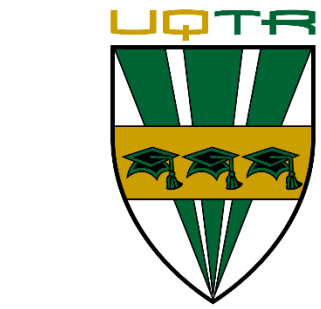


Université du Québec
à Trois-Rivières

UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À

L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN MATHEMATIQUE ET INFORMATIQUE APPLIQUÉES

PAR

MIREILLE KAFANDO

Analyse comparative des approches de prédiction de fautes logicielles basées sur les architectures CNN et Transformers

PROFESSEURS :

- **MOURAD BADRI**
- **FADEL TOURE**

FEVRIER 2026

Abstract

This master's thesis presents a comprehensive comparative analysis of preprocessing strategies and deep learning architectures for software defect prediction using source code metrics. We systematically evaluate approximately 45 configurations across 11 Java projects from the PROMISE corpus (2007–2010). Four research questions are investigated: (RQ1) the impact of preprocessing strategies, (RQ2) the comparative performance of deep learning architectures, (RQ3) the transferability between Within-Project Defect Prediction (WPDP) and Cross-Project Defect Prediction (CPDP) contexts, and (RQ4) the statistical significance of observed differences.

The empirical findings highlight three major observations. First, a marked convergence of performance suggests the presence of an intrinsic plateau. This likely reflects the informational limits of static CK metrics rather than purely algorithmic constraints. Second, architectural simplicity proves competitive, with CNN1D models exhibiting greater stability than Transformer architectures in cross-project settings, consistent with Occam's razor principle. Third, theoretical expectations anticipated systematic performance degradation when moving from within-project to cross-project prediction. Instead, substantial homogeneity is observed across both experimental frameworks.

Statistical analyses include Friedman tests and variance decomposition (ANOVA). They indicate that project-specific characteristics explain 77% of performance variance in within-project contexts, compared to 23% for methodological choices. This dominance evolves notably during cross-project transfer, with methodological choices gaining in importance by a factor of 5.7. The absence of statistically significant differences across most primary comparisons does not constitute experimental failure. Rather, it documents current boundaries of predictability from static code metrics. Given the limited corpus size, any potential differences remain strongly conditioned by inter-project variability.

This work contributes at three levels. Methodologically, it proposes a unified preprocessing pipeline with rigorous statistical validation, supporting reproducibility in the field. Theoretically, it demonstrates the applicability of the parsimony principle to

defect prediction and analyzes how contextual dominance evolves between within- and cross-project settings. Practically, it recommends prioritizing architectural simplicity, adapting approaches to project-specific profiles, and investing in contextual characterization over marginal algorithmic refinement.

This research contributes to the maturation of software defect prediction by explicitly documenting effective and ineffective strategies. It guides future work toward qualitative feature enrichment and context-aware adaptation, rather than incremental architectural complexity.

Keywords: software defect prediction, deep learning, CNN, Transformers, preprocessing strategies, class imbalance, Within-Project Defect Prediction (WPDP), Cross-Project Defect Prediction (CPDP), statistical validation, reproducibility.

RÉSUMÉ

Ce mémoire de maîtrise présente une analyse comparative approfondie des stratégies de prétraitement et des architectures d'apprentissage profond pour la prédiction de défauts logiciels à partir de métriques de code source. Nous évaluons systématiquement environ 45 configurations sur 11 projets Java du corpus PROMISE (2007–2010). Quatre questions de recherche sont examinées : (RQ1) l'impact des stratégies de prétraitement, (RQ2) la performance comparative des architectures d'apprentissage profond, (RQ3) la transférabilité entre les contextes de prédiction intra-projet (Within-Project Defect Prediction, WPDP) et inter-projets (Cross-Project Defect Prediction, CPDP), et (RQ4) la significativité statistique des différences observées.

Les résultats empiriques mettent en évidence trois constats majeurs. Premièrement, une convergence marquée des performances suggère l'existence d'un plateau intrinsèque. Celui-ci reflète possiblement les limites informationnelles des métriques CK statiques plutôt que des contraintes purement algorithmiques. Deuxièmement, la simplicité architecturale se révèle compétitive : les modèles CNN1D présentent une stabilité supérieure aux architectures Transformer en contexte inter-projets, en cohérence avec le principe de parcimonie d'Occam. Troisièmement, les attentes théoriques anticipaient une dégradation systématique lors du passage du contexte intra-projet au contexte inter-projets. Or, nous observons au contraire une homogénéité substantielle des performances entre ces deux cadres expérimentaux.

Les analyses statistiques incluent des tests non paramétriques de Friedman et une décomposition de variance (ANOVA). Elles indiquent que les caractéristiques propres aux projets expliquent 77 % de la variance en contexte intra-projet, contre 23 % pour les choix méthodologiques. Cette dominance évolue sensiblement lors du transfert inter-projets, les choix méthodologiques gagnant en importance d'un facteur 5,7. L'absence de différences significatives dans la majorité des comparaisons ne constitue pas un échec expérimental. Elle documente plutôt les frontières actuelles de la prédictibilité à partir de métriques statiques. Compte tenu de la taille limitée du corpus, toute différence potentielle demeure fortement conditionnée par la variabilité inter-projets.

Les contributions de ce travail se situent à trois niveaux. Sur le plan méthodologique, il propose un pipeline de prétraitement unifié assorti d'une validation statistique rigoureuse, contribuant à la reproductibilité du domaine. Sur le plan théorique, il met en évidence l'applicabilité du principe de parcimonie à la prédiction de défauts logiciels et analyse l'évolution de la dominance des facteurs contextuels entre contextes intra- et inter-projets. Sur le plan pratique, il recommande de privilégier la simplicité architecturale, d'adapter les approches aux profils spécifiques des projets et d'investir dans la caractérisation contextuelle plutôt que dans un raffinement algorithmique marginal.

Cette recherche contribue à la maturation des techniques de prédiction de défauts logiciels en documentant explicitement ce qui fonctionne et ce qui ne fonctionne pas. Elle oriente les travaux futurs vers l'enrichissement qualitatif des caractéristiques et l'adaptation au contexte, plutôt que vers une complexification architecturale incrémentale.

Mots-clés : prédiction de défauts logiciels, apprentissage profond, CNN, Transformers, stratégies de prétraitement, déséquilibre de classes, Within-Project Defect Prediction (WPDP), Cross-Project Defect Prediction (CPDP), validation statistique, reproductibilité.

*À mes parents **Luc** et **Aline**, pour votre amour inconditionnel et vos sacrifices,*

*À ma sœur **Nadège**, pour ta présence et ton soutien constant,*

*À la mémoire de mon petit frère **Yves**, qui reste à jamais dans nos cœurs.*

Je tiens à exprimer ma profonde gratitude envers mes directeurs de recherche, les Professeurs Mourad Badri et Fadel Touré, pour leur encadrement rigoureux, leur disponibilité constante et leurs précieux conseils tout au long de ce projet. Leur patience face à mes questionnements méthodologiques et leur encouragement à adopter une approche de transparence radicale ont été déterminants dans l'orientation et la qualité de cette recherche. Leurs commentaires toujours constructifs m'ont permis d'affiner ma réflexion et de développer un regard critique essentiel au travail de chercheur.

Je souhaite remercier l'Université du Québec à Trois-Rivières et le Département de mathématiques et d'informatique pour les ressources mises à ma disposition et pour le soutien financier qui m'a été accordé. Ce soutien institutionnel a été essentiel pour mener à bien cette recherche dans de bonnes conditions.

Un remerciement spécial s'adresse aux créateurs et mainteneurs du corpus PROMISE, dont les données ouvertes et bien documentées constituent la pierre angulaire de nombreuses recherches en prédiction de défauts logiciels, dont la nôtre. Leur engagement envers la science ouverte et la reproductibilité est exemplaire.

Je souhaite également remercier mes collègues de la maîtrise, en particulier les membres du laboratoire de recherche en génie logiciel, pour les échanges enrichissants, les discussions stimulantes et le soutien mutuel tout au long de ce parcours. Votre camaraderie et votre entraide ont grandement contribué à créer un environnement d'apprentissage motivant et collaboratif.

Sur un plan plus personnel, je tiens à exprimer ma reconnaissance infinie envers ma famille. À mes parents, pour avoir toujours cru en moi et pour m'avoir transmis l'importance de l'éducation et de la persévérance. Votre soutien indéfectible, même à distance, a été ma plus grande source de motivation. À ma sœur, pour ta présence constante et tes encouragements qui m'ont aidée à traverser les moments de doute.

Je pense également à mon petit frère, dont le souvenir lumineux continue d'illuminer nos vies et dont la mémoire m'a accompagnée tout au long de ce parcours académique. Ta présence reste vivante dans nos cœurs et ton esprit continue de nous inspirer chaque jour.

Ma gratitude s'étend également à ma tante Kia et sa famille, dont le soutien et les encouragements ont été précieux. À tous mes proches qui ont cru en moi et m'ont soutenue tout au long de ce parcours, merci de votre présence.

Enfin, mes pensées vont vers amis ainsi que tous ceux qui, de près ou de loin, ont contribué à ce projet par leur soutien moral, leurs encouragements ou simplement leur compréhension durant les longues heures consacrées à cette recherche. Chacune de ces contributions, aussi modeste soit-elle, a compté.

À tous et à toutes, merci.

Avant-propos

Ce mémoire représente l'aboutissement d'un parcours de recherche qui a commencé par une fascination pour l'intelligence artificielle et une conviction profonde : celle que la qualité logicielle peut être améliorée par des approches prédictives intelligentes.

L'idée de cette recherche est née d'une observation personnelle. Malgré mon intérêt pour les avancées considérables en apprentissage profond ces dernières années, je constatais que le domaine de la prédiction de défauts logiciels semblait encore dominé par des approches classiques. Les architectures Transformers, qui ont révolutionné le traitement du langage naturel et la vision par ordinateur, m'apparaissaient comme une piste prometteuse à explorer dans notre domaine. Cette observation a suscité ma curiosité intellectuelle : ces architectures sophistiquées pourraient-elles réellement améliorer notre capacité à prédire les défauts logiciels ?

Table des matières

Table des tableaux

Table des figures

Table des Annexes

CHAPITRE 1

Introduction

1.1. Contexte

La qualité logicielle constitue un enjeu majeur, les systèmes numériques supportant aujourd'hui des infrastructures critiques telles que la santé, l'énergie, le transport et la finance. L'incident technologique mondial du 19 juillet 2024 a montré l'ampleur des conséquences économiques, sociales et sécuritaires que peut avoir la défaillance d'un composant logiciel. La défaillance liée à une mise à jour du capteur CrowdStrike Falcon a affecté environ 8,5 millions de systèmes Windows, perturbant temporairement plusieurs services essentiels, notamment dans l'aviation, la santé et la finance [1]. Les pertes économiques, estimées à plusieurs milliards de dollars en quelques jours, soulignent l'importance cruciale de la prédiction et de la prévention des défauts logiciels.

Au-delà de ces événements spectaculaires, la réalité économique de la qualité logicielle révèle un coût structurel considérable. Le Consortium for Information & Software Quality estime que la mauvaise qualité logicielle a coûté au moins 2,41 billions de dollars US à l'économie américaine en 2022 [2], soit 2 410 milliards de dollars. Ce montant dépasse la masse salariale du secteur informatique américain (1,51 billion de dollars) et représente environ 10 % du PIB. Cette estimation englobe non seulement les défaillances opérationnelles directes, mais également les coûts indirects liés à la maintenance corrective, aux pertes de productivité et à la dette technique accumulée.

Historiquement, la « crise du logiciel » des années 1960 avait déjà révélé l'incapacité des méthodes de développement classiques à maîtriser la complexité croissante des systèmes. Depuis, de nombreux paradigmes méthodologiques se sont succédé : programmation structurée, programmation orientée objet, méthodes agiles et intégration continue. En parallèle, des approches organisationnelles comme DevOps ont également émergé. Centrées sur l'automatisation du déploiement et la collaboration entre équipes, elles répondent aux exigences de livraison continue. Malgré ces évolutions, aucun modèle

organisationnel ou technique n'a permis d'éliminer totalement le risque de défauts logiciels.

La maintenance logicielle demeure une phase prépondérante, représentant 40 à 60 % du coût total de son cycle de vie, dont une part substantielle est consacrée à la localisation et à la correction des défauts. Y. Zhang et al. [3], et A. Alsaeedi et al. [4], confirment que cette répartition asymétrique persiste même dans les organisations adoptant des pratiques modernes. Cette situation révèle un paradoxe économique : alors que la détection précoce coûte significativement moins que la correction en production, les organisations continuent d'investir massivement dans les activités curatives plutôt que préventives.

C'est dans ce contexte que la prédiction de défauts logiciels (Software Defect Prediction, SDP) s'est imposée comme une approche proactive visant à identifier en amont, les modules les plus susceptibles de contenir des défauts. Les travaux de S. Lessmann et al. [5] ont établi les bases méthodologiques en évaluant 22 classifieurs sur 10 ensembles de données (datasets), démontrant qu'aucun algorithme ne domine universellement. Y. Kamei et al. [6], dans leur étude longitudinale sur six systèmes comprenant plus de 400 000 changements de code, ont démontré l'efficacité des approches Just-In-Time, atteignant 68% de rappel tout en réduisant les efforts d'inspection de 35%.

L'avènement de l'apprentissage profond a ouvert de nouvelles perspectives. Les architectures convolutionnelles unidimensionnelles (CNN1D) offrent l'avantage d'identifier automatiquement des patterns complexes dans les séquences de métriques, tandis que les Transformers avec leur mécanisme d'auto-attention, permettent de modéliser des dépendances à longue portée. C. Pan et al. [7] et S. Sahar et al. [8] ont démontré le potentiel de CodeBERT pour la prédiction de défauts, combinant représentations sémantiques du code source et métriques structurelles. Cette évolution technologique révèle cependant des lacunes méthodologiques persistantes qui compromettent l'adoption industrielle des modèles de prédiction et motivent directement notre recherche.

1.2. Problématique

Malgré ces avancées, plusieurs limites méthodologiques demeurent et freinent encore l'adoption industrielle des modèles de prédiction.

- **Déséquilibre des classes**

Les jeux de données publics, comme le corpus PROMISE, souffrent d'un déséquilibre marqué où les observations non défectueuses dépassent souvent 90 %. Ce problème, formalisé par H. He et E. Garcia [9], biaise les modèles vers la classe majoritaire et nuit à la détection des défauts. La recherche de solutions stables est donc primordiale. S. Feng et al. [10] montrent alors que les techniques courantes de sur-échantillonnage, comme SMOTE, introduisent une instabilité significative dans les performances en raison de leur caractère aléatoire. Cette complexité est corroborée et mise en perspective par S. Hosseini et al. [11], qui identifie le déséquilibre des classes comme l'un des défis principaux et non résolus pour les approches modernes de prédiction, notamment celles fondées sur l'apprentissage profond. Notre étude intègre cette problématique fondamentale en évaluant rigoureusement l'impact des stratégies de préparation des données dans notre protocole expérimental.

- **Variabilité intra-projet et inter-projets.**

La généralisation des modèles de prédiction de défauts se heurte à la variabilité intrinsèque des données, qui se manifeste suivant deux axes indissociables. Au niveau temporel, les distributions de métriques évoluent entre les versions successives d'un même projet, une volatilité documentée par S. McIntosh et Y. Kamei [12] révélant comment les caractéristiques logicielles fluctuent au fil du cycle de vie. Au niveau transversal, ces distributions divergent radicalement d'un projet à l'autre, comme l'ont établi T. Zimmermann et al. [13]. Cette dualité conduit à une dérive conceptuelle : les patterns appris perdent progressivement en validité et en transférabilité. D. Lin et al. [14] confirment que ces mouvements constituent des propriétés structurelles incontournables des écosystèmes logiciels. Toute architecture prédictive doit donc intégrer des mécanismes d'adaptation capables de composer avec cette instabilité structurelle.

- **Prétraitement et standardisation**

L'impact du prétraitement des données est solidement établi. Les travaux de C. Tantithamthavorn et al. [15] ont démontré de manière empirique que des décisions comme le rééquilibrage des classes peuvent influencer les performances d'un modèle de prédiction plus fortement que le choix de l'algorithme de classification lui-même. Cette observation fondamentale s'étend aux choix de normalisation (globale, par version, robuste) et de gestion des valeurs aberrantes, qui conditionnent la capacité d'un modèle à généraliser, particulièrement dans le contexte exigeant de la prédiction inter-projets (CPDP). Une étude récente de A. Vescan et al. [16] vient confirmer ce rôle central en explorant spécifiquement l'impact de diverses techniques de prétraitement, dont la normalisation, sur l'efficacité des algorithmes de classification composite en CPDP. Ces recherches mettent en lumière les interactions complexes entre la préparation des données et les architectures des modèles, justifiant pleinement l'attention méthodique que nous portons à cette étape dans notre comparaison expérimentale des architectures CNN1D et Transformer.

