

Methods for reconstructing microservice architectures: A thematic literature review

Olena Grinenko^{1,†}, Andrii Gizun^{1,†}, Artem Morhun^{2,*,†} and Andrii Bekh^{3,†}

¹Taras Shevchenko National University of Kyiv, Volodymyrska Str., 64/13, Kyiv, 01601, Ukraine

²LLC Grammarly Ukraine, Sportyvna Square, 1a, Kyiv, 01001, Ukraine

³Technology Oriented S.R.L., Kyiv, Ukraine

Abstract

This review examines how microservice architectures are reconstructed from existing systems, grouping methods by evidence source into static, dynamic, and hybrid reconstruction while treating model-driven transformation as a cross-cutting strategy. Static approaches draw on source code, annotations, build files, deployment artefacts, or bytecode to recover service dependencies, communication structures, bounded contexts, and context maps. Dynamic approaches work from execution evidence such as traces, access logs, and telemetry to infer runtime dependencies and observed communication paths. Hybrid approaches combine static and runtime sources, whereas model-driven techniques transform recovered evidence into explicit architectural representations. No single artefact source is sufficient. Static reconstruction can be automated and fits development pipelines, but may miss runtime behaviour, indirect calls, and workload-dependent interactions; dynamic reconstruction reflects observed behaviour, yet its quality hinges on instrumentation, workload coverage, and representative data; and hybrid reconstruction can improve coverage but remains sensitive to platform and tooling support, and validation. Model-driven techniques make recovered evidence explicit and transformable, but their accuracy depends on the supported metamodels and transformation rules. Intermediate representations recur throughout, letting the same extracted information be rendered as dependency graphs, 2D diagrams, AR/3D visualisations, or standardised topology models. Persistent limitations remain: language and framework dependence, ambiguous endpoint matching, incomplete runtime coverage, and the absence of a shared intermediate representation across the reviewed methods. Static reconstruction provides a practical baseline, dynamic reconstruction captures observed behaviour, hybrid reconstruction can broaden coverage, and model-driven techniques broaden the available representations.

Keywords

microservices, software architecture reconstruction, architecture recovery, static analysis, dynamic analysis, model-driven engineering, literature review

1. Introduction

Microservice architecture has become a widely adopted style for cloud-native software because it supports decentralised development, independent deployment, and selective scaling [1]. Yet this flexibility carries a cost. In long-lived systems, the architecture fragments: knowledge about services, their dependencies, and their runtime communication is scattered across source repositories, build and configuration files, container descriptors, and monitoring platforms, with the remainder recorded, if at all, only in team documentation. As the system evolves, this picture drifts out of date. Reasoning about how services depend on one another, where the architecture has eroded, and what a proposed change will actually affect becomes progressively harder. These blind spots also have security and operational consequences because analyses of attack paths, trust boundaries, and implemented security mechanisms depend on an accurate architectural view. Architectural recovery can therefore support cyber-hygiene activities in addition to software documentation [2, 3].

WCH'26: Fifth International Workshop on Cyber Hygiene, June 25–26, 2026, Krakow, Poland

*Corresponding author.

†These authors contributed equally.

✉ gsa_ck@ukr.net (O. Grinenko); andriy.gizun@gmail.com (A. Gizun); artem-ua@outlook.com (A. Morhun); andrii.bekh@gmail.com (A. Bekh)

ORCID 0000-0001-9673-6626 (O. Grinenko); 0000-0002-2974-6987 (A. Gizun); 0009-0002-0027-1225 (A. Morhun); 0009-0003-0426-1136 (A. Bekh)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

Software architecture reconstruction (SAR) addresses this problem by deriving higher-level views from lower-level evidence. For microservice systems, this means recovering service dependency graphs, communication diagrams, context maps, and topology models from artefacts such as source code, deployment descriptors, logs, traces, and model-based representations [4–6]. The difficulty is that no single source captures the whole architecture. Source code and static configuration expose the declared structure, including endpoints, data models, and deployment assumptions, yet they may not reveal all indirect calls, dynamically resolved dependencies, and behaviour that surfaces only under particular workloads. Runtime telemetry and traces show the opposite: genuine observed interactions, but only for deployed services and execution paths that are actually exercised. Hybrid approaches bridge this gap by combining complementary sources [5, 7]. Model-driven techniques address a related but distinct problem by structuring recovered evidence as intermediate representations that support analysis, visualisation, smell detection, and architecture evolution [8].

The literature on the topic is expanding, yet it remains fragmented. Studies differ in the artefacts they consume, the techniques they apply, the assumptions their tools make, and the views they output. Some concentrate on source-code and bytecode analysis, others on logs and traces, others again on deployment or topology descriptors or on language-agnostic representations meant to generalise across technology stacks [2, 9, 10]. Two secondary studies are the most closely related to ours. Cerny et al. [11] review reconstruction and visualisation techniques, while Bakhtin et al. [10] classify tools by their inputs and outputs. This review adopts a different organising principle. It groups the field by the *source of architectural evidence* and compares static, dynamic, and hybrid reconstruction across their inputs, recovered views, strengths, and limitations, while examining model-driven transformation as a cross-cutting technique.

The aim is therefore not an exhaustive catalogue of tools but a source-centric synthesis that makes the complementarity of the three families explicit and highlights gaps that recur across the literature: limited benchmarking, reliance on specific languages and frameworks, difficulty in fusing heterogeneous evidence, incomplete runtime coverage, and the lack of a shared intermediate representation across the reconstruction methods reviewed here.

The review is guided by the following research questions:

- **RQ1.** Which methods are used to reconstruct microservice architectures?
- **RQ2.** Which input artefacts do these methods rely on?
- **RQ3.** Which architectural views do they recover as outputs?
- **RQ4.** Which strengths, limitations, and trade-offs shape their applicability?

The remainder of the paper is organised as follows. Section 2 describes the review design, including the search strategy, selection criteria, extraction, and synthesis. Section 3 presents the thematic findings for static, dynamic, and hybrid reconstruction and examines model-driven transformations across these families. Section 4 discusses the methodological and practical implications, Section 5 draws together the open gaps and directions for future work, and Section 6 concludes.

