© JUL 2026 | IRE Journals | Volume 10 Issue 1 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I1-1719451

Intelligent Software Defect Prediction Using Multimodal Deep Learning Through the Integration of Source Code, Software Metrics and Historical Development Data

POOJA GANESH DHONE[1], DR. BRIJENDRA GUPTA[2]
[1,2]*Siddhant College of Engineering, Sadumbare*

Abstract- ***Software defect prediction has historically relied on a single view of source code — either handcrafted metrics, token sequences, or structural representations in isolation. Multimodal deep learning addresses the fundamental limitation that no single code view captures the full richness of software artifacts: their syntax, structure, semantics, history, and natural-language context. This report provides a comprehensive survey of multimodal deep learning approaches for software defect prediction, covering the major modalities (lexical/semantic, structural/AST, control-and-data-flow, metric-based, and natural language) and the fusion architectures that combine them — concatenation, attention-gating, cross-attention, and contrastive multi-view learning. We synthesize findings from over 50 recent publications (2020-2026), including GMCA-SDP cross-attention fusion, FusionVul multimodal vulnerability detection, hierarchical CNN fusion of AST/CFG/DDG, and emerging vision-language model applications. We benchmark performance across datasets, analyze fusion strategy trade-offs, address challenges in modality alignment and missing data, and map the trajectory toward unified multimodal foundation models for software quality assurance.***

I. INTRODUCTION

Software defect prediction (SDP) has evolved through several distinct generations of representation learning. The first generation relied on handcrafted software metrics — lines of code, cyclomatic complexity, coupling and cohesion measures (the CK metric suite) — fed into classical machine learning classifiers. The second generation introduced single-modality deep learning: treating source code as a token sequence (Word2Vec, CodeBERT) or as a structural graph (AST-based tree networks). Each of these unimodal approaches, while effective, captures only a partial view of the software artifact.

Multimodal deep learning represents the third and current generation: combining two or more complementary views of code — lexical/semantic, structural, control-flow, metric-based, and natural language — into a unified representation that captures information no single modality could provide alone. The core insight is that defects manifest differently across these views: a null-pointer bug may be invisible in token sequences but obvious in a data-flow graph; a logic error may be invisible structurally but evident in variable naming patterns; a regression risk may be invisible in the code itself but predictable from commit message sentiment and developer experience.

This report surveys multimodal deep learning for software defect prediction in depth. Section 2 defines the modality taxonomy. Section 3 reviews fusion architectures. Section 4 covers key frameworks and case studies. Section 5 addresses vision-language and emerging multimodal LLM approaches. Section 6 covers JIT multimodal prediction. Section 7 reviews benchmarks and evaluation. Section 8 addresses challenges. Section 9 surveys industry adoption. Section 10 covers future directions and conclusions.

II. MODALITY TAXONOMY FOR CODE

2.1 The Five Core Modalities

Source code and its surrounding software engineering artifacts can be decomposed into distinct modalities, each capturing a different aspect of the software's identity:

© JUL 2026 | IRE Journals | Volume 10 Issue 1 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I1-1719451

Modality	Representation	What It Captures	Typical Encoder
Lexical / Semantic	Token sequences, character sequences (LCS)	Identifier semantics, API usage patterns, idioms	CodeBERT, UniXCoder, Word2Vec
Structural	Abstract Syntax Tree (AST)	Grammatical structure, nesting, syntax patterns	Tree-LSTM, GCN, Tree-CNN
Control/Data Flow	CFG, DFG, PDG, CPG	Execution paths, data propagation, dependencies	GGNN, GAT, Heterogeneous GNN
Metric-Based	Software metrics (LOC, CC, CK suite)	Quantitative complexity and design quality signals	MLP, tabular embeddings
Natural Language	Commit messages, comments, bug reports, docstrings	Developer intent, change rationale, defect context	BERT, RoBERTa, GPT-based encoders
Visual (emerging)	UI screenshots, diagrams, flowcharts	Front-end behavior, design-implementation mismatch	ViT, CLIP, multimodal LLMs

2.2 Modality Pairings in Recent Research

Most multimodal SDP research combines two or three modalities rather than attempting full five-way fusion, as each additional modality introduces alignment complexity and computational overhead. The most common pairings observed in the literature are:

- Semantic + Structural: Code token sequences (LCS) fused with AST-derived structural features — the most widely studied pairing
- Structural + Control-flow: AST combined with CFG and data dependency graphs (DDG) for richer execution-aware structural understanding
- Metric + Semantic: Traditional handcrafted metrics fused with deep semantic embeddings (e.g., GMCA-SDP)
- Code + Natural Language: Source code changes fused with commit messages and bug report text for JIT prediction
- Code + Visual: Emerging work pairing source/UI code with screenshots or diagrams for front-end defect detection

III. MULTIMODAL FUSION ARCHITECTURES

3.1 Early (Feature-Level) Fusion: Concatenation

The simplest fusion strategy concatenates feature vectors from each modality into a single combined vector before feeding it to a classifier. Hierarchical CNN approaches extract different types of semantic features from AST, CFG, and data dependency graphs (DDG), then fuse them via concatenation for software defect prediction. While straightforward to implement, concatenation-based fusion does not model cross-modal interactions — it treats modalities as independent feature blocks rather than allowing them to inform one another.

3.2 Late Fusion: Ensemble of Unimodal Predictors

Late fusion trains separate models on each modality independently, then combines their predictions (via voting, averaging, or a meta-learner) at the decision stage. This approach is robust to missing modalities (a model can still make a prediction if one modality is unavailable) but cannot capture fine-grained cross-modal feature interactions that occur before the decision layer.

3.3 Attention-Based and Cross-Attention Fusion

Attention mechanisms allow the model to dynamically weight the contribution of each modality based on context, rather than treating all modalities as equally important for every prediction. The GMCA-SDP framework (Cross-Attention Gating Mechanism-based Multimodal Feature Fusion, 2025) integrates traditional metric features with multiple code semantic features through a cross-attention gating mechanism that considers both the contribution differences among different types of features and the information interaction between modalities.

GMCA-SDP's design highlights a key insight: different fusion strategies materially impact final model performance, and a weighting strategy that adapts to feature contribution differences outperforms uniform concatenation. The cross-attention gate

learns, for each input, how much to trust the metric-based view versus the semantic view — for example, weighting metrics more heavily for legacy code with sparse documentation, and semantic features more heavily for well-documented modern code.

3.4 Structural Attention Fusion

For pairing semantic and structural modalities specifically, structural attention mechanisms dynamically model interdependencies between code character sequences and AST-derived structural features. Research on multimodal feature fusion for source code defect detection demonstrates that converting code to AST and back loses information — irreversible information loss during AST conversion yields impoverished unimodal representations when relying solely on either semantic or structural approaches. Structural attention addresses this by fusing both views into a comprehensive representation vector rather than relying on lossy intermediate conversion.

3.5 Contrastive Multi-View Learning

Contrastive learning frameworks such as CODE-MVP (Code Multiple Views with Contrastive Pre-training) learn to represent source code from multiple views — including text, AST, and CFG — by pulling representations of the same code from different views closer together in embedding space while pushing apart representations of different code. This contrastive pre-training objective produces embeddings that are inherently multimodal, encoding agreement across views as a signal of semantic correctness, and has demonstrated strong transferability to downstream tasks including defect prediction.

Fusion Strategy	Cross-Modal Interaction	Robustness to Missing Modality	Computational Cost	Best Use Case
Concatenation (Early)	None — independent features	Low	Low	Simple baselines, quick prototyping
Late Fusion / Ensemble	None — decision-level only	High	Medium (parallel models)	Production systems needing graceful degradation
Cross-Attention Gating	Strong — learned dynamic weighting	Medium	Medium-High	Heterogeneous modalities (metric + semantic)
Structural Attention	Strong — bidirectional interaction	Medium	High	Semantic + structural (AST) pairing
Contrastive Multi-View	Implicit — via shared embedding space	High (view-agnostic)	High (pre-training cost)	Transfer learning, pre-training pipelines

IV. KEY FRAMEWORKS AND CASE STUDIES

4.1 GMCA-SDP: Cross-Attention Gating for Metric-Semantic Fusion

GMCA-SDP addresses a specific gap: while semantic-feature-based methods have become mainstream in defect prediction research, the empirical value of traditional metric features remains significant and should not be discarded. The framework's pipeline consists of four stages: feature extraction from each modality, alignment of features into a common space, multimodal feature fusion via cross-attention gating, and final defect prediction via the fused representation. This explicit four-stage design — extraction, alignment, fusion, prediction — has become a template adopted by subsequent multimodal SDP frameworks.