- **Limites des modèles profonds et reproductibilité**

L'enthousiasme initial suscité par les architectures profondes (CNN, Transformers) en SDP se heurte à des limites méthodologiques significatives. Les synthèses de J. Pachouly et al. [17] et G. Giray et al. [18] révèlent un paradoxe : alors que la complexité des modèles s'accroît, la rigueur des protocoles de validation ne suit pas. Les comparaisons architecturales reposent fréquemment sur des configurations expérimentales hétérogènes, sans standardisation des métriques ni validation statistique robuste des différences observées. G. Giray et al. [18] rapportent ainsi que la majorité des études omettent de publier leur code source, rendant impossible la vérification indépendante des résultats. Ce problème de reproductibilité soulève des questions fondamentales sur la validité des gains de performance revendiqués. Notre travail s'inscrit en rupture avec ces pratiques en adoptant une transparence totale sur l'ensemble de notre protocole expérimental.

Face à ces défis méthodologiques documentés, notre recherche s'articule autour d'une problématique centrale intégrative : Comment concevoir et évaluer des pipelines de SDP combinant stratégies de prétraitement et architectures profondes (CNN1D, Transformers) dans un cadre rigoureux, de façon à garantir des performances robustes tant en configuration intra-projet (WPDP) qu'inter-projets (CPDP), malgré le déséquilibre des classes et l'hétérogénéité des données ?

1.3. Objectifs

Notre objectif est de concevoir des pipelines intégrés qui combinent des stratégies de prétraitement et des architectures modernes d'apprentissage profond pour améliorer la SDP, d'abord à partir de métriques de code, puis du code source brut. Nous analysons systématiquement l'influence des choix méthodologiques sur la robustesse, la précision et la généralisabilité des modèles.

Plus précisément, quatre objectifs spécifiques structurent notre démarche scientifique :

- O1. Prétraitement des données. Évaluer l'impact des stratégies de standardisation et des techniques d'équilibrage sur la stabilité et la performance des prédictions. S. Hosseini et al. [11] confirment que la normalisation des métriques peut davantage impacter les performances que le choix de l'algorithme en prédiction inter-projets.
- O2. Architectures d'apprentissage profond. Comparer trois familles de modèles neuronaux exploitant un gradient croissant de pré-entraînement : CNN1D sans pré-entraînement, Transformer encodeur sans pré-entraînement, et architectures multimodales pré-entraînées intégrant métriques et code source.
- O3. Configurations expérimentales. Mesurer la généralisabilité des modèles dans deux scénarios : Within-Project Defect Prediction (WPDP) et Cross-Project Defect Prediction (CPDP). T. Zimmermann et al. [13] démontrent que la CPDP constitue un défi majeur même entre projets similaires.
- O4. Validation statistique et reproductibilité. Évaluer la significativité statistique des différences observées pour distinguer les effets réels des fluctuations

aléatoires, conformément aux standards de J. Demšar [19] et aux recommandations de reproductibilité de M. Heil et al. [20].

1.4. Questions de recherche

Les objectifs ci-dessus se déclinent en quatre questions de recherche (RQ) qui structurent notre démarche, nos expérimentations et orientent nos analyses empiriques :

- RQ1 — Prétraitement

  Quel est l'impact de la standardisation, des techniques d'équilibrage et de la suppression des valeurs aberrantes sur la performance des modèles?

- RQ2 — Choix architectural

  Les architectures Transformer (encodeur seul sans pré-entraînement, architectures multimodales avec gradient de pré-entraînement) surpassent-elles les CNN1D dans les différents contextes de données et de protocoles ?

- RQ3 — Transférabilité

  Quelle est la variation de performance entre les configurations WPDP et CPDP pour chaque combinaison prétraitement × architecture ?

- RQ4 — Validation statistique

  Dans quelle mesure les différences observées entre pipelines et architectures sont-elles statistiquement significatives ? Quelles tailles d'effet caractérisent ces variations ?

Ces quatre questions assurent la cohérence entre les enjeux identifiés et nos choix méthodologiques. Nous adoptons une posture de transparence radicale : nos analyses documenteront exhaustivement nos observations, qu'elles confirment ou infirment nos hypothèses initiales, en quantifiant précisément les limites de nos conclusions.

1.5. Contributions

1.5.1. Contributions originales

Ce mémoire apporte des contributions sur trois dimensions complémentaires qui répondent directement aux lacunes identifiées dans la littérature et opérationnalisent les objectifs O1-O4.

Nous fournissons un pipeline de prétraitement unifié combinant trois stratégies de standardisation et six techniques d'équilibrage avec détection de valeurs aberrantes. Cette approche est évaluée sur 11 projets Java du corpus PROMISE.

Nous comparons cinq architectures explorant un gradient de complexité : deux variantes CNN1D, un Transformer Encodeur non-pré-entraîné, une architecture hybride multimodale, et une architecture entièrement pré-entraînée. Face aux contraintes computationnelles, nous avons évalué 45 configurations représentatives parmi les 120 théoriquement possibles. Comme le soulignent J. Dodge et al. [21], documenter les compromis entre exhaustivité et ressources constitue une pratique essentielle pour la reproductibilité. Le biais d'optimisme potentiel sera explicitement documenté au Chapitre 6.

Nous appliquons un protocole statistique rigoureux combinant ANOVA à mesures répétées à trois facteurs pour décomposer les sources de variabilité, et le test de Friedman pour comparer les rangs moyens, conformément aux standards de J. Demšar [19] pour les comparaisons d'algorithmes. Les tailles d'effet sont quantifiées via Kendall's W, et les comparaisons multiples corrigées via Bonferroni ou Holm-Bonferroni.

- **Contributions théoriques**

Au-delà de nos résultats empiriques spécifiques, notre recherche contribue à la compréhension théorique du domaine en documentant systématiquement les interactions entre choix de prétraitement et architectures neuronales. Nous quantifions la dominance relative des facteurs contextuels (caractéristiques intrinsèques des projets) versus les facteurs méthodologiques (choix algorithmiques) dans la variance des performances

observées. Ce résultat remet en question l'hypothèse implicite selon laquelle l'optimisation méthodologique constitue le levier principal d'amélioration.

Notre analyse statistique permet également de distinguer les différences robustes des variations aléatoires, contribuant à établir des conclusions plus fiables. ous documentons nos observations qu'elles confirment ou infirment les hypothèses initiales. L'absence de significativité statistique délimite les bénéfices réels de la complexité algorithmique plutôt qu'elle ne signale un échec expérimental.

- **Contributions pratiques**

Nos résultats fournissent des recommandations concrètes : configurations de prétraitement les plus robustes, architectures les plus performantes selon le contexte (WPDP vs CPDP), et combinaisons adaptées à différents profils de projets.

Pour faciliter la validation indépendante de nos résultats, nous fournissons notre code source complet via un dépôt GitHub public, accompagné de notebooks documentés permettant l'exécution interactive des expérimentations. Les graines aléatoires sont fixées pour assurer la reproductibilité, et les versions des bibliothèques principales sont explicitement spécifiées. La communauté scientifique a développé des standards de reproductibilité tels que le cadre REFORMS documenté par S. Kapoor et al. [22] qui proposent notamment un archivage pérenne avec DOI, des tests automatisés, et une documentation exhaustive des métadonnées. Cependant, comme le précisent H. Semmelrock et al. [23], l'adoption de ces standards demeure limitée, probablement en raison de contraintes systémiques plutôt que d'un simple manque de rigueur individuelle. Notre contribution documente explicitement les choix, contraintes et limitations du processus expérimental, illustrant ainsi une approche pragmatique de la reproductibilité dans les conditions réelles de la recherche.

Ces contributions reposent sur une approche intégrée évaluant conjointement prétraitement, architecture et contexte de validation, plutôt qu'en optimisant chaque composante isolément.

1.5.2. Organisation du mémoire

Ces contributions se déploient sur six chapitres, dont la progression reflète le cheminement de notre investigation scientifique.

Chapitre 1 : Introduction. Présente le contexte économique et scientifique de la SDP, expose la problématique centrale liée aux lacunes méthodologiques du domaine, détaille les objectifs de recherche (O1 à O4) et les quatre questions de recherche (RQ1 à RQ4), puis résume l'ensemble des contributions méthodologiques, théoriques et pratiques.

Chapitre 2 : Fondements théoriques. Établit les bases conceptuelles nécessaires : hiérarchie faute-erreur-défaillance, métriques logicielles comme variables prédictives, problématique du déséquilibre des classes, justification de la moyenne géométrique (G-Mean) comme métrique principale, stratégies de prétraitement évaluées, justification du recours à l'apprentissage profond, et configurations expérimentales WPDP/CPDP.

Chapitre 3 : État de l'art. Propose une analyse critique de la littérature récente : évolution du domaine, comparaison des architectures profondes, identification des défis méthodologiques persistants, et positionnement nuancé de notre contribution.

Chapitre 4 : Méthodologie. Décrit le cadre méthodologique : corpus PROMISE, pipeline de prétraitement, architectures, protocole expérimental, métriques d'évaluation et analyses statistiques, avec documentation explicite des limitations.

Chapitre 5 : Résultats et discussion. Présente et interprète les performances expérimentales en répondant séquentiellement aux questions RQ1 à RQ4, avec tests statistiques pour distinguer effets réels et fluctuations aléatoires.

Chapitre 6 : Menaces à la validité et conclusion. Synthétise les apprentissages principaux, examine l'étendue et les limites de nos contributions, évalue les menaces à la validité selon la taxonomie de C. Wohlin et al. [24], et esquisse des perspectives de recherche futures.

Cette progression suit la logique de l'investigation scientifique, de la contextualisation théorique jusqu'à l'évaluation critique des résultats.

CHAPITRE 2 : FONDEMENTS DE LA PREDICTION DE FAUTES LOGICIELLES

Nous nous intéressons à la prédiction automatique de fautes logicielles, visant à identifier les modules susceptibles de contenir des défauts. Cette approche repose sur l'hypothèse qu'il existe des corrélations mesurables entre les caractéristiques du code source et la présence de fautes.

Ce chapitre établit les bases conceptuelles et techniques qui sous-tendent notre démarche, en lien direct avec les défis identifiés dans la problématique.

2.1. Fondements théoriques

2.1.1. Hiérarchie conceptuelle

Afin de délimiter précisément notre objet de recherche, nous adoptons la classification établie par A. Avizienis et al. [25] qui distingue trois concepts fondamentaux dans la chaîne causale des anomalies logicielles. Une faute (fault) représente une anomalie statique dans le code source, potentiellement dormante jusqu'à son activation. Lorsque cette faute est activée lors de l'exécution, elle produit une erreur (error), soit une déviation de l'état interne correct du système. Si cette erreur se propage jusqu'à l'interface observable du système, elle se manifeste comme une défaillance (failure), impactant directement l'utilisateur ou les systèmes dépendants.

Dans notre recherche, nous ciblons spécifiquement la prédiction des fautes au niveau du code source à partir de métriques logicielles, cherchant à identifier les modules susceptibles de contenir des fautes latentes plutôt que les défaillances observables.

L'identification des modules défectueux repose sur la norme IEEE Std 1044-2009 pour la classification des anomalies logicielles, facilitant la reproductibilité et la comparabilité avec d'autres études. Toutefois, comme le démontrent K. Herzig et al. [26] après analyse manuelle de plus de 7 000 rapports de signalement provenant de cinq projets open-source,

l'extraction automatique de labels à partir des systèmes de gestion de versions introduit un bruit substantiel. En effet, 33,8 % des « commits » classifiés comme corrections de bogues concernent en réalité des améliorations fonctionnelles, et 39 % des fichiers marqués comme défectueux n'ont jamais contenu de bogue réel. Ce bruit d'étiquetage introduit une limite supérieure aux performances atteignables, indépendamment des modèles utilisés.

2.1.2. Métriques logicielles

Les métriques logicielles constituent les variables explicatives de notre modélisation. Notre sélection s'appuie sur les critères établis par N. Fenton et J. Bieman [27] pour évaluer leur validité scientifique : validité de construction, fiabilité et utilité pratique. Nous utilisons uniquement des métriques statiques extraites du code source, regroupées en trois catégories reflétant les dimensions structurelle, cognitive et architecturale du système.

- Métriques de taille. Le nombre de lignes de code (LOC) quantifie la surface d'exposition aux défauts. H. Zhang [28] identifie LOC comme l'un des prédicteurs les plus significatifs dans 26 études empiriques, bien que la relation taille-défauts demeure débattue (linéaire vs logarithmique).
- Métriques de complexité. La complexité cyclomatique de T. McCabe [29] quantifie les chemins indépendants à travers le code et établit une borne supérieure pour les tests de couverture. Nous utilisons la complexité maximale (max_cc) et moyenne (avg_cc) par classe. A. Watson et T. McCabe [30] démontrent l'utilité de cette métrique pour identifier les modules critiques nécessitant une attention particulière.
- Métriques orientées-objet. La suite CK de S. Chidamber et C. Kemerer [31] (DIT, NOC, WMC, RFC, CBO, LCOM) révèle les interdépendances architecturales des systèmes orientés-objet. V. Basili et al. [32] valident empiriquement leur pouvoir prédictif pour identifier les points de propagation de défauts. Nous complétons cette suite avec des métriques OO additionnelles (CA, CE, NPM, LCOM3, DAM, MOA, MFA, CAM, IC, CBM, AMC) documentées dans la littérature spécialisée.

Tableau 1: Classification des métriques logicielles

Famille	Métriques	Description	Référence
Taille	LOC	Nombre de lignes de code	[27][28]
	NPM	Nombre de méthodes publiques	[27]
Complexité	max_cc	Complexité cyclomatique maximale	[29]
	avg_cc	Complexité cyclomatique moyenne	[29]
	WMC	Somme pondérée des complexités *(Weighted Methods per Class)*	[31]
Héritage	DIT	Profondeur d'héritage *(Depth of Inheritance Tree)*	[31]
	NOC	Nombre de sous-classes *(Number of Children)*	[31]
Couplage	CBO	Couplage entre objets *(Coupling Between Objects)*	[31]
	RFC	Ensemble des réponses *(Response For Class)*	[31]
	CA, CE	Couplage afférent et efférent	[27]
Cohésion	LCOM	Manque de cohésion *(Lack of Cohesion of Methods)*	[31]
	LCOM3	Manque de cohésion (version 3)	[27]
Orienté-objet	DAM	Ratio d'encapsulation des données	[27]
	MOA, MFA	Agrégation de méthodes et d'attributs	[27]
	CAM, IC, CBM, AMC	Métriques de cohésion et complexité OO	[27]

2.1.3. Problématique du déséquilibre

Le déséquilibre des données est un défi central de notre recherche.

Dans les projets logiciels, les modules fautifs représentent typiquement une minorité substantielle des instances, créant un biais d'apprentissage vers la classe majoritaire H. He

et E. Garcia [9]. Un modèle prédisant systématiquement « non-fautif » peut atteindre une exactitude élevée (>80%) tout en étant inutile pour la détection pratique des défauts, illustrant le paradoxe de l'exactitude trompeuse en contexte déséquilibré. Cette problématique influence directement nos choix de métriques d'évaluation. Comme le souligne G. Giray et al. [18], l'exactitude (*accuracy*) peut être trompeuse dans ce contexte. Nous privilégions des métriques adaptées aux données déséquilibrées : G-Mean, F1-macro, et *Accuracy*. Ces indicateurs, détaillés ultérieurement, offrent une vision équilibrée des performances sur les deux classes.

2.2. Préparation des données

2.2.1. Défi de la standardisation

Les échelles de mesure des métriques logicielles varient drastiquement : certaines métriques comme LOC peuvent atteindre plusieurs milliers d'unités, tandis que d'autres comme DIT ou NOC restent typiquement inférieures à 10. Cette hétérogénéité peut biaiser les algorithmes d'apprentissage automatique, particulièrement ceux sensibles aux échelles comme les réseaux de neurones. La standardisation vise à placer toutes les métriques sur une échelle comparable, facilitant la convergence des algorithmes tout en préservant les relations structurelles entre les variables.