2. Methods

2.1. Review design

This study was designed as a thematic literature review supported by a structured search, screening, extraction, and synthesis process. The objective was to identify, classify, and compare research that proposes, evaluates, reviews, or compares methods for reconstructing microservice architectures from existing systems. The review was not intended to be an exhaustive PRISMA-style systematic review. Instead, it aimed to provide a transparent and thematically organised synthesis of the main methodological families used in the reconstruction of microservice architectures.

Software architecture reconstruction was used as the umbrella concept for recovering higher-level architectural representations from lower-level artefacts. In the context of microservices, reconstruction

was interpreted broadly to include the recovery of service dependency graphs, communication diagrams, context maps, topology models, and other architecture-oriented views. Four analytical dimensions guided the extraction and synthesis: (I) the reconstruction method, (II) the input artefacts used, (III) the architectural outputs produced, and (IV) the reported strengths and limitations of each approach.

2.2. Search strategy

The literature search covered Scopus, Web of Science, and Google Scholar. The main searches were conducted between December 2025 and March 2026, and a final update search was performed in April 2026 to identify additional relevant records. Scopus and Web of Science were used as the primary bibliographic databases because they support structured search fields, document-type, subject-area, and language filtering, as well as keyword-based refinement. Google Scholar was used as a complementary discovery source rather than as an exhaustive bibliographic database because its search results are not consistently reproducible and its support for long Boolean expressions is limited [12].

The search strategy combined three concept groups. The first group captured the application domain, using terms such as *microservice*, *microservices*, and *microservice architecture*. The second group captured the reconstruction objective, using terms such as *architecture reconstruction*, *architecture recovery*, *architecture mining*, and *software architecture reconstruction*. The third group captured methodological families identified during preliminary screening, including *static analysis*, *dynamic analysis*, *systematic mapping*, *visualisation*, *service dependency graph*, and *model-driven reconstruction*.

No hard publication-date filter was applied in the database queries. In practice, the retained evidence is concentrated in the period 2020–2025, which reflects the recent growth of microservice-architecture-reconstruction research. A small number of older studies were retained deliberately as conceptual anchors, including foundational SAR theory [4], early microservice-specific reconstruction work [5], and cloud-native background literature [1].

2.3. Scopus search query

The Scopus query was formulated using the TITLE-ABS-KEY field and then refined by exact keyword, document type, subject area, and language. The query was:

```
TITLE-ABS-KEY (
  (
    ( microservice* )
    AND
    (
      "architecture reconstruction"
      OR "architecture mining"
      OR "recovering the software architecture"
      OR "static analysis"
      OR "dynamic analysis"
      OR "systematic mapping"
    )
  )
  OR ( "software architecture reconstruction" )
)
AND (
  LIMIT-TO ( EXACTKEYWORD, "Microservice" )
  OR LIMIT-TO ( EXACTKEYWORD, "Architecture Reconstruction" )
  OR LIMIT-TO ( EXACTKEYWORD, "Static Analysis" )
  OR LIMIT-TO ( EXACTKEYWORD, "Microservices" )
  OR LIMIT-TO ( EXACTKEYWORD, "Software Architecture Reconstruction" )
  OR LIMIT-TO ( EXACTKEYWORD, "Software Architecture" )
)
```

```

OR LIMIT-TO ( EXACTKEYWORD, "Static Code Analysis" )
OR LIMIT-TO ( EXACTKEYWORD, "Software Design" )
OR LIMIT-TO ( EXACTKEYWORD, "Microservice Architecture" )
OR LIMIT-TO ( EXACTKEYWORD, "Visualization" )
OR LIMIT-TO ( EXACTKEYWORD, "Computer Architecture" )
OR LIMIT-TO ( EXACTKEYWORD, "Mapping" )
OR LIMIT-TO ( EXACTKEYWORD, "Native System" )
OR LIMIT-TO ( EXACTKEYWORD, "System Centric" )
OR LIMIT-TO ( EXACTKEYWORD, "System-centric View" )
)
AND (
  LIMIT-TO ( DOCTYPE, "re" )
  OR LIMIT-TO ( DOCTYPE, "cp" )
  OR LIMIT-TO ( DOCTYPE, "ar" )
)
AND (
  LIMIT-TO ( SUBJAREA, "ENGI" )
  OR LIMIT-TO ( SUBJAREA, "COMP" )
)
AND ( LIMIT-TO ( LANGUAGE, "English" ) )

```

After applying the stated filters, the Scopus search returned 243 records. The broad OR branch for *software architecture reconstruction* was retained to capture foundational SAR work, but records from this branch were included in the final synthesis only when they were directly relevant to microservice reconstruction or necessary for defining SAR concepts.

2.4. Web of Science search query

The Web of Science query was translated into Topic Search and refined by author keywords, Keywords Plus, research area, document type, and language. The query was:

```

TS=(
  (
    microservice*
    AND
    (
      "architecture reconstruction"
      OR "architecture mining"
      OR "recovering the software architecture"
      OR "static analysis"
      OR "dynamic analysis"
      OR "systematic mapping"
    )
  )
  OR "software architecture reconstruction"
)
AND (
  AK=(
    Microservice* OR "Architecture Reconstruction" OR "Static Analysis"
    OR "Software Architecture Reconstruction" OR "Software Architecture"
    OR "Static Code Analysis" OR "Software Design"
    OR "Microservice Architecture" OR Visualization OR Mapping
  )
)

```

```

    OR "System Centric" OR "System-centric View"
)
OR KP=(
    Microservice* OR "Architecture Reconstruction" OR "Static Analysis"
    OR "Software Architecture Reconstruction" OR "Software Architecture"
    OR "Static Code Analysis" OR "Software Design"
    OR "Microservice Architecture" OR Visualization OR Mapping
    OR "System Centric" OR "System-centric View"
)
)
AND SU=(Computer Science OR Engineering)

```

The Web of Science results were refined to English-language articles, proceedings papers, and review articles in relevant Computer Science and Engineering categories. After refinement, the Web of Science search returned 268 records.