4.2 FusionVul: Multimodal Vulnerability Detection

FusionVul combines sequential-syntactic features (extracted via UniXCoder from a Linearized Code Sequence) with code property graph (CPG) features extracted via Gated Graph Neural Networks (GGNN). Evaluated across multiple vulnerability datasets including Devign, ReVeal, SVulD, and

DiverseVul, FusionVul achieves the best F1 scores specifically on datasets where function size is highly dispersed and vulnerability types are diverse — indicating that multimodal fusion provides the greatest benefit for complex, heterogeneous vulnerability scenarios rather than simple, homogeneous ones.

Notably, ablation studies on FusionVul reveal an important nuance: on datasets where one modality's embeddings already exhibit high separability (e.g., LCS embeddings on Devign and ReVeal), a purely sequence-based method can match or exceed the fused model. This demonstrates that multimodal fusion is not universally superior — its benefit is conditional on genuine complementary information existing across modalities for the specific dataset and defect type.

4.3 PatternNet: Bug Repair Pattern Prediction via Dual-Branch Fusion

PatternNet, designed for software bug repair pattern prediction, uses a dual-branch architecture: the top branch captures structural features from the buggy code's AST, while the bottom branch is a transformer-based model that simultaneously represents code tokens and the bug report text. By jointly conditioning on structural code features and natural language bug descriptions, PatternNet illustrates a third major fusion target: combining code artifacts with human-authored problem descriptions for more context-aware prediction.

4.4 UniXCoder: Multimodal Pre-training Foundation

UniXCoder is pre-trained on large amounts of multimodal content, including code, code comments, and AST, using self-supervised objectives such as masked language modeling and denoising objectives, supplemented by two supervised pre-training tasks: multimodal contrastive learning and cross-modal generation. As a foundation model, UniXCoder is frequently used as the semantic/sequential encoder branch within larger multimodal defect prediction systems (e.g., as the LCS encoder in FusionVul), demonstrating how multimodal pre-training at the foundation-model level cascades benefits to downstream task-specific multimodal architectures.

4.5 Just-in-Time Defect Prediction as a Multimodal Task

Just-in-time (JIT) defect prediction is explicitly framed as a multimodal task: hand-crafted features, commit logs, and code changes represent three different modalities of a commit. Hand-crafted features are numerical/categorical with clear meanings; commit logs are natural language text; and code changes are written in a programming language. Research bridging expert knowledge with deep learning techniques for JIT defect prediction demonstrates that existing approaches using only expert knowledge or only commit contents are prone to wrong predictions in the absence of multimodal features, since each source presents different, complementary aspects of the commit's risk profile.

Framework	Modalities Combined	Fusion Method	Domain	Key Result
GMCA-SDP	Metrics + multiple semantic views	Cross-attention gating	General SDP	Outperforms single-fusion-strategy baselines
FusionVul	LCS (UniXCoder) + CPG (GGNN)	Learned multimodal fusion	Vulnerability detection	Best F1 on complex/diverse vulnerability datasets
PatternNet	AST + bug report text	Dual-branch transformer	Bug repair pattern prediction	Joint structural + NL context modeling
Hierarchical CNN (Abdu et al.)	AST + CFG + DDG	Concatenation	General SDP	Demonstrated value of 3-way structural fusion
JIT Multimodal (expert+DL)	Hand-crafted metrics + commit logs + code diff	Model fusion / late fusion	JIT defect prediction	Improved accuracy vs. single-source approaches
CODE-MVP	Text + AST + CFG	Contrastive multi-view pre-training	General code representation	Strong transfer to downstream SDP tasks

© JUL 2026 | IRE Journals | Volume 10 Issue 1 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I1-1719451

V. VISION-LANGUAGE MODELS AND EMERGING VISUAL MODALITIES

5.1 The Rise of Multimodal LLMs in Software Engineering

The emergence of multimodal large language models (MLLMs) — GPT-4o, Gemini 2.5, Claude with vision, and open models such as GLM-4.6V — has opened new frontiers for software engineering tasks that involve visual inputs. These models jointly process images and text through unified vision-language pipelines, using either a unified-decoder approach (treating image embeddings as tokens alongside text) or a cross-attention approach (separate encoders per modality, fused via cross-attention layers).