2.2.2 Stratégies de standardisation

Si la nécessité d'une mise à l'échelle est établie, son application dans un contexte temporel soulève un défi supplémentaire : comment standardiser des données qui évoluent structurellement au fil des versions ? Deux impératifs s'opposent. Le premier est la cohérence globale de l'espace de représentation, qui facilite l'apprentissage de patterns stables. Le second est la fidélité aux spécificités locales de chaque version, dont les distributions peuvent varier sous l'effet des transformations continues que subissent les systèmes logiciels. Pour arbitrer cette tension, nous considérons trois stratégies distinctes.

La standardisation globale (SG) calcule les paramètres (μ, σ) sur l'ensemble complet d'entraînement, toutes versions confondues, puis applique uniformément la transformation Z-score. Postulant l'existence de paramètres stables indépendants des spécificités

temporelles, elle est simple à implémenter mais ne tient pas compte de ces transformations structurelles (restructurations, migrations architecturales).

À l'inverse, la standardisation par version (SV) traite chaque version comme une unité distributionnelle distincte, reflet des évolutions continues du système. Elle calcule des paramètres (μ, σ) et standardise chaque version séparément avant de concaténer les résultats, privilégiant l'adaptation locale à la cohérence globale.

Face aux avantages et limites respectifs de ces deux premières approches, une voie médiane a été explorée. La standardisation robuste séquentielle (SR) constitue une stratégie hybride combinant SV et SG via une double transformation séquentielle : (1) standardisation par version sur chaque ensemble, (2) concaténation, (3) seconde standardisation globale. La première couche capture les variations locales, tandis que la seconde assure une cohérence distributionnelle facilitant l'apprentissage. Cette approche représente un compromis entre adaptation contextuelle et invariance globale.

L'application de ces trois stratégies à nos architectures expérimentales, ainsi que les contraintes ayant conduit à leur utilisation asymétrique selon l'architecture est détaillée au Chapitre 4 (section 4.2.1). L'analyse comparative de leurs performances respectives, conduite au Chapitre 5, permettra de déterminer dans quelle mesure ce choix de standardisation influence effectivement les performances prédictives selon les contextes WPDP et CPDP.

2.2.3 Techniques d'équilibrage

Face au déséquilibre des classes documenté à la section 2.1.3, deux philosophies d'équilibrage s'opposent : le sur-échantillonnage, qui enrichit la classe minoritaire, et le sous-échantillonnage, qui réduit la classe majoritaire. Nous évaluons quatre stratégies relevant de ces deux approches, complétées pour deux d'entre elles par un nettoyage préalable des valeurs aberrantes. Le Tableau 2 en présente la synthèse ; leur implémentation opérationnelle est détaillée au Chapitre 4 (section 4.2.2).

Le SMOTE (Synthetic Minority Over-sampling Technique), proposé par N. Chawla et al. [33], est une méthode de sur-échantillonnage de la classe minoritaire qui génère des

instances synthétiques par interpolation entre un point et ses k-plus-proches voisins dans l'espace des caractéristiques. Cette approche préserve l'intégralité des données majoritaires, mais risque d'amplifier le bruit en présence de chevauchement inter-classes. Le sous-échantillonnage aléatoire adopte la philosophie inverse en réduisant la classe majoritaire par sélection uniforme. Il offre l'avantage d'une moindre complexité computationnelle, au prix d'une possible perte d'exemples informatifs aux frontières décisionnelles.

Ces deux techniques de base sont déclinées en variantes intégrant une détection puis nettoyage préalable des valeurs aberrantes. EllipticEnvelope est retenu en contexte WPDP et IsolationForest en contexte CPDP, afin d'améliorer la qualité des données avant équilibrage. A. Balogun et al. [34] montrent l'efficacité de telles combinaisons pour améliorer la performance en prédiction de défauts. L'évaluation comparative de ces quatre stratégies permettra d'isoler les contributions respectives du nettoyage et du rééquilibrage à la performance prédictive finale.

Tableau 2 : Techniques d'équilibrage évaluées

Technique	**Détection des valeurs aberrantes**	**Équilibrage**	**Justification**
SMOTE	—	Sur-échantillonnage synthétique (k-plus-proches voisins)	Préservation de l'intégralité des données majoritaires, génération d'exemples minoritaires synthétiques.
Sous-échantillonnage aléatoire	—	Réduction uniforme de la classe majoritaire	Simplicité, robustesse et réduction drastique du temps d'entraînement, recommandée en contexte de redondance.
SMOTE + nettoyage	EllipticEnvelope (WPDP) IsolationForest (CPDP)	SMOTE appliqué sur données nettoyées	Amélioration de la qualité des interpolations SMOTE en éliminant au préalable le bruit amplificateur.
Sous-échantillonnage + nettoyage	EllipticEnvelope (WPDP)	Sous-échantillonnage aléatoire après nettoyage	Recherche d'un compromis optimal entre qualité des données (nettoyage) et efficacité computationnelle (réduction).

Technique	Détection des valeurs aberrantes	Équilibrage	Justification
	IsolationForest (CPDP)		

2.3. Architectures d'apprentissage profond

2.3.1. Justification théorique

Notre exploration de l'apprentissage profond répond aux limitations des approches traditionnelles (régression logistique, arbres, forêts) qui requièrent une ingénierie manuelle des caractéristiques et capturent difficilement les interactions non-linéaires complexes. Les réseaux multicouches automatisent l'extraction de caractéristiques et apprennent des représentations hiérarchiques potentiellement plus expressives, abordant directement notre question RQ2 sur la comparaison architecturale.

2.3.2. CNN1D pour métriques

Les réseaux convolutifs unidimensionnels (CNN1D) transposent aux données séquentielles les principes des architectures convolutives développées pour le traitement d'images. Leur application aux vecteurs de métriques repose sur trois propriétés : l'extraction automatique de motifs locaux via filtres convolutifs, l'invariance positionnelle permettant de détecter des patterns indépendamment de leur position, et l'efficacité computationnelle découlant de la connectivité éparse et du partage de poids. Li et al. [35] et Pan et al. [36] appliquent cette architecture à la prédiction de défauts logiciels en traitant automatiquement des métriques logicielles et des représentations du code source, démontrant leur capacité à surpasser les méthodes traditionnelles sur plusieurs projets du corpus PROMISE. Deux variantes CNN1D-Simple et CNN1D-Dense sont évaluées afin d'isoler l'impact de la profondeur du réseau sur la généralisation ; leurs spécifications architecturales sont détaillées au Chapitre 4 (section 4.3.1). Toutefois, les filtres à fenêtre fixe ne modélisent pas les dépendances à longue portée entre caractéristiques distantes ;

ce que les mécanismes d'attention des Transformers permettent en calculant directement toutes les relations par paires.

2.3.3 Transformers

L'architecture Transformer de A. Vaswani et al. [37] remplace les opérations récurrentes et convolutives par un mécanisme d'attention auto-référentielle. Contrairement aux CNN dont les interactions sont limitées par la taille du filtre, l'attention calcule pour chaque paire (élément_i, élément_j) un score de pertinence, permettant de capturer des dépendances complexes à longue portée sans supposer de structure spatiale a priori. Notre investigation évalue trois configurations selon un gradient de pré-entraînement croissant (Tableau 3). Dans la Config 1 (Encoder Only) on utilise uniquement les métriques sans pré-entraînement afin d'isoler l'efficacité intrinsèque de l'attention sur données tabulaires. Dans Config 2 (Hybride) on fusionne les métriques et le code source via un décodeur CodeBERT pré-entraîné, qui exploite les représentations linguistiques apprises sur de vastes corpus de code. Enfin dans Config 3 (Full pretrained), on applique le pré-entraînement aux deux modalités (RoBERTa pour métriques, CodeBERT pour code) pour maximiser le transfert de connaissances. Cette progression nous permet d'isoler l'impact du pré-entraînement pour chaque modalité pour distinguer les gains attribuables à l'architecture de l'attention de ceux provenant du transfert linguistique. L'hypothèse directrice est que les représentations universelles pré-apprises amélioreront la transférabilité en contexte CPDP.

Tableau 3 : Configurations selon un gradient de pré-entraînement croissant

Configuration	Modalités	Pré-entraînement	Hypothèse théorique
1: Encoder only	Métriques	Aucun	Efficacité pure de l'attention sur métriques
2: Hybride	Métriques + Code	Décodeur CodeBERT	Fusion intelligente : métriques interrogent code
3: Full pretrained	Métriques + Code	RoBERTa + CodeBERT	Robustesse CPDP via représentations universelles pré-apprises

2.4. Configurations expérimentales

2.4.1 Validation temporelle

La crédibilité de nos résultats dépend du réalisme de notre protocole d'évaluation. C. Tantithamthavorn et al. [38] documentent dans leur analyse comparative de 101 ensembles de données publics que les techniques de validation standard peuvent produire des estimations biaisées si elles ne respectent pas la chronologie naturelle des données logicielles. Notre validation temporelle respecte strictement l'ordre chronologique : entraînement sur versions n-1, prédiction sur version n. Cette approche évite que des informations futures influencent l'entraînement.

2.4.2 Scénarios d'application

Nous distinguons deux scénarios reflétant des besoins industriels différents, structurant notre question RQ3 sur la transférabilité.

Le scénario Within-Project Defect Prediction (WPDP) entraîne le modèle sur les versions historiques d'un projet et le valide sur sa version courante. Il bénéficie de la cohérence des pratiques de développement et des patterns idiosyncrasiques propres au projet. Le scénario Cross-Project Defect Prediction (CPDP) entraîne le modèle sur l'ensemble des projets disponibles à l'exception du projet cible, puis valide sur la dernière version de ce dernier. Il répond au cas d'une organisation démarrant un nouveau projet sans historique de défauts. T. Zimmermann et al. [13] documentent que la volatilité des caractéristiques inter-projets compromet significativement la généralisation dans ce contexte.

Ces deux scénarios permettent d'analyser le compromis entre spécialisation au projet et généralisation inter-projets. Leur analyse comparative permettra de déterminer si les architectures pré-entraînées améliorent la transférabilité en contexte CPDP, et si cet avantage se maintient en contexte WPDP où la cohérence distributionnelle est naturellement assurée.

2.4.3 Métriques d'évaluation

Comme établi en section 2.1.3, l'exactitude (accuracy) est insuffisante en contexte déséquilibré. Nous retenons trois métriques complémentaires, chacune apportant un éclairage distinct sur le comportement des modèles.

Tableau 4 : Métriques d'évaluation pour classes déséquilibrées

Métrique	Formule	Propriété clé
F1-macro	(F1_défauts + F1_non-défauts) / 2	Poids égal aux deux classes
G-Mean	$\sqrt{(Sensibilité \times Spécificité)}$	Pénalise la négligence de la classe minoritaire
Accuracy	(TP + TN) / (TP + TN + FP + FN)	Comparabilité littérature (avec prudence)

Le G-Mean est retenu comme métrique principale, car il tend vers zéro dès lors qu'une classe est mal prédite, même si l'autre présente des performances élevées. Cette propriété en fait une mesure particulièrement adaptée aux données déséquilibrées. Le F1-macro complète cette analyse en combinant précision et rappel pour chaque classe, puis en les moyennant de manière non pondérée. L'accuracy est conservée uniquement à des fins de comparaison avec les travaux antérieurs, tout en étant interprétée avec précaution. D. Chicco et G. Jurman [39] ont démontré empiriquement qu'elle peut conduire à une surestimation de plus de 20 % des performances sur données déséquilibrées, justifiant notre approche de triangulation métrique.

2.5 Fondements épistémologiques et méthodologiques

Notre recherche s'inscrit dans un cadre méthodologique dont nous précisons ici la portée et les frontières.

D'abord, notre validation temporelle s'appuie sur les lois de l'évolution logicielle de M. Lehman [40], qui postulent une transformation continue des systèmes. Ces principes

(croissance continue, complexité croissante, auto-régulation) ont été validés empiriquement dans les projets open source modernes par I. Herraiz et al. [41]. S'entraîner sur les versions passées pour prédire celles futures s'aligne donc sur cette réalité, et rejette l'hypothèse de stationnarité invalide pour ces systèmes.

Ensuite, le bruit d'étiquetage systémique documenté par K. Herzig et al. [26] établit par ailleurs un plafond de performance théorique pour tout modèle de prédiction, même optimal (voir section 2.1.1). Cette incertitude intrinsèque des labels explique partiellement la convergence des performances observées entre différentes configurations méthodologiques.

Enfin, le corpus PROMISE constitue la référence établie pour la reproductibilité en prédiction de défauts, avec des annotations validées et des métriques normalisées facilitant les comparaisons inter-études. Sa description détaillée et les limites de généralisation à des contextes modernes sont discutées au Chapitre 4 (section 4.1) et au Chapitre 6 respectivement.

Le Chapitre 3 positionne notre travail par rapport à l’état de l’art.

CHAPITRE 3 : ÉTAT DE L'ART

La prédiction de défauts logiciels se trouve dans une situation paradoxale : les avancées architecturales s'accompagnent de problèmes de reproductibilité persistants qui fragilisent la crédibilité scientifique du domaine. Z. Mahmood et al. [42] révèlent que seulement 6% des 208 études ont été répliquées, et que 38% des réplications (11 sur 29) produisent des résultats divergents par rapport aux études originales. Cela les conduit à conclure qu'« il demeure incertain à quel point la prédiction de défauts est fiable ». Malgré cet avertissement, G. Giray et al. [18] documentent dans 102 études contemporaines sur l'apprentissage profond la persistance des mêmes problèmes. Il y a l'absence de packages de réplication, l'utilisation d'extraction manuelle de métriques (deux tiers des études), et le traitement hétérogène du déséquilibre de classes. Cette tension soulève une question centrale : les gains empiriques observés reflètent-ils une maturation scientifique réelle, ou sont-ils en partie artefacts de protocoles peu rigoureux ?

3.1 Évolution récente du domaine

- **L'apprentissage profond : promesses et réalité**

Les auteurs J. Li et al. [35] ont proposé un cadre de prédiction de défauts basé sur CNN1D qui exploite les arbres syntaxiques abstraits (AST) pour extraire automatiquement des caractéristiques sémantiques. Leur approche DP-CNN a été évaluée sur plusieurs projets

du corpus PROMISE, et montre que les réseaux de neurones convolutifs peuvent capturer des patterns complexes dans le code source. Toutefois, trois constats en nuancent la portée.

Premièrement, Jiarpakdee et al. [43] ont analysé l'impact des métriques corrélées sur l'interprétation des modèles de défauts et ont conclu que la variance inter-projets des modèles d'apprentissage profond suggère que ces architectures capturent des patterns projet-spécifiques plutôt que des régularités généralisables. Ce résultat est cohérent avec l'hypothèse de S. McIntosh et Y. Kamei [12] sur la dominance des facteurs contextuels. Deuxièmement, S. Wang et al. [44] ont montré dans leur étude sur l'apprentissage automatique de caractéristiques sémantiques que les gains de performance en configuration intra-projet ne se généralisent pas nécessairement aux contextes inter-projets, révélant que la sophistication architecturale ne garantit pas la transférabilité. Troisièmement, comme l'ont souligné A. Farid et al. [45], le coût computationnel des architectures profondes augmente substantiellement sans garantie proportionnelle d'amélioration des performances. En effet, l'architecture combinée nécessite environ 500K paramètres et des temps d'entraînement substantiellement accrus par rapport aux modèles de références classiques, pour des gains en score F1 qui restent modestes sur certains projets PROMISE. Effectivement, bien qu'atteignant une amélioration moyenne de 25% en score F1, la variabilité inter-projets reste substantielle (de gains négligeables sur certains projets à des améliorations marquées sur d'autres).

Ces observations soulèvent une question centrale : dans quels contextes l'apprentissage profond justifie-t-il sa complexité accrue ?

- **Introspection méthodologique émergente**

Parallèlement aux avancées techniques, une prise de conscience méthodologique émerge dans la communauté. M. Shepperd et al. [46] ont conduit une méta-analyse portant sur 42 études et environ 600 ensembles de résultats empiriques. À l'aide d'un modèle ANOVA à effets aléatoires, ils quantifient la contribution de quatre facteurs à la variabilité des performances prédictives : le groupe de recherche, le jeu de données, les métriques d'entrée et le classifieur. Les résultats montrent que le groupe de recherche, c'est-à-dire l'expertise des chercheurs, leurs pratiques de prétraitement et leur protocole expérimental,

est le facteur le plus déterminant, expliquant 31 % de cette variabilité. Le choix du classifieur, en revanche, n'en explique que 1,3 %. Cette observation met en évidence un biais méthodologique systémique qui dépasse largement les différences algorithmiques.