2.5. Google Scholar search

Google Scholar was searched through shorter phrase-based queries rather than one long Boolean expression. The following queries were used:

- “software architecture reconstruction” microservices;
- “microservice architecture reconstruction” “static analysis”;
- “microservice architecture reconstruction” “dynamic analysis”;
- “recovering the software architecture” “microservice-based systems”;
- “microservice architecture” “service dependency graph” reconstruction;
- “microservices” “architecture reconstruction” visualization;
- “microservices” “architecture recovery” “model-driven”;
- “microservices” “architecture reconstruction” “systematic mapping”;
- “software architecture reconstruction” “GraalVM” microservice;
- “microservice telemetry” “software architecture reconstruction”.

For each Google Scholar query, the first 50 results were screened by title and snippet, followed by abstract and full text when necessary. Google Scholar was used to locate additional relevant studies, citation-linked papers, and accessible versions of indexed publications. It was not used to compute a reproducible total number of identified records.

2.6. Study selection criteria

Studies were included if they met at least one of the following criteria:

1. They proposed a method, tool, model, or framework for reconstructing or recovering microservice architecture from existing systems;
2. They used static, dynamic, hybrid, model-driven, telemetry-based, or deployment-based analysis to derive architectural information from microservice systems;
3. They produced architecture-level outputs such as service dependency graphs, communication diagrams, context maps, bounded contexts, topology graphs, intermediate representations, or visual architecture views;
4. They reviewed, mapped, or compared methods and tools for microservice architecture reconstruction;
5. They provided foundational concepts required to define SAR or cloud-native microservice systems.

Table 1

Search and Screening Summary. Google Scholar Records are Reported Separately as a Complementary Discovery Source

Stage	Records
Scopus records identified after filters	243
Web of Science records identified after filters	268
Google Scholar records screened	First 50 results per query; 10 queries
Records after deduplication and scope screening	≈ 200
Records excluded during title/abstract and full-text screening	≈ 178
Studies included in the thematic synthesis	22
reconstruction-focused studies	15
foundational / contextual studies	7

Studies were excluded if they focused only on general microservice design, deployment, education, AI use in the microservice life cycle, anti-pattern cataloguing, or migration from monoliths to microservices without reconstructing the architecture of an existing microservice system. Duplicate records, non-English publications, non-peer-reviewed sources without methodological contribution, and works outside Computer Science and Engineering were also excluded. Because studies on AI in the microservice life cycle were excluded, any discussion of AI-assisted reconstruction is treated as a future research implication rather than as a synthesised review finding.

2.7. Screening and corpus construction

The selection process combined deduplication, scope screening, and multi-stage relevance screening. The Scopus and Web of Science results were first merged; duplicates and records lying outside the review scope (including hits returned only through the broad software-architecture-reconstruction query branch and generic keyword matches) were removed, leaving approximately 200 candidate records together with the Google Scholar results. These candidates were then screened in three stages. First, titles and abstracts were screened to exclude clearly irrelevant records. Second, potentially relevant records were assessed using their introduction, methods, and conclusion sections. Third, the remaining studies were examined in full text to determine whether they contributed directly to at least one analytical dimension of the review. When studies were included only as conceptual or contextual anchors, they were distinguished from reconstruction-focused studies during synthesis; of the 22 included studies, 15 are reconstruction-focused studies and 7 provide foundational or contextual material. Screening and data extraction were carried out by two of the authors, with disagreements resolved through discussion. The resulting counts at each stage are summarised in Table 1. Because the Google Scholar branch is not fully reproducible and the database branches returned many out-of-scope records, the counts after deduplication and scope screening are reported as approximations; the table is intended to make the screening funnel transparent rather than to claim exhaustive coverage.

2.8. Data extraction

For each included study, the following data items were extracted: bibliographic information, study type, evidence-source family, input artefacts, core reconstruction technique, architectural output, evaluation strategy, reported strengths, reported limitations, and contribution to the synthesis. The extraction structure was designed to distinguish between the evidence used by each method, the processing technique applied, and the architectural views produced as outputs. It also allowed the corpus to be grouped into static, dynamic, and hybrid reconstruction, with model-driven processing, intermediate representations, visualisation, and tool comparison recorded as cross-cutting characteristics.

2.9. Synthesis procedure

The extracted data were synthesised thematically. Studies were assigned to static, dynamic, or hybrid reconstruction according to their primary evidence sources. Static studies relied on artefacts available without executing the system, dynamic studies relied on runtime observations, and hybrid studies combined static and runtime evidence. Model-driven processing was treated as a cross-cutting technique because it can operate on static, dynamic, or combined evidence without determining the evidence-source category itself. Intermediate representations, visualisation, and comparative tool evaluation were likewise treated as cross-cutting characteristics rather than as separate families. Within each family, the synthesis compared input artefacts, reconstruction logic, architectural outputs, evaluation evidence, strengths, and limitations.

The strength of evidence for each reconstruction family was assessed qualitatively using three signals: (I) the number of supporting studies, (II) the depth of evaluation, and (III) the consistency of findings across studies. For signal (II), the evaluation maturity of each study was classified as a proof-of-concept or tool demonstration, a case study, a benchmark-based evaluation, an empirical comparison, or a secondary study, and this classification is reflected in the type/method column of Table 2; no formal risk-of-bias instrument was applied. Labels were assigned conservatively. A label of *strong* required several supporting studies, empirical evaluation on real or benchmark systems in most of those studies, and broadly consistent findings. A label of *moderate* was assigned when evidence was concentrated in a smaller number of tool papers, proof-of-concept evaluations, or secondary studies. A label of *limited* was used when a family rested on very few primary studies, for example, a single tool study supplemented by secondary sources. Where the signals conflicted, the lower confidence label was retained. Signal (III) was interpreted cautiously, because several studies in the corpus share authors and tooling (Section 4.3), so agreement between them is weaker evidence of independent corroboration. This appraisal was intended to compare the maturity of evidence families, not to rank individual tools.

The synthesis was interpretive rather than statistical. No meta-analysis was performed because the included studies used heterogeneous systems, artefacts, outputs, evaluation procedures, and quality criteria. The review, therefore, identifies recurring methodological patterns, trade-offs, and research gaps rather than estimating quantitative effect sizes.