For defect prediction specifically, this enables an emerging class of front-end and UI-related bug detection: comparing rendered UI screenshots against design mockups to detect implementation defects, identifying visual regressions across application versions, and detecting design-implementation mismatches that purely code-based analysis cannot see. The SWE-Bench Multimodal benchmark specifically extends SWE-Bench to evaluate language models on software engineering tasks involving visual inputs such as screenshots and UI mockups, signaling growing community interest in this direction.

5.2 Screenshot-to-Code and Visual Bug Detection

Research on screenshot-to-code generation (Design2Code, ScreenCoder, Widget2Code) demonstrates MLLMs' growing capability to reverse-engineer UI implementations from visual input. While primarily framed as a generation task, the underlying visual-structural alignment capability is directly applicable to defect detection: a model that can map a screenshot to expected code can equally be used to flag discrepancies between actual rendered output and expected/specified behavior — effectively a visual regression and defect detector.

5.3 Limitations of Current Vision-Language Approaches for SDP

Despite their general-purpose visual reasoning strength, current MLLMs often underperform on domain-specific structured generation and detection tasks such as UI-to-code synthesis or visual defect detection. This gap stems from a lack of inductive biases for spatial layout reasoning, hierarchical planning, and code structuring — capabilities that are critical for front-end software engineering tasks but not explicitly optimized for during general MLLM pre-training. Modular multimodal agent architectures (e.g., ScreenCoder) that decompose the task into specialized sub-modules show improved performance over monolithic MLLM prompting.

Outlook: Visual Modality in Defect Prediction

Visual modalities remain the least mature dimension of multimodal defect prediction compared to code-text-structure fusion, but are rapidly advancing alongside general MLLM capability growth. Expect visual defect detection to first mature for front-end/UI-heavy codebases before generalizing to backend logic, where structural and data-flow modalities will likely remain dominant.

VI. JUST-IN-TIME MULTIMODAL DEFECT PREDICTION

6.1 Why JIT Prediction Is Naturally Multimodal

Just-in-time defect prediction operates at the commit level, immediately after a developer submits a code change — making it naturally suited to multimodal analysis since a commit inherently bundles multiple data types: the code diff itself (programming language modality), the commit message (natural language modality), and derivable hand-crafted metrics such as lines changed, files touched, and developer experience (tabular/numerical modality).

6.2 Inserting Expert Knowledge into Deep Learning Pipelines

A key research question in multimodal JIT prediction is how to optimally insert expert knowledge (hand-crafted metrics) into deep-learning-based approaches that primarily process commit logs and code changes. Studies bridging this gap show that treating expert knowledge as a distinct modality to be fused — rather than discarding it in favor of purely learned semantic representations, or relying on it exclusively without semantic context — produces the most accurate predictions, since hand-crafted features and

© JUL 2026 | IRE Journals | Volume 10 Issue 1 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I1-1719451

learned semantic features present different, complementary aspects of commit risk.

6.3 Bimodal Change Representation Learning

Recent JIT approaches explicitly model code changes via bimodal representation: jointly encoding the 'before' and 'after' states of modified code alongside the diff operations themselves (insertions, deletions, replacements). This bimodal change representation captures not just what the code looks like, but how it changed — a temporal-structural modality distinct from static code snapshots, and one that has shown particular value for detecting regression-prone modifications.