C. Tantithamthavorn et al. [15] ont quantifié l'impact souvent négligé des techniques d'équilibrage de classes qui influencent différentiellement les performances selon la mesure utilisée (précision, rappel, F-mesure, AUC). Tandis que l'algorithme de classification employé remet en question l'existence d'une approche universellement optimale. Cette observation motive directement notre évaluation comparative des stratégies de standardisation (voir Chapitre 4, section 4.2.1).

N. Bhat et S. Farooq [47] ont mené une évaluation comparative exhaustive couvrant les contextes WPDP et CPDP. Leurs résultats confirment empiriquement que les approches combinant de multiples dimensions de métriques (processus, produit, hybrides) surpassent significativement les approches mono-famille, indépendamment des considérations d'effort. Cette étude révèle aussi que la variabilité méthodologique (choix de métriques, techniques d'équilibrage, protocoles de validation) explique une part substantielle de la variance observée entre études.

La communauté a développé des standards de reproductibilité (REFORMS, DOME-ML), mais comme le souligne H. Semmelrock et al. [23], l'adoption reste limitée en raison de contraintes systémiques (pressions de publication, absence d'incitations institutionnelles) plutôt que d'un simple manque de rigueur individuelle.

Nous examinons maintenant les trois familles architecturales dominantes de la littérature, en évaluant les conditions de leur efficacité.

3.2 Trois familles architecturales

- **Convolutional Neural Network (CNN)**

L'application des réseaux de neurones convolutifs (CNN1D) à la prédiction de défauts logiciels s'est développée selon deux axes complémentaires.

Le premier axe exploite les arbres de syntaxe abstraite (AST) pour extraire des caractéristiques sémantiques du code source. L'approche pionnière DP-CNN de J. Li et al. [35] combine une couche convolutive avec des caractéristiques traditionnelles. Elle améliore la méthode DBN de référence de 12 % en F-mesure sur sept projets PROMISE. C. Pan et al. [36] raffinent cette architecture en approfondissant le réseau d'une à trois couches convolutives séquentielles avec régularisation. Ce raffinement apporte un gain de 2,2 % en F-mesure par rapport au CNN original et de 6 % par rapport aux meilleurs modèles d'apprentissage automatique classiques. Ces travaux révèlent néanmoins une hétérogénéité substantielle des performances entre projets, suggérant une forte dépendance aux caractéristiques projet-spécifiques.

Le second axe applique directement les CNN aux métriques logicielles tabulaires. Les auteurs K. Wongpheng et P. Visutsak [48] confirment la viabilité de cette approche (70,2% de précision sur cinq projets NASA), mais observent des gains limités face aux forêts aléatoires. Cette observation suggère que la sophistication architecturale seule ne garantit pas d'amélioration décisive sur des corpus de taille modérée.

Face au défi de la généralisation inter-projets, où les distributions de métriques et patterns de défauts varient substantiellement, des mécanismes d'adaptation de domaine deviennent nécessaires. S. Qiu et al. [49] ont intégré le Maximum Mean Discrepancy dans une architecture Transfer CNN pour réduire les distances distributionnelles entre projets sources et cible. Leur méthode atteint une F-mesure moyenne de 0,532 en CPDP. J. Deng et al. [50] affinent cette approche en explorant systématiquement différentes granularités de nœuds AST combinées à un CNN multi-noyaux, démontrant que la granularité de représentation impacte significativement la capacité de transfert inter-projets.

La synthèse de cette littérature met en lumière une tension fondamentale : bien que prometteurs et adaptables, les CNN présentent des performances inégales et sont susceptibles de suroptimisation sur un sous-ensemble non représentatif de projets. Ces constats motivent notre évaluation comparative sur les 11 projets PROMISE selon un gradient de complexité architecturale (Chapitre 4, section 4.2.2).

- **Transformers**

L'adaptation des architectures Transformer à la prédiction de défauts s'articule autour de trois configurations principales, chacune exploitant différemment les mécanismes d'attention et le transfert d'apprentissage.

Les architectures Transformer exploitant l'auto-attention appliquent ces mécanismes directement aux séquences de code ou aux métriques logicielles. Q. Zhang et B. Wu [51] démontrent avec DP-Transformer que les mécanismes d'attention multi têtes capturent efficacement les dépendances structurelles dans les représentations AST. Le framework DP-Transformer se compose d'encodeurs empilés avec attention positionnelle et couches entièrement connectées, permettant d'extraire automatiquement les caractéristiques sémantiques sans ingénierie manuelle.

Les mécanismes d'attention semblent toutefois privilégier les métriques de taille simple (LOC, SLOC) plutôt que les métriques structurelles (LCOM, CBO). Cela suggère que l'attention redécouvre des corrélations connues, ou que la taille est le facteur de défaut dominant, questionnant ainsi la valeur ajoutée des suites métriques CK.

Les modèles pré-entraînés exploitent le transfert d'apprentissage depuis de vastes corpus de code. C. Pan et al. [7] démontrent avec CodeBERT que le pré-entraînement sur des millions de lignes de code open-source permet d'extraire des représentations sémantiques pertinentes sans ingénierie manuelle de caractéristiques. W. Ahmad et al. [52] généralisent cette approche avec PLBART, un modèle unifié pour la compréhension et génération de programmes, pré-entraîné par débruitage autoencoder sur des fonctions Java et Python. Ces capacités se sont révélées transférables à des tâches discriminatives incluant la réparation de code, la détection de clones et la détection de code vulnérable.

Ces résultats prometteurs nécessitent toutefois une interprétation prudente. Car comme le démontrent L. Song et L. Minku [53], les validations croisées aléatoires utilisées majoritairement dans ces études peuvent surestimer substantiellement les performances réelles en ne respectant pas l'ordre temporel des commits.

Les architectures hybrides multimodales combinent caractéristiques sémantiques du code et métriques traditionnelles. M. Uddin et al. [54] proposent SDP-BB, un modèle exploite

BERT pour générer des représentations vectorielles sémantiques du code source, puis utilise BiLSTM pour capturer l'information contextuelle bidirectionnelle dans les séquences de tokens. Évalué sur dix projets PROMISE, SDP-BB surpasse les modèles de références en score F1 moyenne, avec des couvertures optimales variant selon le contexte (90% pour WPDP, 70-80% pour CPDP), révélant que différents contextes prédictifs nécessitent des fenêtres d'analyse adaptées.

Tableau 5 : Performances rapportées dans la littérature récente

Architecture	G-Mean WPDP	Écart CPDP	Protocole validation	Source
RF (baseline)	0.66 ± 0.07	-21%	Temporelle	[27]
CNN1D (DP-CNN)	~0.70	-20%	Mixte	[36]
CNN multicanaux	~0.73	Variable	Temporelle partielle	[35]
Transfer CNN	—	~0.53	CPDP uniquement	[49]
DP-Transformer	~0.74	-19%	Aléatoire*	[51]
SDP-BB (BERT+BiLSTM)	~0.76	Variable	Aléatoire*	[54]
PLBART	~0.75-0.79	Variable	Aléatoire*	[52]

**Validation croisée aléatoire : potentiellement surestimée selon Song et Minku [53]*

Ce tableau révèle un pattern intéressant : les gains incrémentaux entre architectures sophistiquées (CNN → Transformers pré-entraînés) demeurent modestes (3-8%) malgré une complexité paramétrique substantiellement accrue. Notre recherche explore ce gradient de pré-entraînement à travers trois configurations. Cette progression permet d'évaluer empiriquement si la sophistication architecturale croissante se traduit par des gains substantiels sur un corpus de taille modérée comme PROMISE, ou si nous approchons une limite intrinsèque liée aux données disponibles.

- **Les approches multimodales**

Les architectures multimodales exploitent la complémentarité entre différentes représentations du code pour améliorer la détection de défauts, s'inscrivant dans une logique de fusion informationnelle.

A. Farid et al. [45] proposent CBIL, leur architecture extrait d'abord les caractéristiques spatiales via CNN appliqué sur les séquences AST, puis utilise Bi-LSTM bidirectionnel pour capturer les dépendances séquentielles à longue portée entre tokens. Le modèle CBIL préserve les caractéristiques clés tout en ignorant le bruit, améliorant ainsi la précision de prédiction. Évaluée sur sept projets PROMISE, cette approche hybride améliore significativement les modèles de références mono-architecture, supportant l'hypothèse que l'exploitation de la complémentarité extraction spatiale/temporelle capture plus efficacement l'information structurelle du code.

S. Wang et al. [44] démontrent aussi que l'apprentissage automatique de caractéristiques sémantiques via Deep Belief Networks évite l'ingénierie manuelle de caractéristiques, ouvrant la voie aux approches end-to-end. Leur approche utilise Deep Belief Networks (DBN) pour extraire automatiquement des caractéristiques pertinentes depuis les AST. Leur framework apprend des représentations hiérarchiques du code en empilant plusieurs couches de Restricted Boltzmann Machines (RBM), capturant progressivement des abstractions de plus en plus complexes.

H. Liang et al. [55] affinent cette approche avec Seml, exploitant Word2Vec et LSTM pour apprendre automatiquement les représentations sémantiques depuis les séquences de tokens AST, avec une efficacité démontrée en contextes WPDP et CPDP.

J. Deng et al. [50] ont poussé cette logique en explorant systématiquement trois granularités de nœuds AST pour identifier les niveaux d'abstraction optimaux. Ces travaux révèlent une observation importante : la granularité de représentation impacte significativement la performance, certaines abstractions préservant mieux l'information sémantique critique pour la détection de défauts.

Les progrès semblent ainsi davantage conditionnés par la richesse des données disponibles que par la sophistication architecturale, ce qui orienterait les efforts futurs vers

l'enrichissement des représentations (métriques de processus, contexte domaine) plutôt que vers une complexification des modèles.

3.3. Défis méthodologiques persistants

Quatre catégories de défis freinent la maturation du domaine : problèmes de préparation des données, protocoles d'évaluation, fragmentation comparative, et reproductibilité. Cette classification distingue la négligence corrigible (erreurs évitables) des tensions inhérentes nécessitant des compromis éclairés.

- **Préparation des données**

Au-delà des critiques répétées, les erreurs de préparation des données persistent dans la littérature récente. Z. Mahmood et al. [42] notent que 94% des 208 études de SDP analysées n'ont jamais été répliquées, et que la documentation méthodologique insuffisante rend impossible la détection des erreurs. Ces erreurs sont, entre autres, le calcul des paramètres de normalisation sur l'ensemble complet (entraînement + test) plutôt que sur l'ensemble d'entraînement seulement. Cette absence généralisée de transparence méthodologique perpétue potentiellement des violations du principe fondamental de séparation train/test.

Au-delà de ces erreurs manifestes, A. Vescan et al. [16] documentent un dilemme épistémologique fondamental. La standardisation globale (calculée sur l'ensemble d'entraînement du projet) préserve la comparabilité inter-projets mais ignore l'évolution temporelle des métriques, facteur potentiellement crucial dans les projets à longue durée de vie. La standardisation locale (calculée indépendamment pour chaque version) capture mieux la dynamique temporelle mais compromet la transférabilité inter-projets en créant des échelles de mesure incompatibles. Aucune stratégie ne domine universellement ; le choix optimal dépend du contexte d'application (WPDP vs CPDP, stabilité du projet). Cette réalité motive notre évaluation comparative de trois stratégies : SG-globale, SV-par version, et SR-hybride robuste.

L'interaction entre techniques d'échantillonnage (SMOTE, sous-échantillonnage) et architectures d'apprentissage profond reste presque complètement inexplorée. S. Feng et

al. [10] documentent l'instabilité de SMOTE en SDP sur données tabulaires, mais leur analyse se limite aux méthodes traditionnelles (RF, SVM). Il est possible que les architectures profondes, avec leurs capacités de représentation accrues, réagissent différemment aux données synthétiques générées par SMOTE ; une hypothèse que notre recherche teste empiriquement.

- **Protocoles et métriques**

Malgré les critiques répétées, certaines études récentes continuent d'utiliser des validations croisées aléatoires violant l'ordre chronologique des données. Cette persistance reflète probablement des pressions de publication : les résultats avec validation temporelle sont moins spectaculaires, compromettant la compétitivité dans les revues prestigieuses. Cette dynamique crée un biais systémique où les performances rapportées surestiment ce qui serait obtenu en déploiement réel.

Le choix des métriques d'évaluation constitue un défi majeur dans notre contexte de déséquilibre de classes (typiquement 10 à 20 % de classes fautives). L'exactitude (accuracy) est d'emblée écartée car elle favorise la classe majoritaire et ne reflète pas la capacité réelle à détecter les défauts. Le G-Mean, moyenne géométrique de la sensibilité et de la spécificité, constitue une alternative plus robuste en pénalisant les modèles qui négligent la classe minoritaire. J. Yao et M. Shepperd [56] démontrent empiriquement que le coefficient de corrélation de Matthews (Matthews Correlation Coefficient, MCC) est encore plus fiable. En effet, sur 4 017 comparaisons pairées issues de huit études, le classement des classifieurs diffère entre F1 et MCC dans 23 % des cas, révélant le biais de F1 face aux données déséquilibrées. Nous retenons néanmoins le G-Mean comme métrique principale afin de maintenir la comparabilité directe avec les travaux de référence du domaine, tout en rapportant le MCC à titre complémentaire.

- **Comparabilité et reproductibilité**

L'optimisation des hyperparamètres pose un défi méthodologique substantiel en apprentissage profond. C. Pan et al. [36] documentent le phénomène de l’ « instabilité des hyperparamètres ». Différentes configurations d'hyperparamètres génèrent des

performances moyennes similaires sur l'ensemble des projets, mais divergent significativement pour des projets individuels. M. Rakha et al. [57] montrent empiriquement que l'impact de l'optimisation des hyperparamètres varie substantiellement selon le scénario de validation considéré. En contexte IVDP (Inner Version Defect Prediction, validation intra-version), les gains de performance sont systématiquement plus élevés qu'en contexte CVDP (Cross Version Defect Prediction, validation inter-versions). Pour la métrique AUC, cette différence est statistiquement significative pour 17 des 28 algorithmes testés soit 61 %. Selon l'algorithme considéré, l'écart de performance entre les deux scénarios peut être modeste ou, au contraire, très marqué.

Cette sensibilité soulève une question méthodologique fondamentale : comment garantir des comparaisons objective entre architectures aux besoins différents ? Faut-il optimiser intensivement tous les modèles (équité substantielle) au risque de biais computationnels, ou utiliser des hyperparamètres standardisés (équité procédurale) ignorant les spécificités architecturales ? Cette tension reste largement non résolue dans la littérature.

Notre propre recherche n'échappera pas entièrement à ces tensions méthodologiques. Nous reconnaîtrons clairement au Chapitre 6 des limitations similaires : asymétrie d'évaluation imposée par des contraintes computationnelles (CNN évalués sous SG-SR, Transformers sous SG seulement). Cette transparence documente explicitement nos propres compromis, qui est précisément l'approche méthodologique que nous plaidons.

L'analyse de ces défis méthodologiques persistants révèle 4 lacunes structurantes dans la littérature actuelle. Ces lacunes motivent directement le positionnement et le design expérimental de notre recherche.

3.4. Lacunes structurantes et positionnement de notre recherche

3.4.1 Lacunes structurantes

Notre analyse révèle 4 lacunes fondamentales qui persistent dans la littérature contemporaine. Plutôt que simplement les identifier, nous cherchons à comprendre pourquoi elles persistent et comment notre recherche peut contribuer à les adresser systématiquement.

L1 : Fragmentation des stratégies de prétraitement. Bien que le prétraitement soit reconnu comme critique en SDP, aucune étude ne compare systématiquement les stratégies de standardisation (globale, par version, robuste) spécifiquement pour l'apprentissage profond. A. Vescan et al. [16] comparent différentes techniques de prétraitement en CPDP à travers des algorithmes composites, mais n'examinent pas les architectures profondes avec leurs capacités d'extraction automatique des caractéristiques.

L2 : Interactions méconnues entre rééquilibrage et architectures profondes. Les techniques de rééquilibrage sont largement étudiées, mais leurs interactions avec les architectures profondes restent inexplorées. S. Feng et al. [10] démontrent que la stabilité de SMOTE varie selon les classificateurs, tandis que C. Tantithamthavorn et al. [15] établissent ainsi que l'impact du rééquilibrage dépend de la mesure de performance. Néanmoins, ces travaux n'examinent pas comment SMOTE et sous-échantillonnage interagissent avec CNN et Transformers, dont les mécanismes d'apprentissage diffèrent radicalement.

L3 : Évaluations fragmentées des architectures profondes. La revue systématique de G. Giray et al. [18] révèle que les CNN représentent l'architecture la plus utilisée en SDP profond. L'émergence récente des Transformers (Q. Zhang et B. Wu [51] ; Dhawan et Bansal [62]) n'a pas fait l'objet de comparaisons systématiques avec les CNN selon des protocoles unifiés. Les études disponibles utilisent des jeux de données, stratégies de prétraitement et métriques hétérogènes, rendant difficile toute conclusion sur la supériorité relative de ces architectures.