3. Results

The results are based on the extraction and thematic synthesis of 22 studies. Table 2 first records what each study contributes: its type and reconstruction method, input artefacts, and architectural output. The narrative that follows groups the studies by the source of architectural evidence they use. This yields three families: static reconstruction from code and related development artefacts, dynamic reconstruction from runtime evidence, and hybrid reconstruction that combines static and runtime sources. Model-driven transformation is examined across these families as a processing technique rather than as an evidence source.

3.1. Characteristics of included studies

Table 2 summarises the 22 studies retained for synthesis.

Although varied, the studies fall into recognisable patterns. They include reviews and mapping studies, tool papers, empirical comparisons, case studies, and model-driven work. Their inputs fall into a smaller set of recurring sources: source code, bytecode, configuration and deployment descriptors on the static side, and logs, traces, metrics, and telemetry on the runtime side. Some methods combine these sources, while others transform the extracted evidence through modelling or topology representations. Their outputs are also more concentrated than the range of methods might suggest. Service dependency graphs, communication diagrams, context maps, bounded contexts, and topology models recur throughout the corpus. The evaluations are less uniform, ranging from prototypes and case studies to benchmark-based assessments and tool comparisons. The limitations reported across the studies are similarly

Table 2
Characteristics of the Included Studies

Study	Type / method	Primary artefacts	Architectural output
[1]	Mapping; cloud-native background	Cloud-native literature	CNA concepts/trends
[13]	Tool; dynamic telemetry analysis	Service-mesh access logs; other telemetry discussed	Service dependency graph; heatmap/runtime views
[4]	Taxonomy/review; general SAR	Code, expert, dynamic, historical data	Architectural views / SAR outputs
[5]	Tool/case; hybrid/MDE recovery	GitHub, Docker, containers, logs	MicroART-DSL; deployment/software architecture
[7]	Toolchain; hybrid with model-driven transformation	Kubernetes manifests + monitored interactions	μ TOSCA / TOSCA topology graph
[8]	Method/case; static SAR with model-driven reconstruction	Documentation, source code, build/configuration files; optional technology models	Reconstructed MSA viewpoint models
[14]	Empirical/tool; static SAR with model-driven transformation	Java source, build/configuration files, and Docker artefacts	Platform-specific and platform-independent architecture models; service, communication, and infrastructure views
[15]	Rapid review; reasoning synthesis	Studies on analysis, testing, metrics, visualisation	Roadmap incl. SDG/metrics/views
[16]	Mapping; runtime visualisation	Runtime anti-pattern studies	Runtime anti-pattern detection/visualisation synthesis
[17]	Tool; static + AR visualisation	Source code/static-analysis artefacts	Service view; 3D/AR microservice views
[18]	Approach; static + visual models	Codebase + static-analysis output	Holistic/dependency visual models
[19]	Empirical/tool; static evolution analysis	Versioned microservice source code	Evolution metrics; inter-service/data views
[6]	Tool; static SAR	Microservice source code	Communication diagram; context map; bounded contexts
[20]	Tool/case; static SAR	Java bytecode via GraalVM Native Image/Graal IR; service metadata	Service dependency graph; context map
[21]	Tool/case; static SAR	Source code + code artefacts	Context maps; communication diagrams; data model
[22]	Tool; static SAR	Code, bytecode, config/deployment files	Domain, service, technology, operation views
[23]	Case study; static smell detection	Source code + bytecode	Code-smell / graph output
[2]	Empirical comparison; static tool comparison	Static tools, benchmarks, ground truth diagrams	Recovered components, connections, endpoints
[9]	Tool; static, language-agnostic	Source code, LAAST, benchmarks	Components, data entities, endpoints
[24]	Position paper; static-analysis agenda	Cloud-native/static-analysis tool experience	Challenges and direction for future tools
[11]	Review; static/dynamic review	Prior studies/tools/visualisations	Static/dynamic SAR and visualisation synthesis
[10]	Mapping; tool classification	Published tools/studies	Tool taxonomy; input/output classes

consistent: language and framework dependence, endpoint-matching ambiguity, incomplete runtime coverage, toolchain complexity, and limited industrial validation. A key limitation was recorded for each study during extraction, and because these individual limitations recur rather than diverge, they are synthesised here and, by family, in Table 3 instead of being listed per study.

Among the 15 reconstruction-focused studies, twelve use static evidence, one uses dynamic evidence, and two use hybrid evidence. The remaining seven studies are still useful, but in a different way: they provide cloud-native context, foundational SAR taxonomy, tool-classification mappings, position statements, or visualisation-focused evidence rather than an evaluated reconstruction method of their own. Some of these cross-cutting studies also inform the family-level synthesis in Table 3. Model-driven processing cuts across the evidence-source count: Rademacher et al. and MiSAR are assigned to static reconstruction because their recovery evidence is available without runtime observation, whereas MicroART and μ TOSCA are assigned to hybrid reconstruction because they combine static or deployment evidence with runtime information [5, 7, 8, 14]. The boundary turns on the source used to recover the architecture: a method that emits an explicit model from static artefacts remains static, whereas a method that also observes the running system is hybrid. The corpus is therefore best read as evidence organised around sources of architectural information, not as a catalogue of interchangeable tools. The following synthesis focuses on the three reconstruction families while drawing on cross-cutting studies where they clarify terminology, outputs, or limitations.

3.2. Thematic findings

The three families differ in what they accept as evidence. Static approaches start from declared or inferable structure, dynamic approaches from observed interaction, and hybrid approaches combine static and runtime sources. Model-driven techniques may structure and transform the evidence within any of these families.

3.2.1. Static reconstruction methods

Static reconstruction recovers architectural views from artefacts that can be inspected without running the system. In this corpus, those artefacts include source code, bytecode, annotations, build files, configuration files, and deployment descriptors. The recovered views are mainly structural, centred on service dependencies, inter-service communication, and the data each service owns [2, 6, 19–21]. MiSAR demonstrates that static reconstruction need not rely on source code alone: it integrates Java source, build, configuration, and Docker artefacts before transforming the extracted information into explicit architectural models [14].

The main advantage is practical: static methods can be run before deployment and can be integrated into development-time documentation, CI pipelines, and architecture-drift checks. The same evidence source also explains their limits. They depend on the languages, frameworks, and conventions understood by the analyser. Endpoint matching can be ambiguous, and dependencies hidden behind reflection, dynamic binding, generated code, or implicit framework behaviour may not be recovered. A static view is therefore strongest as evidence of declared or inferable structure, not as a complete account of runtime behaviour [2, 6, 9, 20].