VII. BENCHMARKS & EVALUATION

7.1 Datasets Used in Multimodal SDP Research

Dataset	Modalities Available	Domain	Scale	Common Use
PROMISE Repository	Metrics + source code	Java, multi-project	~10K files, 40+ projects	Metric + semantic fusion baselines
Devign	Code (LCS) + CPG-derivable	C function-level vulnerabilities	~27K functions	Vulnerability detection fusion
ReVeal	Code + CPG	C/C++ vulnerabilities	Real-world CVE-linked	Vulnerability fusion evaluation
DiverseVul / SVulD	Code + CPG, diverse CWE types	Multi-language vulnerabilities	Large-scale	Complex/heterogeneous fusion testing
Bench4BL / Bugs.jar	AST + bug report text + patch	Java bug repair	Multi-project bug-fix pairs	Repair pattern, dual-branch fusion
JIT commit datasets (Kamei-style)	Metrics + commit msg + diff	Multi-project commit history	Tens of thousands of commits	JIT multimodal fusion
SWE-Bench Multimodal	Code + UI screenshots/mockups	Visual SE tasks	Growing benchmark	Vision-language SE evaluation

7.2 Evaluation Considerations Specific to Multimodal Models

- Ablation by modality: Reporting performance with each modality removed individually is essential to demonstrate genuine fusion benefit rather than one dominant modality carrying the result
- Modality contribution analysis: Attention weight visualization or SHAP-style attribution per modality helps interpret which modality drives which predictions
- Missing modality robustness: Real-world deployment often has incomplete data (e.g., commits without messages); evaluating graceful degradation is critical
- Standard metrics: AUC-ROC, F1, MCC, and Popt@20% remain the primary metrics, applied identically to multimodal and unimodal baselines for fair comparison

VIII. KEY CHALLENGES & LIMITATIONS

8.1 Modality Alignment

Different modalities operate at different granularities and exist in fundamentally different representational spaces — token sequences are linear, ASTs are hierarchical trees, CFGs are directed graphs, and metrics are scalar values. Aligning these into a shared representation space without losing modality-specific information is a core technical challenge. Poor alignment can cause one modality to dominate the fused representation, effectively negating the benefit of multimodality.

8.2 Information Loss in Intermediate Conversions

Converting code to AST and back during multimodal pipeline construction can cause irreversible information loss, yielding impoverished representations even when multiple modalities are nominally present. This underscores that simply having multiple input modalities is insufficient — the

fusion architecture must preserve, not discard, modality-specific information through the pipeline.

8.3 Computational Overhead

Multimodal architectures — particularly those using cross-attention or contrastive pre-training — incur substantially higher computational cost than unimodal baselines, due to the need to process and align multiple parallel encoders. This creates a practical trade-off for production deployment: cross-attention fusion may achieve marginally higher F1 scores than concatenation but at significantly higher inference latency, which matters for real-time CI/CD quality gates.

8.4 Conditional, Not Universal, Benefit

As demonstrated by FusionVul's ablation results, multimodal fusion does not universally outperform strong unimodal baselines — its benefit is conditional on the dataset exhibiting genuine complementary signal across modalities. When one modality's embeddings already achieve high separability for a given task, added modalities can introduce noise rather than signal, occasionally degrading performance relative to a well-tuned single-modality model.

8.5 Missing Modality Handling

Real-world software repositories frequently have incomplete data: commits without descriptive messages, legacy code without available metric history, or files without associated bug reports. Multimodal models must handle missing modalities gracefully — through architectural choices like late fusion (which degrades gracefully) or learned imputation mechanisms — rather than failing or requiring complete data for every prediction.

8.6 Class Imbalance Compounded Across Modalities

The class imbalance problem inherent to defect prediction (5-20% defective at file level) is compounded in multimodal settings, where each modality may have different missing-data patterns correlated with the defect label itself (e.g., poorly documented commits being both less likely to have rich commit messages and more likely to be defective), introducing subtle confounds that complicate both model training and evaluation.

Challenge	Severity	Primary Mitigation Approach	Research Maturity
Modality Alignment	High	Cross-attention gating, learned projection layers	Active
Information Loss in Conversion	Medium-High	Structural attention, direct multi-view encoding	Active
Computational Overhead	Medium	Efficient attention variants, distillation	Active
Conditional Fusion Benefit	Medium	Per-dataset ablation, adaptive fusion gating	Early
Missing Modality Robustness	Medium-High	Late fusion, modality dropout training, imputation	Active
Compounded Class Imbalance	Medium	Modality-aware resampling, cost-sensitive fusion	Early
Interpretability Across Modalities	Medium	Per-modality attention visualization, SHAP	Early

IX. INDUSTRY TOOLS & ADOPTION

9.1 Commercial and Open-Source Tooling

While dedicated commercial multimodal defect prediction products remain less mature than single-modality static analysis tools, the underlying technology is increasingly embedded within broader AI-assisted development platforms. Code review tools are beginning to fuse code diffs with commit message analysis and historical metrics for risk scoring, reflecting JIT multimodal research findings in production contexts.

