References

[1] P. Cruz, H. Astudillo, and L. Salinas, "Quick evaluation of a software architecture using the decision centric architecture review," in *Proc. ECSA Companion*, 2020.

[2] G. Buchgeher, S. Schöberl, V. Geist, B. Dorninger, P. Haindl, and R. Weinreich, "Using architecture decision records in open source projects — an MSR study on GitHub," *IEEE Access*, vol. 11, pp. 63 725–63 740, 2023. DOI: 10.1109/ACCESS.2023.3287654.

[3] S. Dasanayake, J. Markkula, S. Aaramaa, and M. Oivo, "Software architecture decision-making practices and challenges: An industrial case study," in *2015 24th Australasian Software Engineering Conference (ASWEC)*, IEEE, 2015. DOI: 10.1109/ASWEC.2015.20.

[4] R. Singh, A. Yeboah-Ofori, S. Kumar, and A. Ganiyu, "Fortifying cloud devsecops security using terraform infrastructure as code analysis tools," in *Proc. ICECER*, 2024.

[5] A. Mittal and V. Venkatesan, "Practical integration of large language models into enterprise ci/cd pipelines for security policy validation: An industry-focused evaluation," in *2025 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, IEEE, 2025.

[6] O. Kopp, A. Armbruster, and O. Zimmermann, "Markdown architectural decision records: Format and tool support," in *ZEUS*, 2018.

[7] R. F. Elcullada-Encarnacion, "Academic advising system using data mining method for decision making support," in *2018 4th International Conference on Computer and Technology Applications (ICCTA)*, IEEE, 2018, ISBN: 978-1-5386-6995-2.

[8] A. Jansen, J. van der Ven, P. Avgeriou, and D. K. Hammer, "Tool support for architectural decisions," in *2007 Working IEEE/IFIP Conference on Software Architecture (WICSA'07)*, IEEE, Jan. 2007, pp. 4–4. DOI: 10.1109/wicsa.2007.47.

[9] A. Jansen, "Architectural design decisions," Ph.D. dissertation, University of Groningen, 2008.

[10] D. Dermeval et al., "STREAM-ADD - supporting the documentation of architectural design decisions in an architecture derivation process," in *2012 IEEE 36th Annual Computer Software and Applications Conference*, 2012. DOI: 10.1109/COMPSAC.2012.81.

[11] C. López, V. Codocedo, H. Astudillo, and L. M. Cysneiros, "Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach," *Science of Computer Programming*, vol. 77, no. 1, pp. 66–80, Jan. 2012, ISSN: 0167-6423. DOI: 10.1016/j.scico.2010.06.009.

[12] H. Astudillo, G. Valdes, and C. Becerra, "Empirical measurement of automated recovery of design decisions and structure," in *2012 IEEE VI Andean Region International Conference*, IEEE, Nov. 2012, pp. 105–108. DOI: 10.1109/andescon.2012.33.

[13] A. Shahbazian, Y. K. Lee, D. Le, Y. Brun, and N. Medvidović, "Recovering architectural design decisions," in *International Conference on Software Architecture (ICSA)*, 2018. DOI: 10.1109/ICSA.2018.00019.

[14] M. Bhat, C. Tinnes, K. Shumaiev, A. Biesdorf, U. Hohenstein, and F. Matthes, "ADeX: A tool for automatic curation of design decision knowledge for architectural decision recommendations," in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, IEEE, Mar. 2019. DOI: 10.1109/icsa-c.2019.00035.

[15] R. Dhar, K. Vaidhyanathan, and V. Varma, "Can LLMs generate architectural design decisions? - an exploratory empirical study," in *2024 IEEE 21st International Conference on Software Architecture (ICSA)*, IEEE, Jun. 2024, pp. 79–89. DOI: 10.1109/icsa59870.2024.00016.

[16] R. Dhar, A. Kakran, A. Karan, K. Vaidhyanathan, and V. Varma, "DRAFT-ing architectural design decisions using LLMs," arXiv preprint server, Tech. Rep., 2025. DOI: 10.48550/arXiv.2504.08207.

[17] M. M. Hasan, F. A. Bhuiyan, and A. Rahman, "Testing practices for infrastructure as code," in *Proceedings of the 1st ACM SIGSOFT International Workshop on Languages and Tools for Next-Generation Testing (LANGETI'20)*, ACM, 2020, pp. 7–12. DOI: 10.1145/3416504.3424334.

[18] W. Begoug, M. Chouchen, and A. Ouni, "Terrametrics: An open source tool for infrastructure-as-code (iac) quality metrics in terraform," in *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension (ICPC)*, ACM, 2024. DOI: 10.1145/3643737.3644461.

[19] X. Zhou et al., "Using llms in generating design rationale for software architecture decisions," arXiv preprint server, Tech. Rep., 2025. DOI: 10.48550/arXiv.2504.20781.

[20] ADR-Synth-Trace Project, "ADR-Synth-Trace: Trazabilidad verificable de decisiones arquitectónicas," Unpublished manuscript, 2025.