3.2.2. Dynamic reconstruction methods

Dynamic reconstruction works from evidence produced while the system runs: logs, traces, metrics, telemetry streams, monitoring data, and, in some cases, service-mesh observations. The focus is not on what the code declares, but on which services actually communicate during observed executions. This makes dynamic evidence suitable for runtime dependency graphs, communication-frequency views such as heatmaps, and analyses of operational behaviour or runtime anti-patterns [11, 13, 16].

Its main strength is behavioural fidelity, since the recovered edges correspond to interactions observed during execution; its main weakness is incompleteness. A dependency that is not triggered by the observed workload, not captured by the monitoring configuration, or lost through weak instrumentation

will not appear in the reconstructed view [11, 13]. The evidence base is also smaller than for static reconstruction. In the reviewed corpus, purely dynamic reconstruction is represented by a single primary tool study [13], with the broader picture provided by review and mapping studies [11, 16]. For that reason, Table 3 rates the confidence for this family as limited rather than moderate: it rests on a single primary tool study, so the evidence is too thin to support consistent corroboration.

3.2.3. Hybrid reconstruction and model-driven transformations

Hybrid reconstruction starts from the observation that neither static nor dynamic evidence is complete on its own. These methods combine sources such as repositories, Docker or Kubernetes descriptors, container information, runtime logs, and monitored interactions [5, 7]. Model-driven transformation is distinct: it structures and refines recovered evidence through explicit architectural models regardless of whether the evidence is static, dynamic, or combined. MicroART-DSL, μ TOSCA, LEMMA-based viewpoint models, and MiSAR’s platform-specific and platform-independent models illustrate this cross-cutting role [5, 7, 8, 14].

Hybrid approaches can cover more of the architecture because static artefacts expose declared services, endpoints, data structures, and deployment assumptions, while runtime evidence confirms which components communicate during observed execution. Model-driven techniques make the recovered architecture analysable, refinable, and transformable into other views, but require suitable metamodels, transformation rules, and validation. MiSAR applies such transformations exclusively to static evidence. Across its three evaluated systems, it reported F-measures ranging from 59% to 94%; however, its source-level analysis remains centred on supported Java/Spring technologies, and the expected architectures used for evaluation were constructed manually [14]. MiSAR therefore strengthens empirical support for model-driven reconstruction but does not constitute evidence for static–runtime fusion. Hybrid approaches also remain sensitive to platform and tooling support, workload coverage, and manual refinement, which helps explain why the hybrid evidence base remains smaller than that for static reconstruction [5, 7].

3.3. Summary of evidence

Table 3 summarises the evidence from the three families identified in the thematic synthesis.

The confidence labels describe the breadth and evaluation maturity of the published evidence within this corpus. They should not be interpreted as counts of independent replications because several supporting publications share authors, tools, or benchmark systems. Model-driven transformation is not assigned a separate confidence label because it is a processing technique used within the evidence-source families rather than an additional family.

Table 3 compares the three evidence-source families directly. Figure 1 then shows their relationship: static and dynamic reconstruction produce complementary partial views, while hybrid reconstruction joins evidence from both. Model-driven transformations can structure the evidence produced within any of these families.

Across the three families, the strength of evidence is uneven. Static reconstruction has the broadest base, with several tool and empirical studies, a recent comparison of static recovery tools, and the quantitatively evaluated MiSAR approach. Dynamic reconstruction provides the closest view of observed behaviour, but its primary evidence is thinner. Hybrid reconstruction offers the broadest architectural coverage, though it is supported by only a small set of tool studies. Model-driven transformations occur in both static and hybrid work rather than forming a fourth evidence-source family. Two points cut across all three families. First, reusable architecture views usually depend on some intermediate representation, whether a dependency graph, context map, topology graph, DSL, or model. Second, tool fragmentation still makes comparison, replication, and adoption difficult.

Table 3
Summary of Evidence by Evidence-Source Family

	Static reconstruction	Dynamic reconstruction	Hybrid reconstruction
Characteri- sation	Recover code-level and design-proximal architecture from source, bytecode, annotations, build/configuration artefacts, and deployment descriptors.	Recover behaviour-proximal architecture from observed runtime communication and telemetry data.	Combine static or deployment artefacts with runtime observations to improve reconstruction coverage.
Typical artefacts	Source code; bytecode; annotations; build files; configuration files; deployment descriptors.	Logs; traces; metrics; telemetry streams; service-mesh data; runtime monitoring data.	Source code or deployment descriptors combined with runtime logs, monitored interactions, or telemetry.
Architectural outputs	Service dependency graphs; communication diagrams; context maps; bounded contexts; data-oriented views; PSM/PIM or viewpoint models.	Runtime service dependency graphs; observed communication views; heatmaps; runtime anti-pattern views.	Integrated architecture models; MicroART-DSL models; μ TOSCA topology graphs; refined service/deployment views.
Strengths	Applicable before deployment; suitable for CI/CD and continuous architecture documentation.	Captures observed service interactions and operational behaviour; less dependent on programming language.	Potentially broader architectural coverage by combining declared or deployment structure with observed behaviour.
Limitations	Language/framework and mapping-rule dependence; endpoint ambiguity; reflection, dynamic binding, and runtime-only dependencies may be missed.	Depends on workload coverage, observability quality, and monitored execution scenarios.	Platform and tooling dependence; workload-dependent runtime coverage; manual refinement and validation may be required.
Confidence Supporting studies	Strong [2, 6, 8, 9, 14, 17–23]	Limited [11, 13, 16]	Moderate [5, 7]