L4 : Persistance de protocoles optimistes. M. Shepperd et al. [46] ont révélé qu'une part substantielle de la variabilité des résultats est attribuable à l'équipe de recherche elle-même : deux groupes appliquant nominalement la même méthode peuvent rapporter des performances sensiblement différentes. S. Herbold et al. [58] proposent un benchmark standardisé pour améliorer la comparabilité en CPDP, mais peu d'études adoptent de tels cadres et documentent explicitement les sources de biais et leur impact sur la généralisabilité.

3.4.2 Positionnement scientifique

Nous nous positionnons à l'intersection de ces lacunes par une approche factorielle prétraitement-architecture reconnaissant explicitement leurs interdépendances. Notre contribution se distingue sur trois plans.

Premièrement, nous explorons les interactions prétraitement-architecture. Théoriquement, une standardisation optimale pour le CNN pourrait affecter différemment les Transformers dont les mécanismes d'attention présentent des sensibilités différentes. Notre design teste cette hypothèse pour les CNN (SG, SV, SR) mais se limite à SG pour les Transformers en raison de contraintes techniques. Cette asymétrie, loin de constituer un échec, documente les contraintes pratiques du domaine.

Deuxièmement, notre gradient de pré-entraînement (entraîné de zéro, semi-pré-entraîné, entièrement pré-entraîné) teste l'hypothèse que le transfert d'apprentissage varie selon les modalités. Les métriques structurées (espaces 20-30D, distributions continues) et le code source (séquences discrètes, sémantique compositionnelle) présentant des caractéristiques fondamentalement différentes, les bénéfices du pré-entraînement pourraient être asymétriques entre modalités.

Troisièmement, nous adoptons une transparence radicale documentant explicitement nos compromis et limitations : validation temporelle exclusive, asymétrie CNN/Transformer, évaluation multicritère privilégiant le G-Mean tout en rapportant d'autres métriques pour la comparabilité. Cette posture reconnaît qu'aucune méthodologie n'est parfaite et qu'admettre ses limitations renforce la crédibilité scientifique.

Notre objectif n'est pas d'identifier une configuration universellement optimale, mais de cartographier empiriquement les compromis entre choix méthodologiques, afin de fournir aux praticiens une base de décision adaptée à leur contexte.

3.5. Synthèse

Les quatre lacunes identifiées (prétraitement, rééquilibrage, comparaison architecturale, protocoles) convergent vers un même constat : le domaine manque d'études factorielles

documentant explicitement les interactions entre ces dimensions. C'est précisément le positionnement de notre recherche.

L'analyse de la dernière décennie montre que les architectures profondes ont apporté des gains dans certains contextes spécifiques, notamment en WPDP avec pré-entraînement sur de vastes corpus de code. Ces gains s'accompagnent toutefois de contreparties comme la complexité computationnelle accrue, la fragilité face aux variations inter-projets, et la sensibilité aux hyperparamètres. L'apprentissage profond n'est donc pas une solution universelle mais un outil dont l'utilité dépend du contexte d'application.

La littérature révèle une dégradation de performance généralement observée entre contextes WPDP et CPDP, suggérant des limites à la généralisation inter-projets. Cette observation soulève d'importantes questions : les difficultés de généralisation reflètent-elles des limites intrinsèques des métriques CK, ou résultent-elles de choix méthodologiques sous-optimaux ? Notre approche factorielle permettra d'explorer systématiquement ces questions.

Le Chapitre 4 concrétise notre démarche en détaillant précisément notre protocole expérimental à savoir les jeux de données, les architectures, les stratégies de prétraitement évaluées, les configurations de validation, et les analyses statistiques effectuées.

CHAPITRE 4 : MÉTHODOLOGIE

Ce chapitre présente le cadre méthodologique développé pour répondre aux questions de recherche formulées au Chapitre 1. Les six sections suivantes couvrent successivement le

corpus expérimental (4.1), les stratégies de prétraitement (4.2), les architectures d'apprentissage profond (4.3), le protocole expérimental intégré (4.4), les métriques d'évaluation (4.5) et les limitations méthodologiques (4.6). La Figure 1 offre une vue synthétique de l'ensemble

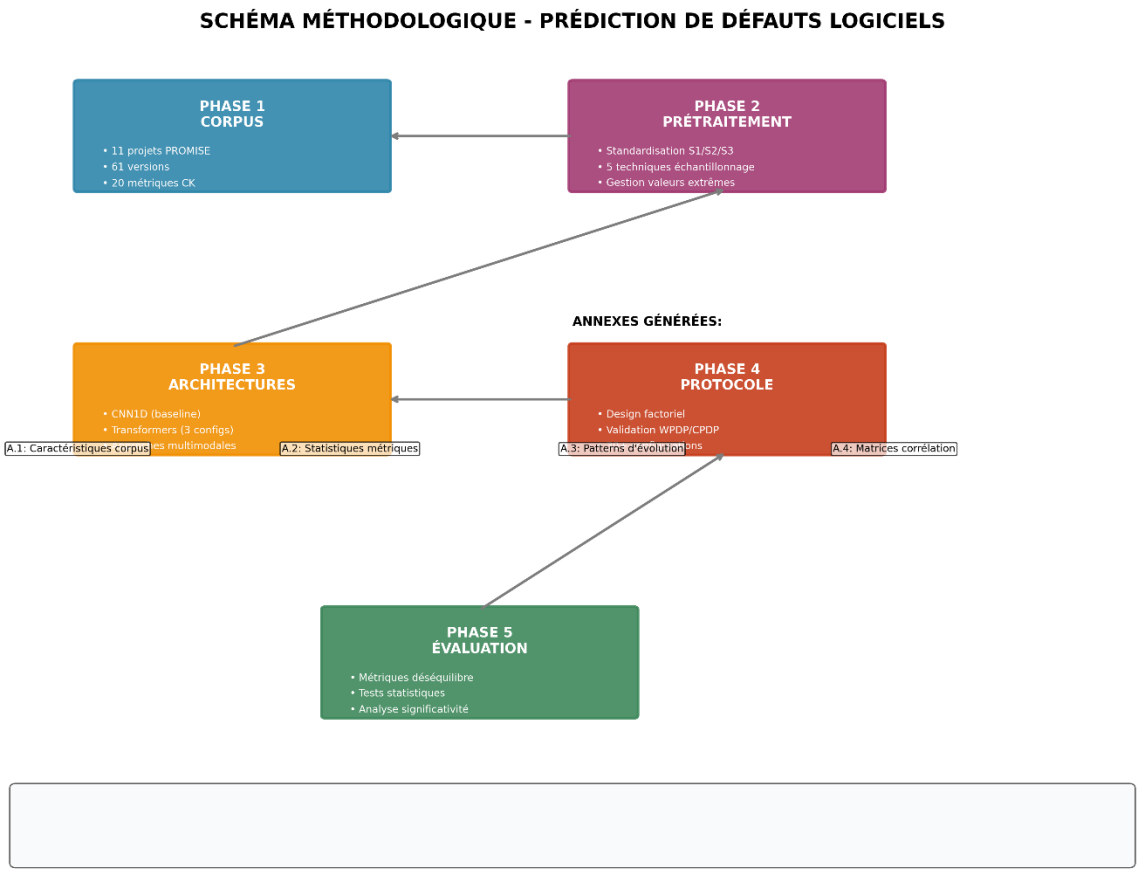


Figure 1 : Schéma méthodologique – Prédictions de défauts logiciels

4.1 Corpus expérimental PROMISE

4.1.1 Justification de la sélection

Nous avons retenu le corpus PROMISE (PRedictOr Models In Software Engineering) pour trois raisons principales, en lien avec nos objectifs de recherche.

D'abord, sa reconnaissance comme standard de facto documentée par S. Hosseini et al. [11], facilitant la comparaison avec les travaux du Chapitre 3 et assurant la reproductibilité de nos résultats. Ensuite, l'hétérogénéité contrôlée de ses 11 projets Java open source,

caractérisés par M. Jureczko et L.Madeyski [59] avec 20 métriques CK par classe. Ce corpus couvre des domaines variés allant de l'outillage système (Ant, Ivy) aux bibliothèques spécialisées (Lucene, Log4j), permettant de tester la robustesse des interactions prétraitement-architecture visées. Enfin, la disponibilité conjointe de métriques quantitatives CK et de code source constitue un atout décisif pour notre évaluation des architectures Transformer multimodales.

Cette approche répond aux préoccupations de Z. Mahmood et al. [42], dont l'analyse révèle que seulement 6% des études de prédiction de défauts ont fait l'objet de réplications, soulignant l'importance critique de jeux de données publics standardisés pour la reproductibilité du domaine.

4.1.2 Caractérisation empirique

Les 11 projets sélectionnés présentent une diversité architecturale avec des profils contrastés essentiels à la validation de nos hypothèses (Tableau 4.1). Les taux de défauts varient de 10,2 % (Xerces) à 28,7 % (Velocity), créant des conditions d'apprentissage qui testent naturellement l'adaptabilité des techniques d'échantillonnage que nous évaluons (RQ1, Chapitre 1). Ces projets couvrent la période 2007-2010 et représentent des systèmes Java matures avec historiques de défauts validés, établissant le corpus standard pour la recherche en prédiction de défauts logiciels.

Tableau 6 : Vue d'ensemble du corpus expérimental PROMISE

Projet	Nb. versions	Classes	Défauts %	LOC µ	WMC µ	Domaine
Ant	5	1 692	20,7	296	10,8	Build
Camel	4	2 784	20,2	111	8,4	Messaging
Ivy	3	704	16,9	248	11,0	Dépendances
JEdit	5	1 749	17,3	457	12,8	Éditeur
Log4j	3	449	57,9	177	7,7	Logging
Lucene	3	782	56,0	277	9,8	Indexation

Projet	Nb. versions	Classes	Défauts %	LOC μ	WMC μ	Domaine
POI	4	1 378	51,3	289	13,9	MS Office
Synapse	3	635	25,5	196	7,6	Web services
Velocity	3	639	57,5	253	8,9	Templates
Xalan	4	3 320	54,4	413	11,1	XSLT
Xerces	4	1 643	39,8	340	10,8	Parseur XML
TOTAL	**41**	**15 775**	**36,3**	**298**	**10,6**	—

Note. *μ = moyenne pondérée par le nombre de classes. Classes = nombre total de classes toutes versions confondues. LOC μ = Lines of Code moyenne. WMC μ = Weighted Methods per Class moyenne (complexité). Les statistiques détaillées par version sont présentées en Annexe A.1.*

- Caractéristiques distributionnelles critiques : Les métriques présentent des violations systématiques de la normalité, avec des coefficients de variation (CV) dépassant souvent les 100%, et atteignant parfois 200% pour certains projets. Velocity illustre ce phénomène : pour la métrique LOC, un coefficient de variation de 116% (moyenne de 289,1, écart-type de 335,2) invalide les hypothèses gaussiennes de certaines standardisations classiques. Les valeurs maximales extrêmes que nous avons observées (RFC atteignant 390 pour POI, WMC dépassant 102 pour certaines classes de Xalan) ne constituent pas nécessairement des anomalies systématiques. Elles peuvent refléter des patterns architecturaux légitimes. Cette réalité a justifié notre intégration de techniques sophistiquées de détection des valeurs aberrantes qui distinguent les valeurs extrêmes légitimes du bruit réel (section 4.2.2).

L'hétérogénéité inter-projets que nous documentons ici confirme qu'aucune stratégie de standardisation unique ne sera optimale pour tous les contextes, motivant notre évaluation comparative systématique des trois stratégies SG (ensembliste), SV (par version) et SR-Robuste (séquentielle) sur les architectures CNN1D. Pour les architectures Transformer, les contraintes computationnelles et de temps nous ont conduits à nous limiter à l'évaluation avec SG, choix que nous documentons et justifions en section 4.3.3.

Tableau 7 : Caractéristiques distributionnelles de certaines métriques CK par projet

Projet	WMC		CBO		RFC		LOC	
	μ	**CV(%)**	**μ**	**CV(%)**	**μ**	**CV(%)**	**μ**	**CV(%)**
Ant	8,4	139	11,0	203	33,6	101	245,3	122
Camel	8,4	124	10,6	191	20,9	111	198,5	131
Ivy	7,3	129	3,9	99	18,3	89	187,2	114
JEdit	9,2	135	9,2	132	22,5	110	287,6	128
Log4j	7,7	104	7,3	125	23,6	98	189,3	121
Lucene	8,9	145	7,2	100	22,2	94	267,9	117
POI	13,9	102	9,2	190	29,9	115	312,5	117
Synapse	7,9	112	13,2	89	29,5	82	276,1	113
Velocity	8,9	160	10,7	114	23,1	117	289,1	116
Xalan	11,5	142	14,5	133	30,2	118	378,9	112

Note. μ = moyenne; CV = coefficient de variation (%). Métriques retenues : WMC et LOC (taille), CBO (couplage), RFC (complexité). Les statistiques complètes (incluant DIT, NOC, LCOM) ainsi que les détails par version sont disponibles en Annexe A.2.

- Patterns d'évolution temporelle : Notre analyse qualitative de l'évolution temporelle des métriques entre versions successives a révélé quatre patterns distincts qui ont guidé nos hypothèses concernant l'efficacité relative des stratégies de standardisation. Quatre patterns sont observés. Les projets stables (Ant, Log4j, POI, Ivy) présentent des métriques relativement constantes entre versions. Les projets en croissance (Lucene, Xalan, Synapse) montrent des tendances systématiquement croissantes. Les projets oscillatoires (Velocity, Camel) exhibent des variations périodiques. Enfin, les projets en décroissance (JEdit, Xerces) suggèrent des efforts de restructuration continu. Ces patterns motivent collectivement l'évaluation comparative des trois stratégies SG, SV et SR.. Les analyses détaillées incluant WMC, CBO et les évolutions normalisées sont présentées en Annexe A.3.

- Corrélations inter-métriques : Les métriques CK que nous utilisons présentent des corrélations importantes. Par exemple, WMC corrèle fortement avec RFC (coefficient de corrélation de Spearman $\rho > 0,7$ pour la plupart des projets), reflétant le fait que les classes avec de nombreuses méthodes (WMC élevé) tendent également à invoquer de nombreuses méthodes externes (RFC élevé). Ces corrélations justifient conceptuellement l'utilisation de mécanismes d'attention dans nos architectures Transformer (section 4.3.2), car ces mécanismes peuvent capturer automatiquement ces dépendances complexes entre métriques sans nécessiter une ingénierie de caractéristiques explicite. Les patterns de corrélation détaillés pour les six métriques CK (WMC, CBO, RFC, DIT, NOC, LCOM) sont documentés en Annexe A.4, incluant les matrices de corrélation par projet.

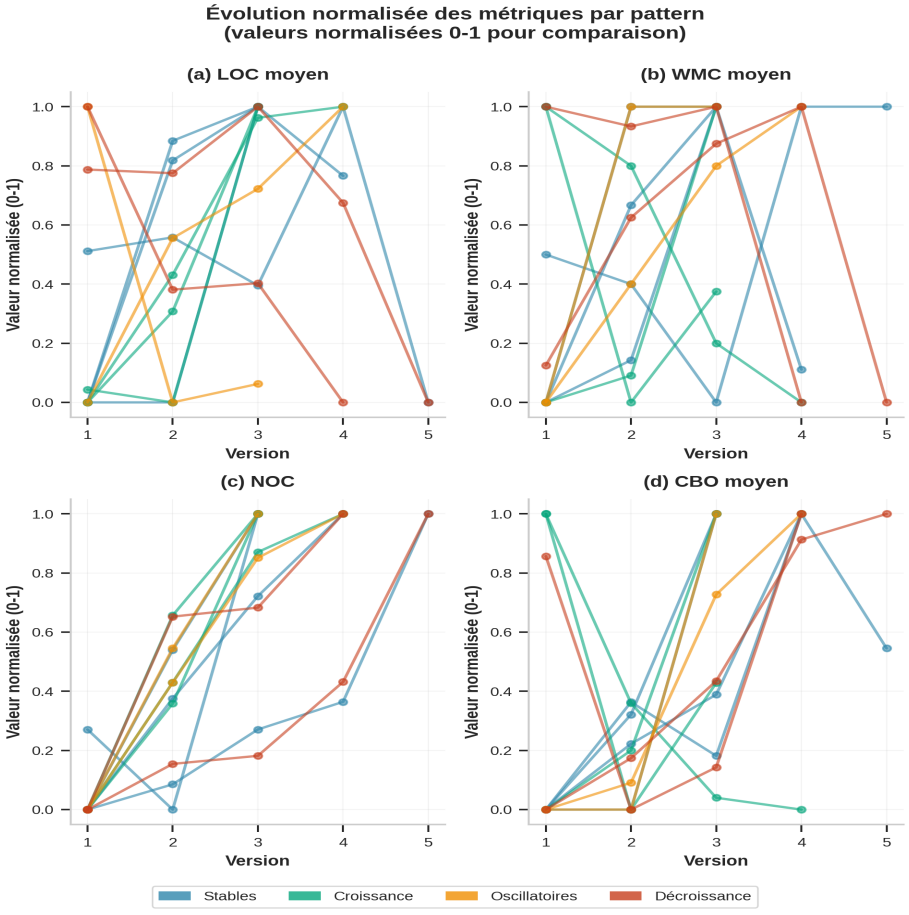


Figure 2 : Patterns d'évolution temporelle

4.1.3 Corpus multimodal

Pour nos architectures Transformer les plus avancées (configurations 2 et 3 que nous décrivons en section 4.3.2), nous avons étendu le corpus traditionnel de métriques en intégrant le code source complet des classes Java. Cette extension répond à l'objectif O2 du Chapitre 1 d'évaluer des architectures multimodales. Elle permet d'explorer la synergie potentielle entre représentations quantitatives (métriques CK) et représentations sémantiques obtenues par tokenisation via CodeBERT.