Tool / Platform Category	Modalities Fused	Integration Point	Maturity
AI-assisted code review (e.g., Copilot-style)	Code diff + NL comments/messages	PR review	Maturing
Static analysis + ML hybrid (SonarQube-class)	Metrics + rule-based + some semantic	CI pipeline	Mature (metric+semantic)
Vulnerability scanners (research-grade)	CPG + sequential code (FusionVul-style)	Security pipelines	Active research → early production
Front-end visual regression tools	Code + UI screenshots	Visual testing pipelines	Early (MLLM-driven)
Custom enterprise JIT gates	Metrics + commit msg + diff	Commit hook / CI webhook	Active research → pilots

X. FUTURE RESEARCH DIRECTIONS

10.1 Unified Multimodal Foundation Models for Code

Following the trajectory of contrastive pre-training frameworks like CODE-MVP and multimodal foundation models like UniXCoder, the field is moving toward unified foundation models pre-trained jointly across code, structure, natural language, and eventually visual modalities, from which defect prediction (and other downstream tasks) can be fine-tuned with minimal task-specific architecture. This mirrors the trajectory of general-purpose multimodal LLMs in the broader AI field.

10.2 Adaptive and Learned Fusion Strategies

Rather than fixing a single fusion architecture (concatenation, cross-attention, etc.) for all inputs, future systems are likely to adopt adaptive fusion that learns, per-instance, which modalities to weight most heavily — extending the cross-attention gating concept of GMCA-SDP toward fully dynamic, context-aware modality selection that can also detect and compensate for missing or unreliable modalities at inference time.

10.3 Visual Modality Maturation

As vision-language models continue to improve and benchmarks like SWE-Bench Multimodal mature, expect visual modalities (UI screenshots, architecture diagrams, sequence diagrams) to become a standard third or fourth modality in defect prediction pipelines — particularly for front-end-heavy and mobile application codebases where visual-functional mismatches are a significant defect category not addressed by code-only analysis.

10.4 Multimodal Explainability

As multimodal models grow more complex, explaining predictions in terms of modality-specific contributions becomes essential for developer trust. Future work will likely formalize per-modality attribution techniques (extending SHAP and attention visualization to multi-encoder architectures) so that a flagged defect can be explained not just as 'high risk' but as 'high risk primarily due to control-flow complexity, secondarily due to sparse commit documentation.'

10.5 Efficient Multimodal Architectures for Real-Time Use

Given the computational overhead challenge identified in Section 8.3, an important research direction is developing efficient multimodal fusion architectures — via knowledge distillation, modality-specific pruning, or lightweight cross-attention variants — suitable for real-time CI/CD integration without the latency cost of full multi-encoder pipelines.

10.6 Cross-Project and Cross-Language Multimodal Transfer

Combining multimodal fusion with cross-project and cross-language defect prediction remains underexplored. Since different modalities may transfer differently across project/language boundaries (e.g., structural AST patterns may generalize better than lexical token patterns across languages), multimodal architectures offer a natural mechanism for modality-aware transfer learning that could substantially improve cross-project

generalization compared to single-modality transfer approaches.

Research Direction	Time Horizon	Key Enabler	Expected Impact
Unified multimodal foundation models	Mid-term (2027-2028)	Large-scale contrastive pre-training, compute scaling	High — reduces task-specific engineering
Adaptive/learned fusion strategies	Near-term (2026-2027)	Dynamic gating networks, meta-learning	High — robustness to data variability
Visual modality maturation	Mid-term (2027-2029)	Vision-language model improvement, SWE-Bench MM	Medium — front-end defect coverage
Multimodal explainability	Near-term (2026-2027)	Per-modality SHAP/attention extensions	High — developer trust & adoption
Efficient real-time architectures	Near-term (2026)	Distillation, pruning, lightweight attention	High — production CI/CD viability
Cross-project/language multimodal transfer	Mid-term (2027-2028)	Modality-aware domain adaptation	Medium-High — generalization