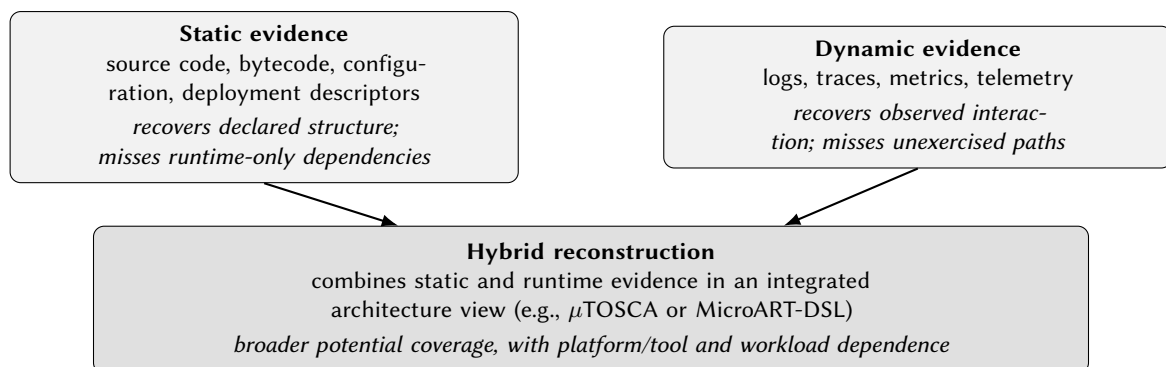


Figure 1: Complementarity of evidence sources across the three reconstruction families. Static evidence (source code, bytecode, configuration, and deployment descriptors) and dynamic evidence (logs, traces, metrics, and telemetry) recover partial, complementary views, which hybrid reconstruction combines. Model-driven transformations are not a fourth evidence source; they can structure evidence within any family through representations such as μ TOSCA, MicroART-DSL, LEMMA, and MiSAR’s PSM/PIM models.

4. Discussion

4.1. Main interpretation

The main result of the review is that microservice architecture reconstruction is not a single-technique problem. Architectural evidence is scattered across source code, bytecode, configuration files, deployment artefacts, runtime logs, traces, and telemetry streams. Each reconstruction family therefore sees the system from a different angle: static methods recover declared or inferable structure, dynamic methods recover interactions that were actually observed, and hybrid methods reconcile these partial views [6, 7, 11, 13, 20]. Model-driven techniques provide a separate processing dimension by expressing recovered evidence as explicit models [8, 14].

This is why none of the families should be treated as universally sufficient. Static reconstruction is repeatable and can be used before deployment, but it can miss dependencies that exist only at runtime or are hidden behind framework behaviour, reflection, dynamic binding, or generated infrastructure code [2, 6, 20]. Dynamic reconstruction is closer to operational reality, but only for the workloads and scenarios that are monitored with enough detail [11, 13]. Hybrid approaches respond to this gap by combining complementary evidence sources, but remain dependent on supported platforms and tools, representative runtime workloads, and refinement and validation of the recovered model [5, 7]. Model-driven techniques instead shift attention to metamodel coverage, transformation correctness, and validation of the resulting representation [8, 14].

Organising the evidence by source yields a more specific claim. A more complete reconstruction is constrained not only by the capability of the individual extraction techniques, which already recover rich structural and behavioural views yet still miss runtime-only dependencies and face endpoint ambiguity, but also by the absence of a shared intermediate representation and a dependable way to fuse heterogeneous evidence. On this reading, the family that combines sources is not simply one option among three; it is where many of the open problems concentrate, and progress now depends as much on integration and representation as on better single-source analysers.

4.2. Methodological and practical implications

The practical implication is that the reconstruction method should be chosen for the question being asked, not for the promise of a complete architectural picture. If the aim is continuous documentation, architecture-drift analysis, or development-time feedback, static reconstruction is usually the most practical baseline because it can be run without a deployed system and can be aligned with code-analysis workflows [6, 19, 21]. If the aim is to understand operational dependencies, observed communication paths, traffic concentration, or runtime anti-patterns, dynamic reconstruction is more appropriate, provided that the observability data are representative enough [11, 13, 16].

The same trade-off matters for tool builders. Combining evidence sources is the clearest route to broader potential coverage, but the toolchain must support the relevant platforms, exercise representative workloads, and permit validation and manual refinement of incomplete results [5, 7]. Model-driven transformations can make the result explicit and reusable, but only when their metamodels and mapping rules cover the relevant technologies [8, 14]. For practitioners, the more modest lesson is often the more useful one: choose the lightest method that answers the immediate reconstruction question, and add runtime evidence or an explicit model when the initial static view is not enough [6, 13].

These choices also matter beyond documentation and maintenance. An accurate, current architectural view can also support security and cyber-hygiene work on microservice systems: tasks such as enumerating the attack surface, identifying trust boundaries and unintended service-to-service paths, and estimating the blast radius of a change or compromise all build on knowing how services actually connect. Here, the families can play complementary roles. Static reconstruction can flag declared but unintended connectivity before deployment, whereas dynamic reconstruction can reveal communication paths that occur at runtime but were never declared; an outdated or single-source view is likely to weaken any analysis built on top of it. In this sense, reconstruction is not only a comprehension

aid but also a potential foundation for the kind of architectural assurance that conflict management in networked systems requires [2].

4.3. Limitations of the evidence

The evidence base also limits how far these conclusions can be taken. Most approaches are evaluated as prototypes, benchmark experiments, limited case studies, or tool demonstrations rather than as long-running deployments in industrial settings. That makes it hard to judge maintenance cost, scalability, and everyday usability in systems that keep changing after the evaluation ends [6, 14, 17, 18, 20].

A further limitation concerns the corpus’s composition. A large share of the reconstruction-focused studies reviewed here originates from a small number of closely collaborating research groups, and several share authors, tools, and benchmark systems. This concentration reflects the current state of microservice-specific reconstruction, which has so far been driven by a few active laboratories, but it also means that consistency across studies must be read with caution: agreement between studies that share authorship or tooling is weaker evidence of independent corroboration than agreement between fully independent teams. The confidence labels in Table 3 should therefore be read as summaries of evidence maturity within this corpus rather than as claims of independent replication across the field, and broader participation from independent groups remains necessary before the static, dynamic, and hybrid families can be compared on fully neutral ground.

The literature is also concentrated around particular ecosystems, especially Java, Spring, Kubernetes, and related cloud-native tooling. Some work moves toward language-agnostic analysis, but the evidence is not yet broad enough to demonstrate that current methods perform equally well across highly polyglot microservice systems [6, 7, 9, 14, 19, 20]. A further obstacle is the fragmentation of the tool landscape. Mapping and comparison studies show substantial variation in supported inputs, assumptions, outputs, availability, and evaluation procedures, which makes systematic comparison difficult even when tools pursue similar reconstruction goals [2, 10].