Notre pipeline maintient rigoureusement la correspondance exacte entre chaque ligne de métriques et la classe Java associée via un système de vérification croisée basé sur les noms qualifiés des classes et les timestamps de version. Cette correspondance est critique pour les architectures Transformer hybrides où l'attention croisée doit aligner correctement les représentations métriques et code. Les détails d'implémentation de cette correspondance, incluant la gestion des cas particuliers (classes internes, classes anonymes), sont présentés dans le dépôt GIT.

Cette approche s'inspire des travaux de A. Abdu et al. [60] qui montrent l'efficacité de la fusion des caractéristiques traditionnelles (métriques) et sémantiques (code) en SDP. Elle étend leur travail à une évaluation comparative systématique intégrant différentes stratégies de prétraitement et différents degrés de pré-entraînement.

Notre contribution distinctive réside dans cette évaluation factorielle prétraitement × architecture que nous avons identifiée comme lacune au Chapitre 3.

4.1.4 Synthèse et articulation

La caractérisation du corpus PROMISE révèle trois défis méthodologiques.

Premièrement, les métriques présentent une hétérogénéité distributionnelle extrême, avec des coefficients de variation dépassant régulièrement 100 % et atteignant parfois 200 %. Ces valeurs révèlent des violations systématiques des hypothèses de normalité sous-tendant la standardisation classique Z-score.

Ce constat nous a conduits à développer une taxonomie de standardisation à trois volets : SG (ensembliste assumant l'invariance), SV (par version capturant la volatilité

temporelle), et SR-Robuste (séquentielle combinant adaptation locale et cohérence globale via statistiques robustes). Cette taxonomie, constitue notre première contribution méthodologique (C1, Chapitre 1) en réponse à RQ1.

Deuxièmement, le déséquilibre variable entre classes que nous avons mesuré (ratios allant de 1:5 pour Velocity à 1:9 pour Xerces) exige des techniques d'échantillonnage adaptatives plutôt qu'une approche universelle. Cette variabilité, combinée aux caractéristiques distributionnelles problématiques (asymétrie, valeurs extrêmes), nous a conduit à concevoir un pipeline d'échantillonnage sophistiqué à quatre techniques : SMOTE adaptatif, sous-échantillonnage aléatoire, et des approches hybrides combinant détection des valeurs aberrantes et équilibrage. Cette diversité technique nous permet d'évaluer empiriquement quelles stratégies fonctionnent dans quels contextes.

Troisièmement, la richesse multimodale disponible (métriques CK + code source complet) ouvre des perspectives d'exploitation sémantique que les approches traditionnelles basées uniquement sur les métriques ne pouvaient pas explorer. Cette opportunité a motivé notre développement d'architectures Transformer multimodales exploitant CodeBERT pour le code source, avec différents degrés de pré-entraînement testant l'hypothèse que le transfert d'apprentissage améliore différentiellement les performances selon les modalités de données (RQ2).

4.2 Stratégies de prétraitement

4.2.1 Standardisation des métriques

La standardisation doit trouver un équilibre entre invariance (pour la généralisation) et adaptation (pour les spécificités locales). Nous étendons les travaux de J. Nam et al. [61] sur ce point en proposant une stratégie hybride : la Standardisation Robuste. Cette approche combine les avantages des méthodes globale et locale, et utilise des statistiques robustes pour résister aux violations de normalité que nous avons quantifiées.

- **Standardisation ensembliste ou globale (SG)**

Cette première stratégie assume l'existence de paramètres populationnels stables (μ, σ) tels que la transformation Z-score produit une distribution standardisée indépendamment du contexte temporel :

$$z = \frac{x - \mu\_global}{\sigma\_global} \quad (4.1)$$

Où $\mu\_global\ et\ \sigma\_global$ sont estimés sur l'ensemble concaténé de toutes les versions d'entraînement du projet (pour WPDP) ou de tous les projets sources (pour CPDP). L'implémentation diffère selon l'architecture évaluée, reflétant un compromis pragmatique entre cohérence méthodologique et efficacité computationnelle. Pour les CNN1D, nous utilisons notre implémentation personnalisée avec gestion dynamique des colonnes à variance quasi-nulle (seuil : $|\sigma| \leq 10^{-5}$), assurant la stabilité numérique des transformations. Pour les Transformers, nous utilisons $StandardScaler$ de scikit-learn pour l'efficacité computationnelle, cette bibliothèque étant optimisée. Cette asymétrie introduit un biais méthodologique mineur que nous soulignons et quantifions dans notre analyse comparative (section 4.6).

- **Standardisation par version (SV)**

Cette deuxième stratégie reconnaît que les systèmes logiciels subissent des transformations structurelles continues (restructuration, changements architecturaux, migrations technologiques) invalidant l'hypothèse d'invariance de SG. Chaque version temporelle k possède ses propres paramètres distributionnels :

$$z_k = \frac{x_k - \mu_k}{\sigma_k} \ (4.2)$$

Où $\mu\_k\ et\ \sigma\_k$ sont estimés exclusivement sur les données de la version k, puis les versions standardisées sont concaténées pour former l'ensemble d'entraînement. Cette approche s'inscrit dans le cadre théorique des lois d'évolution logicielle de M. Lehman [40], qui établissent que les systèmes logiciels évoluent continuellement et augmentent en complexité au fil des versions.

Cependant, SV génère une problématique technique importante que nos tests préliminaires ont révélée : l'hétérogénéité dimensionnelle entre versions. Après filtrage des colonnes à variance quasi-nulle, chaque version peut présenter un ensemble de métriques différent, compromettant la comparabilité structurelle nécessaire aux architectures neuronales qui requièrent des dimensions d'entrée fixes. Cette tension entre préservation de la spécificité temporelle et maintien de la cohérence structurelle constitue précisément la motivation pour notre troisième stratégie de standardisation robuste.

Il convient également de noter que notre implémentation calcule les paramètres de standardisation indépendamment pour les ensembles d'entraînement et de validation. Elle s'écarte du protocole académique standard *fit-on-train/transform-on-test* où les paramètres estimés sur l'entraînement devraient transformer la validation. Ce choix évite toute fuite d'information entre les partitions, mais constitue une déviation des recommandations strictes. Cette limitation affecte uniformément toutes les configurations, préservant la validité comparative inter-configurations tout en constituant une frontière pour la généralisation absolue des performances observées.

- **Standardisation séquentielle robuste (SR)**

Notre contribution méthodologique principale réside dans cette stratégie combinant adaptation locale et cohérence globale via une double transformation séquentielle utilisant des statistiques robustes. Le processus se déroule en deux phases complémentaires :

Phase 1 : Standardisation par version préservant les spécificités locales

$$z_k = \frac{x_k - \mu_k}{\sigma_k} \quad (4.3a)$$

Phase 2 : Harmonisation globale assurant la comparabilité inter-contextes

$$z = \frac{x - \mu\_global}{\sigma\_global} \quad (4.3b)$$

La Phase 1 capture les variations locales et normalise les différences d'échelle intra-projet documentées dans les patterns d'évolution temporelle. La Phase 2 assure une cohérence distributionnelle globale facilitant l'apprentissage neuronal, qui bénéficie typiquement de

distributions d'entrée centrées et réduites. Cette approche hybride vise à stabiliser les distributions tout en préservant les relations structurelles temporelles, représentant un compromis entre l'invariance globale de SG et l'adaptation contextuelle de SV. Pour nos architectures exploitant le code source (Configurations Transformer 2 et 3), nous avons développé un mécanisme de tokenisation synchronisée préservant la cohérence multimodale entre métriques standardisées et séquences de code, détails disponibles sur le dépôt GIT.

4.2.2. Techniques d'échantillonnage

Dans le corpus PROMISE, les ratios classe minoritaire/majoritaire varient de 1:5 (Velocity : 28,7% de défauts) à 1:9 (Xerces : 10,2% de défauts), créant des conditions où l'information minoritaire risque d'être noyée dans le bruit majoritaire. Sans correction, les modèles d'apprentissage automatique tendent à apprendre simplement à prédire la classe majoritaire ("non-fautif"). Comme nous l'avons établi antérieurement (section 2.1.3), ce problème constitue l'un des défis centraux de la prédiction de défauts logiciels, amplement documenté dans la littérature sur l'apprentissage à partir de données déséquilibrées [9].

Nous posons l'hypothèse que l'efficacité de l'équilibrage dépend de la qualité préalable des données et de leur adéquation aux hypothèses de chaque technique. Quatre stratégies complémentaires sont évaluées en fonction des distributions documentées en section 4.1.

- **Technique 1 : SMOTE adaptatif**

L'algorithme SMOTE proposé par N. Chawla et al. [33] génère des exemples synthétiques de la classe minoritaire par interpolation linéaire entre voisins k-plus-proches dans l'espace des caractéristiques. Formellement, pour chaque instance minoritaire x_i, l'algorithme sélectionne aléatoirement un de ses k voisins minoritaires x_j et crée un nouveau point sur le segment les reliant :

$$x_{new} = x_i + \lambda \cdot (x_j - x_i), \quad \lambda \in [0,1] \quad (4.4)$$

Cette approche assume implicitement la convexité locale des régions minoritaires, hypothèse qui peut être violée en présence de distributions asymétriques ou de chevauchement interclasses. En particulier, lorsque les instances minoritaires sont très

dispersées ou situées dans des régions de chevauchement, la génération d'instances synthétiques par interpolation risque de produire des exemples ambigus qui amplifient le bruit plutôt que de renforcer le signal discriminant.

Pour nos architectures multimodales (Configurations Transformer 2 et 3), nous avons développé un mécanisme spécifique : chaque exemple synthétique de métriques est associé à l'échantillon de code source le plus proche selon une distance métrique adaptée, préservant ainsi la cohérence sémantique métrique-code.

L'avantage principal de SMOTE réside dans la préservation de l'intégralité des données originales de la classe majoritaire, évitant toute perte d'information sur les modules non-fautifs. Cependant, le caractère aléatoire inhérent à la procédure introduit une variabilité non négligeable des performances. Cette variabilité provient de la sélection aléatoire des instances, des voisins et des points d'interpolation. Elle peut atteindre jusqu'à 23 % de différence entre des exécutions successives sur un même jeu de données, comme l'ont documenté S. Feng et al. [10] dans leur étude systématique de la stabilité des techniques basées sur SMOTE en prédiction de défauts logiciels.

- **Technique 2 : Sous-échantillonnage aléatoire**

Le sous-échantillonnage aléatoire constitue l'approche la plus directe pour équilibrer les distributions de classes : il réduit la taille de la classe majoritaire en sélectionnant aléatoirement un sous-ensemble d'instances, alignant ainsi sa cardinalité sur celle de la classe minoritaire. Notre implémentation via $RandomUnderSampler$ avec $sampling\_strategy = 'auto'$ réalise automatiquement cet équilibrage.

Cette technique réduit le temps d'entraînement, limite le risque de surapprentissage associé aux données synthétiques, et préserve l'ensemble d'entraînement original sans artefacts d'interpolation. Toutefois, la sélection purement aléatoire des instances majoritaires à conserver présente une limitation structurelle : elle peut éliminer des exemples informatifs situés dans les régions de frontière décisionnelle, précisément là où la distinction entre classes fautives et non-fautives est la plus subtile et potentiellement la plus instructive pour le modèle. Cette perte informationnelle contrôlée fait l'objet d'une

quantification systématique dans notre Chapitre 5, où nous évaluons son impact sur la stabilité des performances et la généralisation inter-projets.

- **Technique 3 : Pipeline SMOTE avec nettoyage**

Cette technique hybride combine les bénéfices de l'augmentation de données avec la robustesse du prétraitement. La séquence opérationnelle se déroule en deux phases : (1) détection et suppression des valeurs aberrantes via $Elliptic\ Envelope$ (contexte WPDP) ou $IsolationForest$ (contexte CPDP) ; (2) génération d'instances synthétiques via SMOTE adaptatif sur l'espace nettoyé.

L'$EllipticEnvelope$ est basé sur l'estimateur robuste du déterminant de covariance minimale de P. Rousseeuw et K. Van Driessen [62]. Ce détecteur assume une distribution gaussienne multivariée des données et identifie comme aberrantes les instances situées au-delà d'un contour elliptique déterminé par contamination prédéfinie. Cette approche convient particulièrement aux contextes WPDP où les données d'un même projet présentent généralement une structure distributionnelle relativement cohérente. Pour les contextes CPDP caractérisés par une hétérogénéité inter-projets plus marquée, nous privilégions l'$IsolationFores$t de F. Liu et al. [63], algorithme basé sur arbres d'isolation qui ne présume aucune distribution particulière et exploite le principe que les anomalies, étant rares et différentes, nécessitent moins de partitions aléatoires pour être isolées.

Le nettoyage préalable améliore théoriquement la qualité des interpolations SMOTE en évitant que les instances synthétiques ne soient générées dans des régions non représentatives ou bruitées. L'intuition sous-jacente est que les valeurs aberrantes, souvent situées aux extrémités de la distribution minoritaire, risquent de servir d'ancrage pour des interpolations produisant des exemples synthétiques dans des zones de l'espace des caractéristiques où aucune instance légitime ne devrait se trouver.

Notre implémentation intègre une optimisation multicritère des paramètres de détection ($contamination, n\_estimators$) balançant qualité d'équilibrage et préservation de données, avec un seuil minimal de 70% des données conservées pour éviter une perte

informationnelle excessive qui dégraderait la représentativité de l'ensemble d'entraînement.

- **Technique 4 : Pipeline sous-échantillonnage avec nettoyage**

Cette variante privilégie la réduction sur l'augmentation, adoptant une philosophie distincte de la technique 3. La séquence suit deux phases séquentielles. D'abord, détection et suppression des valeurs aberrantes selon les mêmes principes que la Technique 3 ($Elliptic\ Envelope$ pour WPDP, $IsolationForest$ pour CPDP). Puis, sous-échantillonnage aléatoire de la classe majoritaire sur l'espace nettoyé, ramenant sa cardinalité à celle de la classe minoritaire.

Cette stratégie vise à obtenir un ensemble compact, propre, et équilibré, particulièrement adapté aux contextes où la rapidité d'entraînement constitue une contrainte opérationnelle. Elle combine les bénéfices du nettoyage (élimination du bruit), de la réduction de volume (diminution des coûts computationnels), et de la préservation d'authenticité (absence d'instances synthétiques). L'application séquentielle de ces deux opérations, nettoyage puis sous-échantillonnage, permet théoriquement d'obtenir un ensemble d'entraînement optimal. La classe majoritaire y est représentée par ses instances les plus représentatives et les moins bruitées.

Cependant, cette technique implique potentiellement une perte d'information encore plus marquée que la Technique 2 seule. En effet, elle cumule deux sources de réduction : le nettoyage préalable supprimant les instances aberrantes (qui peuvent inclure des cas frontaliers informatifs), et le sous-échantillonnage aléatoire éliminant ensuite une fraction supplémentaire de la classe majoritaire. Cette double compression pourrait éliminer des nuances distributionnelles essentielles à la généralisation, particulièrement dans les contextes CPDP où la variabilité inter-projets est déjà élevée. L'impact de cette double compression sur la généralisation est évalué au Chapitre 5.