XI. CONCLUSION

Multimodal deep learning has emerged as a powerful paradigm for software defect prediction, addressing the fundamental limitation that no single code representation — lexical, structural, metric-based, or natural language — fully captures the multifaceted nature of software defects. The research surveyed in this report demonstrates that fusion of complementary modalities, when properly architected, consistently improves predictive performance over unimodal baselines: cross-attention gating mechanisms like GMCA-SDP successfully balance traditional metrics with deep semantic features; structural fusion frameworks like FusionVul achieve their strongest gains precisely on the most complex and heterogeneous vulnerability datasets; and JIT prediction research confirms that hand-crafted metrics, commit messages, and code diffs each carry distinct, complementary risk signals.

At the same time, this survey identifies important nuances that temper unconditional enthusiasm for multimodality: fusion benefit is conditional rather than universal, information can be lost during intermediate modality conversions if not carefully architected, computational overhead is non-trivial, and missing-modality robustness requires deliberate architectural planning. The most mature multimodal pairings — semantic-structural (code + AST/CPG) and metric-semantic — have produced validated, reproducible gains. The visual/vision-language modality remains comparatively immature but is advancing rapidly alongside broader multimodal LLM progress.

Looking forward, the trajectory points toward unified multimodal foundation models for code, adaptive per-instance fusion strategies, and richer explainability across modalities — converging with parallel developments in agentic AI and LLM-augmented software engineering. Researchers and practitioners investing in robust modality alignment, efficient fusion architectures, and rigorous per-modality ablation will be best positioned to advance this promising but still-maturing field.

REFERENCES

[1] A Cross-Attention Gating Mechanism-Based Multimodal Feature Fusion Method for Software Defect Prediction (GMCA-SDP). (2025). Applied Sciences (MDPI), 15(20), 11259.

[2] FusionVul: A Multimodal Feature Fusion Framework for Source Code Vulnerability Detection. (2025/2026). arXiv:2606.08553.

[3] Defect Detection in Source Code via Multimodal Feature Fusion. (2025). Applied Sciences (MDPI), 15(17), 9358.

[4] Bridging Expert Knowledge with Deep Learning Techniques for Just-In-Time Defect Prediction. (2024). arXiv:2403.11079.

[5] Just-In-Time Software Defect Prediction via Bi-modal Change Representation Learning. (2024). arXiv:2410.12107.

[6] Wang, C., et al. (2022). CODE-MVP: Learning to Represent Source Code from Multiple Views with Contrastive Pre-Training. NAACL 2022 Findings. arXiv:2205.02029.

[7] Guo, D., et al. (2022). UniXcoder: Unified Cross-Modal Pre-training for Code Representation. ACL 2022.

[8] Multi-View Feature Fusion Model for Software Bug Repair Pattern Prediction (PatternNet). (2023). Wuhan University Journal of Natural Sciences, 28(6), 493-508.

[9] Abdu, A., et al. Hierarchical CNN Approach for AST/CFG/DDG Multimodal Software Defect Prediction. Journal of Systems and Software.

[10] Investigation and Research on Several Key Issues of Software Defect Prediction. (2025). IET Software.

[11] Zou, Y., Wang, H., Lv, H., Zhao, S. (2025). Deep Learning-Based Cross-Project Defect Prediction: A Comprehensive Survey. QRS-C 2025.

[12] SWE-Bench Multimodal: Evaluating Language Models on Visual Software Engineering Tasks. (2025). AI Benchmarks Compendium.

[13] ScreenCoder: Advancing Visual-to-Code Generation for Front-End Automation via Modular Multimodal Agents. (2025). arXiv:2507.22827.

[14] Si, C., et al. (2025). Design2Code: Benchmarking Multimodal Code Generation for Automated Front-End Engineering. NAACL 2025.

[15] Giray, G., Bennin, K.E., Köksal, Ö., Babur, Ö., Tekinerdogan, B. (2023). On the Use of Deep Learning in Software Defect Prediction. Journal of Systems and Software, 195, 111537.

[16] Omri, S. & Sinz, C. (2020). Deep Learning for Software Defect Prediction: A Survey. ICSE 2020 Workshops.