Finally, visualisation-oriented studies show that reconstructed evidence can be turned into useful views, including dependency graphs, heatmaps, 3D models, and augmented-reality representations. The open question is not whether these views can be produced, but whether they continue to help engineers reason accurately when the system is large, changing, and operationally messy. Existing evaluations are often task-specific, prototype-based, or small in scale, so more evidence is needed before strong claims can be made about their effectiveness in industrial practice [11, 16–18].

5. Gaps and future directions

The review points to five gaps that still need work, and they are connected rather than independent. The most basic one is the lack of a common evaluation ground. Reconstruction methods are still rarely tested against shared benchmark systems with explicit ground-truth architectures, and although prototypes, case studies, and benchmark-based evaluations exist, the absence of common datasets, workload descriptions, and missed-dependency reporting makes it hard to compare static, dynamic, and hybrid methods on equal terms [2, 6, 13, 20].

Closely tied to this, and following from the language bias noted in Section 4.3, current approaches are only partially language- and platform-agnostic. Future work should test reconstruction across diverse programming languages, frameworks, deployment models, and communication technologies, with explicit attention to polyglot systems [2, 9, 24, 25].

A harder problem is integrating heterogeneous evidence. Static approaches may miss runtime-only dependencies, while dynamic approaches omit interactions that are never exercised during observation. Hybrid methods only partly close this gap because their recovered views remain dependent on platform and tooling support, representative runtime workloads, and validation or manual refinement of incomplete results [3, 5, 7, 11, 13]. Model-driven techniques add a related challenge: the explicit model remains only as complete and accurate as its metamodels, mapping rules, and source coverage [8, 14].

A related obstacle is that no shared intermediate representation for reconstructed microservice architectures has emerged across the reviewed studies. Existing studies employ different ones, including topology graphs, language-agnostic component models, MicroART-DSL, μ TOSCA, LEMMA-based models, MiSAR platform-specific and platform-independent models, and visual architecture views. This diversity supports experimentation but limits interoperability between reconstruction, visualisation, smell detection, evolution analysis, and refactoring tools [5, 7–9, 11, 14, 18, 20].

The last direction, and the most speculative, is AI-assisted reconstruction. Because studies on the use of AI in the microservice life cycle were outside the inclusion criteria (Section 2.6), this direction is advanced as an agenda motivated by the gaps above rather than as a synthesis of AI-specific evidence. The reviewed evidence-source methods and cross-cutting model-driven techniques still face limitations in endpoint ambiguity, heterogeneous artefact interpretation, dependency reconciliation, missing runtime or structural evidence, and model validation [2, 7–10, 13, 14, 20]. These limitations suggest that AI should strengthen hybrid reconstruction pipelines rather than replace existing techniques: it could be investigated for interpreting heterogeneous artefacts, matching endpoints, reconciling static and dynamic evidence, identifying missing architectural links, and validating reconstructed models against multiple evidence sources. Accordingly, future work should prioritise validated, interoperable, multi-source reconstruction pipelines over further isolated tools. These pipelines should be evaluated using shared benchmarks and realistic workloads, with explicit measures of accuracy, scalability, and reproducibility [2, 10, 13].

6. Conclusions

This review was guided by four research questions, which the synthesis answers as follows.

- **RQ1 (methods).** Reconstruction is organised into three evidence-source families: static analysis of development and deployment artefacts, dynamic analysis of runtime evidence, and hybrid reconstruction combining static and runtime sources. Model-driven transformations constitute a cross-cutting strategy for representing and refining recovered evidence.
- **RQ2 (artefacts).** Static methods rely on source code, bytecode, annotations, build and configuration files, and deployment descriptors; dynamic methods rely on logs, traces, metrics, telemetry, and service-mesh data; and hybrid methods combine evidence from these sources. Model-driven techniques transform the extracted evidence through topology models, DSLs, or metamodels rather than introducing a separate source type.
- **RQ3 (outputs).** The recovered views are predominantly service dependency graphs, communication diagrams, context maps, bounded contexts, runtime interaction and topology views, and integrated architecture models.
- **RQ4 (trade-offs).** The governing trade-off is among repeatability, behavioural fidelity, and coverage (Section 4.1): static methods favour repeatability, dynamic methods behavioural fidelity, and hybrid methods coverage, with no family optimising all three at once. On the available evidence, confidence is strongest for static reconstruction and more limited for dynamic and hybrid reconstruction, which rest on fewer primary evaluations, as recorded in Table 3.

In sum, the reviewed methods are complementary rather than competing: static, dynamic, and hybrid reconstruction recover different and partially overlapping views of the same architecture, while model-driven transformations turn recovered evidence into explicit, reusable representations. The principal unresolved challenges, namely language and framework dependence, incomplete dependency recovery, limited runtime coverage, fragmented tooling, and the absence of a shared intermediate representation [2, 10, 14], motivate the agenda set out in Section 5: interoperable, language-agnostic, and empirically validated reconstruction pipelines, evaluated on shared benchmarks for heterogeneous cloud-native systems.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