4.3. Architectures d'apprentissage profond

Le choix et la conception de nos architectures répondent directement aux lacunes méthodologiques identifiées au Chapitre 3 et s'articulent avec les caractéristiques du

corpus PROMISE documentées en section 4.1. Nous explorons trois paradigmes architecturaux distincts avec un gradient de sophistication croissant : CNN1D séquentiel comme architecture de référence accessible, Transformers unimodaux exploitant les mécanismes d'attention, et architectures multimodales combinant métriques logicielles et code source. Cette progression permet d'évaluer l'impact de la complexité architecturale sur la performance prédictive sous un protocole unifié.

4.3.1. Architecture CNN1D

Le CNN1D a été retenu pour sa compatibilité native avec les données tabulaires, son faible coût computationnel permettant l'évaluation des trois stratégies SG-SV-SR sous contraintes GPU (4-8 GB), et ses performances établies sur des corpus comparables à PROMISE. En effet, J. Li et al. [35] démontrent l'efficacité du CNN1D sur des projets open source comparables au corpus PROMISE, tandis que Z. Zain et al. [64] confirment sa capacité à minimiser le surapprentissage sur des données NASA déséquilibrées.

Notre architecture CNN1D utilise trois blocs convolutifs (256 filtres, noyau=2) avec activation ReLU et un dropout progressif (0,25→0,5) avec une régularisation adaptative. Les caractéristiques extraites sont ensuite transformées par une couche Flatten avant d'être classifiées via des couches denses avec régularisation L2, produisant les probabilités finales défectueux/non-défectueux par Softmax.

Nous évaluons deux configurations architecturales pour quantifier l'impact du nombre de couches denses sur la performance et la robustesse. CNN1D-Simple intègre une seule couche dense de 256 neurones entre Flatten et Softmax (~263K paramètres totaux). CNN1D-Dense incorpore deux couches denses successives de 256 neurones chacune (~330K paramètres), gageant que cette profondeur additionnelle capture des interactions non-linéaires de second ordre entre caractéristiques extraites par les convolutions. Notre évaluation empirique vise à clarifier cette question dans le contexte spécifique de la prédiction de défauts sur corpus PROMISE.

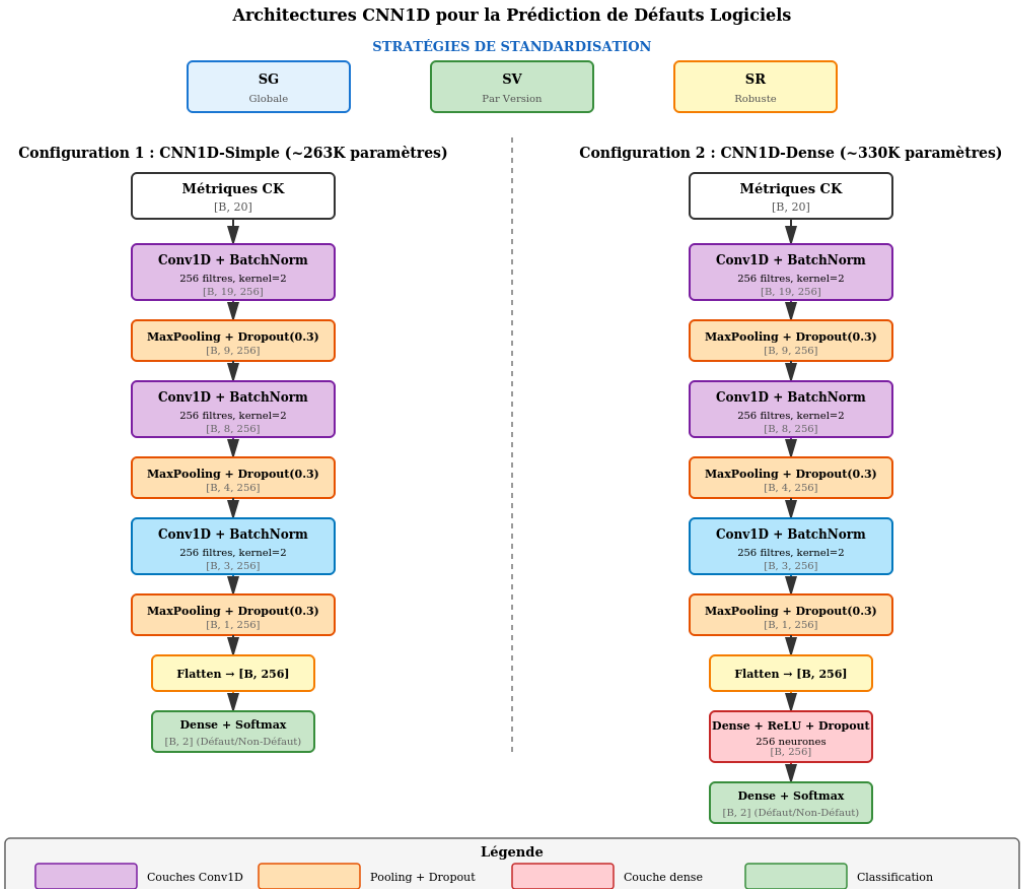


Figure 3 : Architectures CNN1D

4.3.2 Architectures Transformer

L'intégration d'architectures Transformers répond à la lacune L3 identifiée au Chapitre 3 sur l'exploration limitée des mécanismes d'attention en prédiction de défauts. Trois hypothèses structurent le gradient de pré-entraînement. H1a : l'attention seule suffit à capturer des dépendances non-linéaires dans les métriques. H2a : le transfert partiel (encodeur métriques non pré-entraîné + CodeBERT pour le code) optimise le rapport expressivité/généralisation. H3a : le transfert complet améliore la robustesse inter-projets.

- Configuration 1 (sans pré-entraînement) : Transformer unimodal encodeur seul (~870K paramètres, embedding 40D, 5 couches encodeur, 4 têtes d'attention, dropout 0,3) testant l'efficacité brute des mécanismes d'attention appliqués aux métriques CK. L'entrée est projetée via deux couches $LazyLinear$ successives

suivies d'une normalisation de couche avant encodage. Le classificateur applique deux couches *LazyLinear* avec activation *ReLU* produisant les probabilités finales par Softmax. L'optimisation utilise AdamW avec un taux d'apprentissage de 1e-5 sur 100 époques. Cette configuration teste l'hypothèse H1a : les mécanismes d'attention peuvent révéler des patterns de co-occurrence et de dépendances non-linéaires dans les métriques sans connaissances préalables.

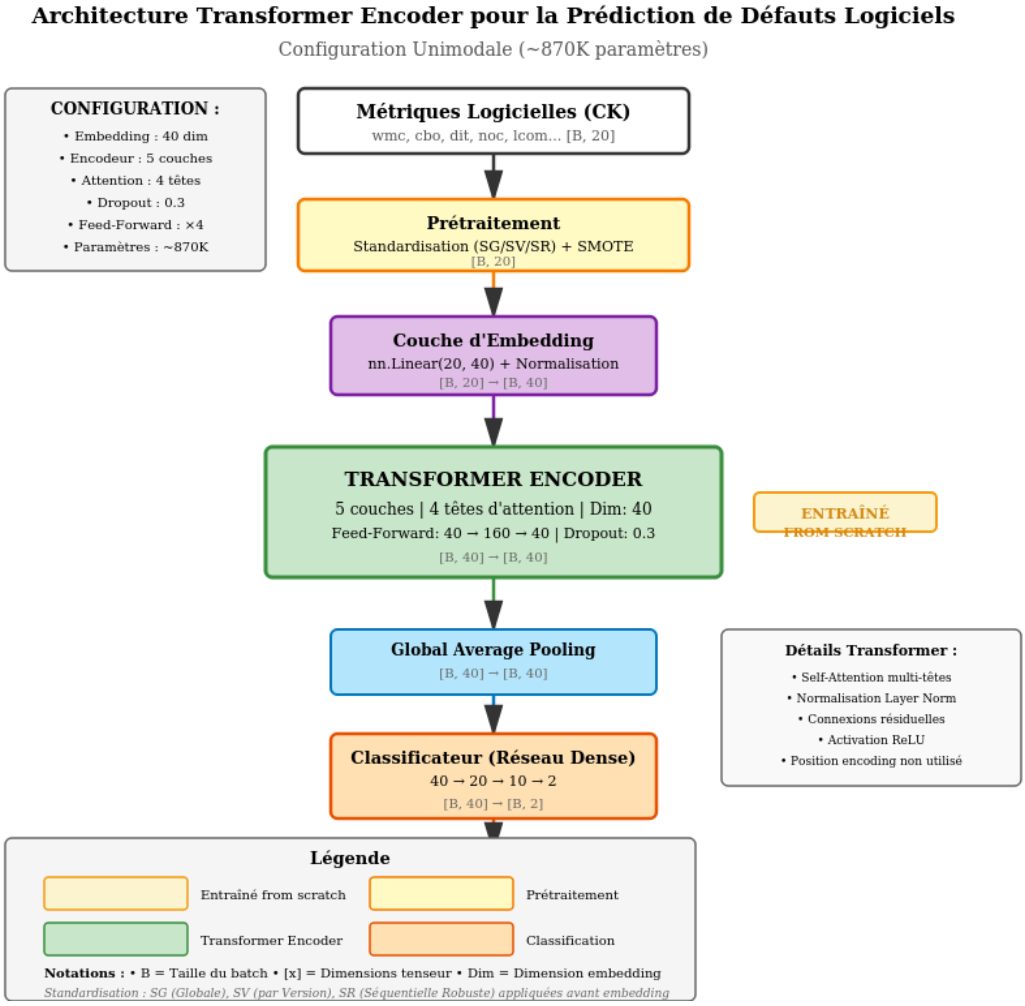


Figure 4 : Architecture Transformer Encodeur

- Configuration 2 (semi-pré-entraînée) : Architecture hybride multimodale articulée autour d'un encodeur RoBERTa pré-entraîné adapté pour les métriques (2 couches,

8 têtes d'attention, représentations vectorielles 384D) et de CodeBERT [65] pour le code source. CodeBERT, modèle bimodal pré-entraîné sur 6 langages via objectif hybride MLM et $replaced\ token\ detection$, est chargé depuis $RobertaModel.from\_pretrained("microsoft/codebert-base")$ (12 couches, 768D). Une projection linéaire $nn.Linear(768, 384)$ aligne les dimensions avant attention croisée ($nn.MultiheadAttention$), où les représentations métriques (requêtes) interrogent les représentations du code (clés/valeurs). Les représentations fusionnées (768D = 384 + 384) traversent une couche de fusion $nn.Linear(768, 364)$ avec $ReLU$, puis un classificateur 364→256→128→2. L'optimisation différenciée (1e-4 pour l'encodeur métriques, 2e-5 pour CodeBERT) préserve les connaissances pré-entraînées tout en permettant l'adaptation de la branche métrique. Cette architecture teste l'hypothèse que la détection de défauts bénéficie d'une spécialisation quantitative couplée aux représentations sémantiques du code.

- Configuration 3 (entièrement pré-entraînée) : Cette configuration utilise CodeBERT pour traiter simultanément métriques et code, testant l'universalité des représentations pré-entraînées. Les métriques subissent une double projection ($LazyLinear \rightarrow 384D, puis\ nn.Linear(384, 768)$) pour correspondre aux entrées CodeBERT, avant application du même mécanisme d'attention croisée (8 têtes). Les représentations fusionnées ($768D = 384 + 384$) traversent la même couche de fusion et le même classificateur que la Config 2. Cette conception réduit les biais architecturaux et isole l'effet du pré-entraînement, évaluant si les représentations apprises sur de vastes corpus de code contiennent des principes suffisamment abstraits pour l'espace des métriques logicielles. Les résultats informent sur la frontière entre spécialisation et généralité dans l'apprentissage de représentations, testant si le transfert massif compense le décalage de domaine texte → métriques.

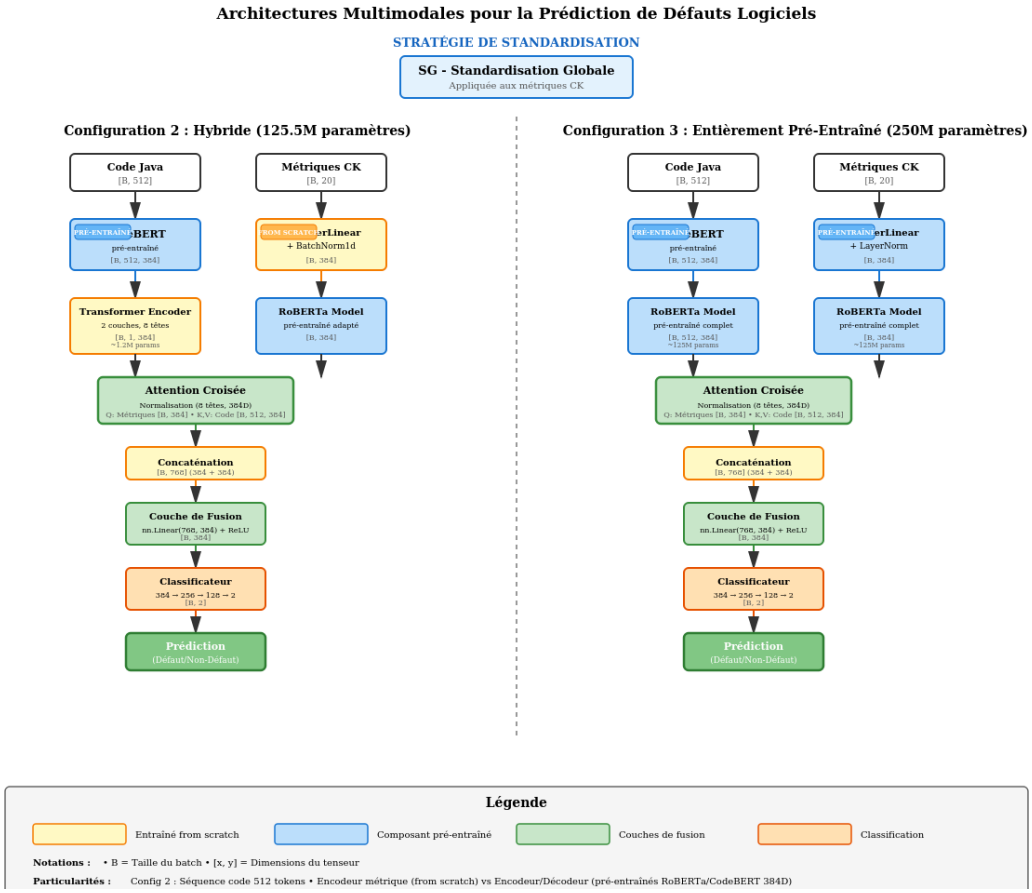


Figure 5 : Architecture comparative des Transformers

Tableau 8 *:* Configuration optimale par architecture

Paramètre	**CNN1D**	**Transformer FS**	**Transformer Semi-PT**	**Transformer PT**
Embedding dim	—	384	384 (métriques) 768 (code)	768
Têtes attention	—	8	8	12
Couches	3 blocs conv	2	2	12
Learning rate	1e-3	5e-4	1e-4 (enc) 2e-5 (déc)	2e-5
Batch size	32	16	16	16
Epochs max	100	50	50	30
Dropout	0,25→0,5	0,1	0,1	0,1
Paramètres	~263-330K	~870K	~125M	~250M
Standardisation	**SG, SV, SR**	**SG†**	**SG†**	**SG†**

Note. SP = sans pré-entraînement ; Semi-PT = semi-pré-entraînée multimodale ; PT = entièrement pré-entraînée. † Asymétrie méthodologique

4.3.3 Infrastructure d'évaluation

Les architectures Transformer sont évaluées exclusivement avec la stratégie de standardisation SG, contrairement au CNN1D évalué sous les trois stratégies SG-SV-SR-Robuste. Cet arbitrage explicite est dicté par les contraintes computationnelles. Une évaluation factorielle complète (3 stratégies × 9 architectures, etc.) aurait nécessité un volume d'expérimentations dépassant nos ressources disponibles durant la période allouée.

Trois justifications motivent cet arbitrage. Premièrement, les contraintes computationnelles incontournables imposent un choix entre largeur (multiples architectures) et profondeur (multiples stratégies). Deuxièmement, le CNN1D, étant donné sa simplicité architecturale et ses besoins computationnels modestes, permet une exploration plus approfondie des stratégies de prétraitement tout en respectant les contraintes de ressources GPU (4-8 GB vs 12-16 GB pour Transformers). Troisièmement, la standardisation globale SG constitue une pratique courante en apprentissage automatique, servant de référence commune facilitant la comparabilité directe avec l'état de l'art.