- [1] N. Kratzke, P.-C. Quint, Understanding cloud-native applications after 10 years of cloud computing – a systematic mapping study, *Journal of Systems and Software* 126 (2017) 1–16. doi:10.1016/j.jss.2017.01.001.
- [2] S. Schneider, A. Bakhtin, X. Li, J. Soldani, A. Brogi, T. Cerny, R. Scandariato, D. Taibi, Comparison of static analysis architecture recovery tools for microservice applications, *Empirical Software Engineering* 30 (2025) 128. doi:10.1007/s10664-025-10686-2.
- [3] J. S. Al-Azzeh, M. Al Hadidi, R. S. Odarchenko, S. Gnatyuk, Z. Shevchuk, Z. Hu, Analysis of self-similar traffic models in computer networks, *International Review on Modelling and Simulations* 10 (2017).
- [4] S. Ducasse, D. Pollet, Software architecture reconstruction: A process-oriented taxonomy, *IEEE Transactions on Software Engineering* 35 (2009) 573–591. doi:10.1109/TSE.2009.19.
- [5] G. Granchelli, M. Cardarelli, P. Di Francesco, I. Malavolta, L. Iovino, A. Di Salle, Towards recovering the software architecture of microservice-based systems, in: 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), IEEE, 2017, pp. 46–53. doi:10.1109/ICSAW.2017.48.
- [6] V. Bushong, D. Das, A. Al Maruf, T. Cerny, Using static analysis to address microservice architecture reconstruction, in: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2021, pp. 1199–1201. doi:10.1109/ASE51524.2021.9678749.
- [7] J. Soldani, G. Muntoni, D. Neri, A. Brogi, The μ TOSCA toolchain: Mining, analyzing, and refactoring microservice-based architectures, *Software: Practice and Experience* 51 (2021) 1591–1621. doi:10.1002/spe.2974.
- [8] F. Rademacher, S. Sachweh, A. Zündorf, A modeling method for systematic architecture reconstruction of microservice-based software systems, in: Enterprise, Business-Process and Information Systems Modeling, volume 387 of *Lecture Notes in Business Information Processing*, Springer, 2020, pp. 311–326. doi:10.1007/978-3-030-49418-6_21.
- [9] M. Schiewe, J. Curtis, V. Bushong, T. Cerny, Advancing static code analysis with language-agnostic component identification, *IEEE Access* 10 (2022) 30743–30761. doi:10.1109/ACCESS.2022.3160485.
- [10] A. Bakhtin, X. Li, J. Soldani, A. Brogi, T. Cerny, D. Taibi, Tools reconstructing microservice architecture: A systematic mapping study, in: Software Architecture. ECSA 2023 Tracks, Workshops, and Doctoral Symposium, volume 14590 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 3–18. doi:10.1007/978-3-031-66326-0_1.
- [11] T. Cerny, A. S. Abdelfattah, V. Bushong, A. Al Maruf, D. Taibi, Microservice architecture reconstruction and visualization techniques: A review, in: 2022 IEEE International Conference on Service-Oriented System Engineering (SOSE), IEEE, 2022, pp. 39–48. doi:10.1109/SOSE55356.2022.00011.
- [12] M. Gusenbauer, N. R. Haddaway, Which academic search systems are suitable for systematic reviews or meta-analyses? evaluating retrieval qualities of google scholar, pubmed, and 26 other resources, *Research Synthesis Methods* 11 (2020) 181–217. doi:10.1002/jrsm.1378.
- [13] A. Al Maruf, A. Bakhtin, T. Cerny, D. Taibi, Using microservice telemetry data for system dynamic analysis, in: 2022 IEEE International Conference on Service-Oriented System Engineering (SOSE), IEEE, 2022, pp. 29–38. doi:10.1109/SOSE55356.2022.00010.
- [14] N. Alshuqayran, N. Ali, R. Evans, A model-driven architecture approach for recovering microservice architectures: Defining and evaluating MiSAR, *Information and Software Technology* 186 (2025) 107808. doi:10.1016/j.infsof.2025.107808.

- [15] A. S. Abdelfattah, T. Cerny, Roadmap to reasoning in microservice systems: A rapid review, *Applied Sciences* 13 (2023) 1838. doi:10.3390/app13031838.
- [16] G. Parker, S. Kim, A. Al Maruf, T. Cerny, K. Frajtak, P. Tisnovsky, D. Taibi, Visualizing anti-patterns in microservices at runtime: A systematic mapping study, *IEEE Access* 11 (2023) 4434–4442. doi:10.1109/ACCESS.2023.3236165.
- [17] T. Cerny, A. S. Abdelfattah, V. Bushong, A. Al Maruf, D. Taibi, Microvision: Static analysis-based approach to visualizing microservices in augmented reality, in: *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, IEEE, 2022, pp. 49–58. doi:10.1109/SOSE55356.2022.00012.
- [18] T. Cerny, A. S. Abdelfattah, J. Yero, D. Taibi, From static code analysis to visual models of microservice architecture, *Cluster Computing* 27 (2024) 4145–4170. doi:10.1007/s10586-024-04394-7.
- [19] A. S. Abdelfattah, T. Cerny, J. Yero Salazar, X. Li, D. Taibi, E. Song, Assessing evolution of microservices using static analysis, *Applied Sciences* 14 (2024) 10725. doi:10.3390/app142210725.
- [20] R. Hutcheson, A. Blanchard, N. Lambaria, J. Hale, D. Kozak, A. S. Abdelfattah, T. Cerny, Software architecture reconstruction for microservice systems using static analysis via GraalVM Native Image, in: *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, IEEE, 2024, pp. 12–22. doi:10.1109/SANER60148.2024.00008.
- [21] V. Bushong, D. Das, T. Cerny, Reconstructing the holistic architecture of microservice systems using static analysis, in: *Proceedings of the 12th International Conference on Cloud Computing and Services Science (CLOSER)*, SciTePress, 2022, pp. 149–157. doi:10.5220/0011032100003200.
- [22] A. Walker, I. Laird, T. Cerny, On automatic software architecture reconstruction of microservice applications, in: *Information Science and Applications*, volume 739 of *Lecture Notes in Electrical Engineering*, Springer, 2021, pp. 223–234. doi:10.1007/978-981-33-6385-4_21.
- [23] A. Walker, D. Das, T. Cerny, Automated code-smell detection in microservices through static analysis: A case study, *Applied Sciences* 10 (2020) 7800. doi:10.3390/app10217800.
- [24] T. Cerny, D. Taibi, Microservice-aware static analysis: Opportunities, gaps, and advancements, in: *Joint Post-proceedings of the Third and Fourth International Conference on Microservices (Microservices 2020/2022)*, volume 111 of *Open Access Series in Informatics (OASIs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, pp. 2:1–2:14. doi:10.4230/OASIs.Microservices.2020-2022.2.
- [25] Z. Hu, S. Gnatyuk, T. Okhrimenko, S. Tynymbayev, M. Iavich, High-speed and secure PRNG for cryptographic applications, *International Journal of Computer Network and Information Security* 12 (2020) 1–10. doi:10.5815/ijcnis.2020.03.01.