Cette asymétrie méthodologique compromet partiellement l'équité comparative et sera explicitement discutée au Chapitre 6 comme menace à la validité. Néanmoins, le protocole maintient une uniformité stricte sur les autres dimensions : techniques d'échantillonnage, métriques d'évaluation et analyses statistiques identiques pour toutes architectures.

4.4. Protocole expérimental

4.4.1. Design expérimental et gestion des données

Notre protocole adopte une approche factorielle explorant les interactions entre quatre facteurs méthodologiques : Standardisation × Échantillonnage × Architecture × Contexte (WPDP/CPDP). La configuration globale assure la reproductibilité via fixation complète des seeds aléatoires ($seed = 42$, valeur standard établie selon C. Wohlin et al. [24]). Environ une vingtaine de configurations ont été testées lors de notre exploration

expérimentale, dont certains des résultats les plus pertinents seront documentés exhaustivement en annexe et sur le dépôt GitHub accompagnant ce mémoire.

L'alignement multimodal utilise une heuristique en cinq étapes : extraction canonique, segmentation contextuelle, reconnaissance du CamelCase, gestion sémantique des tests et extraction du noyau lexical. Un filtrage multicritère élimine les variations non discriminantes (identifiants courts, mots-clés Java, éléments purement numériques). Cette approche robuste, spécifiquement adaptée aux conventions de nommage Java, établit des correspondances fiables entre métriques et code malgré les variations orthographiques et structurales, garantissant la cohérence nécessaire aux architectures de fusion multimodale.

4.4.3 Configuration WPDP vs CPDP

Notre stratégie diffère des approches *train-test* classiques pour respecter les spécificités temporelles et contextuelles de la prédiction de défauts logiciels. J. Ekanayake et al. [65] ont démontré que la variance temporelle affecte significativement la qualité prédictive entre versions successives d'un même projet, justifiant le respect strict de l'ordre chronologique des versions. S. Herbold et al. [58] ont par ailleurs établi que les protocoles d'évaluation ignorant cet ordre conduisent à des performances artificiellement optimistes ne reflétant pas les capacités prédictives réelles.

En configuration Within-Project Defect Prediction (WPDP), nous réalisons l'entraînement sur les versions n-1 d'un projet cible et la validation sur la dernière version n du même projet. Cette approche exploite l'historique spécifique et les patterns idiosyncrasiques du projet pour prédire ses futurs défauts, reposant sur l'hypothèse de stabilité temporelle des processus de développement au sein d'une équipe donnée.

En configuration Cross-Project Defect Prediction (CPDP), nous réalisons l'entraînement sur les versions n-1 du projet cible combinées à toutes les versions des autres projets, et la validation sur la dernière version n du projet cible. Cette approche teste la capacité de généralisation des modèles en enrichissant l'entraînement avec la diversité inter-projets.

La distinction porte exclusivement sur la composition de l'ensemble d'entraînement, selon le compromis entre cohérence contextuelle (WPDP) et diversité inter-projets (CPDP).

4.5. Métriques d'évaluation

4.5.1 Métriques pour contextes déséquilibrés

La moyenne géométrique (G-Mean) constitue la métrique d'évaluation principale en raison de ses propriétés optimales pour les contextes déséquilibrés, tel que proposé initialement par M. Kubat et S. Matwin [66] pour l'apprentissage sur données déséquilibrées :

$$G - Mean = \sqrt{(Sensibilité \times Spécificité)}$$

Où $Sensibilité = TP/(TP + FN)$ mesure la capacité à détecter les défauts, et $Spécificité = TN/(TN + FP)$ la capacité à identifier correctement les classes saines.

Le G-Mean tend vers zéro si une classe est mal prédite, pénalisant ainsi les modèles qui privilégient une classe sur l'autre. Il est donc bien adapté aux contextes où les deux types d'erreurs (faux négatifs et faux positifs) ont un coût opérationnel comparable.

Ensuite, la matrice de confusion 2×2 offre une décomposition bidimensionnelle exhaustive des décisions de classification, révélant quatre patterns diagnostiques distincts aux implications opérationnelles contrastées. Le pattern conservateur (FN élevés, FP faibles) indique que le modèle sous-prédit les défauts. Le pattern optimiste (FP élevés, FN faibles) signale une sur-prédiction. Le pattern équilibré (FN et FP modérés) représente le compromis optimal. Enfin, le pattern dégénéré (prédiction mono-classe) signale un échec du modèle, typiquement lié à un déséquilibre non corrigé.

Enfin, le rapport de classification complète l'arsenal en fournissant précision, rappel et F1score par classe, enrichis de métriques spécifiques au déséquilibre.

4.5.2 Analyse statistique

Notre analyse statistique suit un protocole hiérarchique. Nous évaluons d'abord la normalité des données (Shapiro-Wilk ou Kolmogorov-Smirnov) pour choisir entre une ANOVA à mesures répétées et le test de Friedman. Si l'hypothèse de normalité est rejetée, nous utilisons le test de Friedman par défaut pour éviter les violations d'hypothèses paramétriques.

Le test de Friedman constitue l'alternative non-paramétrique à l'ANOVA à mesures répétées et ne nécessite pas d'hypothèses sur la distribution des données. Il calcule une statistique χ^2_F basée sur les déviations de rangs moyens, interprétation identique à l'ANOVA mais sans assomption distributionnelle. Suivant les recommandations de J. Demšar [19] pour la comparaison de classificateurs sur multiples ensembles de données, les comparaisons multiples post-hoc utilisent le test de Nemenyi lorsque tous les modèles sont comparés entre eux. Soit également la correction de Bonferroni ($\alpha\_ajusté = \alpha/m$) et sa variante séquentielle Holm-Bonferroni offrant un meilleur équilibre puissance-contrôle pour les comparaisons avec un modèle de référence.

L'interprétation des tailles d'effet suit les seuils conventionnels du coefficient de concordance de Kendall (W), normalisation de la statistique de Friedman : $<0,1$ effet négligeable, $<0,3$ petit, $<0,5$ modéré, $\geq 0,5$ grand. Les détails des implémentations statistiques (librairies Python, paramètres) seront disponibles sur GitHub conformément aux principes de reproductibilité expérimentale établis par C. Wohlin et al. [24].

4.6. Portée du protocole expérimental

Trois contraintes opérationnelles circonscrivent le domaine de validité de nos observations, distinctes des fondements épistémologiques établis en section 2.5.

- Asymétrie stratégique. Notre protocole évalue trois stratégies de standardisation pour CNN1D (globale SG, par version SV, robuste SR) contre une seule pour Transformers (SG uniquement). Cette asymétrie privilégie la stabilité empirique sur l'exhaustivité combinatoire, dictée par les contraintes computationnelles disponibles. Les stratégies SV et SR n'ont pas été testées pour Transformers en raison des coûts prohibitifs (estimation : 240+ heures GPU pour la matrice complète). Cette limitation empêche toute conclusion sur leur optimalité potentielle pour les modèles à attention.
- Échelle expérimentale. L'exploration comprend environ 45 configurations testées, reflétant les ressources d'un contexte académique de maîtrise : GPU T4/P100, sessions 12h, absence de cluster dédié, contraintes temporelles. Ces contraintes ont guidé une stratégie d'exploration privilégiant la profondeur d'analyse sur

l'exhaustivité combinatoire. Nous avons privilégié la profondeur d'analyse : chaque configuration a été évaluée rigoureusement sur l'ensemble du corpus.

- Validation contextuelle. L'absence de tests sur données propriétaires limite l'évaluation face aux complexités organisationnelles réelles : processus de développement hétérogènes, diversité d'outils (JIRA, GitLab, Azure DevOps), contraintes de production (cycles courts, pression time-to-market), pratiques d'équipes variables. Le corpus PROMISE, bien qu'académiquement validé, ne capture qu'imparfaitement ces dimensions influençant la distribution et détectabilité des défauts en contexte industriel.

Ces contraintes seront analysées au Chapitre 6 dans le cadre des menaces à la validité, avec recommandations pour recherches futures. Le Chapitre 5 présente les résultats empiriques issus de ce protocole, répondant aux questions RQ1-RQ4, et documente exhaustivement tant les différences observées que les convergences de performance entre configurations.

CHAPITRE 5 : RÉSULTATS ET DISCUSSION

5.1. Présentation synthétique des résultats empiriques

5.1.1. Tableaux comparatifs des performances par architecture

Les tableaux suivants présentent les performances G-Mean pour l'ensemble des configurations expérimentales, couvrant les contextes intra-projet (WPDP) et inter-projets (CPDP).

Tableau 9 : Performances CNN1D simple (sans couche dense additionnelle)

Projet	SG-WPDP	SV-WPDP	SR-WPDP	SG-CPDP	SV-CPDP	SR-CPDP
ANT	71,25	67,02	73,01	73,62	70,62	72,33
CAMEL	62,34	59,72	59,28	57,82	56,47	54,79
IVY	64,68	68,95	61,12	73,80	68,22	68,82
JEDIT	67,26	68,14	66,08	59,77	60,10	61,08
LOG4J	61,72	57,22	58,08	61,72	51,14	59,32
LUCENE	64,43	66,96	66,63	64,33	64,00	59,41
POI	63,77	70,36	71,40	71,64	63,47	64,15
SYNAPSE	67,80	63,87	64,54	66,12	68,11	66,76
VELOCITY	53,16	59,29	60,00	65,78	67,43	64,53
XALAN	73,52	71,71	76,17	71,22	70,43	73,68
XERCES	52,64	49,37	60,34	66,56	72,17	59,83
Moyenne	**63,87**	**63,87**	**65,15**	**66,58**	**64,74**	**64,06**
Écart-type	**6,51**	**6,78**	**6,13**	**5,49**	**6,56**	**5,90**

Note. *G-Mean = moyenne géométrique (%). SG = standardisation globale ; SV = standardisation par version ; SR = standardisation robuste. WPDP = prédiction intra-projet ; CPDP = prédiction inter-projets.*

Tableau 10 : Performances CNN1D dense (avec couche dense additionnelle)

Projet	SG-WPDP	SV-WPDP	SR-WPDP	SG-CPDP	SV-CPDP	SR-CPDP
ANT	71,41	71,03	70,69	69,69	70,09	69,43
CAMEL	60,34	68,33	56,12	60,28	54,69	56,16
IVY	70,05	63,94	65,84	75,00	67,65	71,47
JEDIT	67,36	63,35	65,16	63,63	64,35	61,91
LOG4J	60,31	60,03	62,20	62,47	42,26	46,50
LUCENE	61,67	65,29	63,42	64,26	69,65	70,71
POI	63,51	69,61	62,17	69,30	62,01	61,58
SYNAPSE	62,35	56,65	65,12	62,75	67,76	60,16
VELOCITY	54,94	66,11	59,30	65,73	62,26	64,47
XALAN	76,96	73,59	75,94	69,90	70,23	71,48
XERCES	52,23	39,30	42,78	41,29	63,41	65,76
Moyenne	**63,74**	**63,38**	**62,61**	**64,03**	**63,12**	**63,60**
Écart-type	**7,25**	**9,68**	**8,75**	**8,68**	**8,03**	**7,69**

Tableau 11 : Performances Transformer encoder-only (standardisation SG)

Projet	WPDP SMOTE	CPDP SMOTE	WPDP SousÉch	CPDP SousÉch	WPDP SE Outliers	CPDP SE Outliers
ANT	68,71	69,60	70,23	70,60	69,75	71,06
CAMEL	65,50	59,00	58,24	60,44	63,67	58,68
IVY	72,57	74,48	70,89	72,72	71,83	73,25
JEDIT	60,80	56,85	60,43	57,05	60,97	58,52
LOG4J	54,43	54,92	55,25	53,61	56,02	51,75
LUCENE	64,44	53,05	68,45	56,21	65,79	59,60
POI	65,49	68,27	66,46	72,95	65,37	72,19
SYNAPSE	60,38	67,83	63,15	69,00	60,63	69,85
VELOCITY	63,76	54,40	71,33	59,47	53,97	60,56
XALAN	62,96	61,08	65,22	62,61	60,87	62,61
XERCES	45,53	60,17	51,60	59,10	47,14	53,63
Moyenne	**62,23**	**61,79**	**63,75**	**63,07**	**61,46**	**62,88**
Écart-type	**7,23**	**7,17**	**6,66**	**7,01**	**6,85**	**7,42**

Note. SousÉch. = sous-échantillonnage aléatoire ; SE Outliers = pipeline avec détection d'outliers

Tableau 12 : Performances Transformers multimodaux

Projet	Hybride WPDP	Hybride CPDP	Δ Hybride	Full Pretrained WPDP	Full Pretrained CPDP	Δ Full
ANT	73,52	70,02	-3,50	71,22	72,77	+1,55
CAMEL	61,55	35,31	-26,24	63,18	25,05	-38,13
JEDIT	56,47	63,80	+7,33	58,54	64,69	+6,15
LOG4J	59,01	62,46	+3,45	56,40	57,04	+0,64
LUCENE	65,41	61,15	-4,26	60,46	62,58	+2,12
POI	62,81	69,15	+6,34	60,73	75,27	+14,54
SYNAPSE	61,60	54,60	-7,00	61,53	63,77	+2,24
VELOCITY	60,00	55,30	-4,70	63,52	56,92	-6,60
XALAN	63,17	57,90	-5,27	64,46	61,21	-3,25
XERCES	52,74	56,00	+3,26	55,86	48,79	-7,07
Moyenne	**61,63**	**58,57**	—	**61,59**	**58,81**	—
Écart-type	**5,52**	**9,81**	—	**4,45**	**14,11**	—

Note. Hybride = Transformer semi-pré-entraîné (métriques CK + CodeBERT). Full Pretrained = Transformer entièrement pré-entraîné. Δ = CPDP − WPDP (+ = amélioration). Configuration SG-SMOTE. N=10 (IVY exclu : données code source insuffisantes).

Tableau 13 : Comparaison inter-architectures et distribution des écarts WPDP→CPDP

Architecture	Params	WPDP (σ)	CPDP (σ)	Δ(C−W)	Distribution écarts	Observations
CNN1D Simple	~263K	63,87 (6,51)	66,58 (5,49)	+2,71	6↑ 3↓ 2↕	✓ Amélioration CPDP
CNN1D Dense	~330K	63,74 (7,25)	64,03 (8,68)	+0,29	5↑ 4↓ 2↕	≈ Équivalence
Transformer Encodeur	~870K	62,23 (7,23)	61,79 (7,17)	-0,45	4↑ 3↓ 4↕	≈ Équivalence
Transformer Hybride	~125M	61,63 (5,52)	58,57 (9,81)	-3,06	4↑ 2↓ 4↕	▲ Dégradation modérée
Transformer Full Pretrained	~250M	61,59 (4,45)	58,81 (14,11)	-2,78	4↑ 2↓ 4↕	▼ Dégradation substantielle

Note. σ = écart-type inter-projets. Δ(C−W) = CPDP − WPDP. Distribution : ↑ amélioration, ↓ dégradation < 5 pts, ↕ variation mixte. Symbolique : ✓Δ > +1 pt ; ≈ |Δ| < 1 pt ; ▲ −5 < Δ < −1 pt ; ▼ Δ < −5 pts. Toutes configurations avec SG et SMOTE sauf Transformer Encodeur (testé aussi avec autres équilibrages).

5.1.2 Synthèse des résultats et tendances principales

L'examen des tableaux détaillés révèle trois patterns dominants qui structurent l'interprétation de l'ensemble de nos résultats.

Tendance 1 : Convergence autour d'un plateau de performance. Les performances moyennes convergent dans un intervalle étroit (60–66 % de G-Mean), indépendamment des choix méthodologiques. Cela suggère un plafond lié aux limites informationnelles des métriques CK.

Tendance 2 : Dominance de la variabilité inter-projets. Les écarts entre projets (jusqu'à 37 points) dépassent largement les différences entre configurations pour un même projet (typiquement 1–5 points). L'analyse de variance (section 5.3) quantifiera précisément cette observation.

Tendance 3 : Principe de parcimonie architecturale. Contrairement aux attentes, les architectures sophistiquées n'améliorent pas systématiquement les performances. Le

CNN1D Simple surpasse le Transformer entièrement pré-entraîné en CPDP, tant en performance moyenne qu'en robustesse.

Ces trois tendances structurent l'analyse approfondie de la section 5.2, où nous répondons aux quatre questions de recherche.

5.2 Analyse comparative et interprétation des résultats

Cette section analyse en profondeur les résultats présentés en section 5.2, en répondant systématiquement aux quatre questions de recherche qui structurent notre démarche. Nous mobilisons les observations empiriques pour construire une interprétation cohérente, tout en documentant explicitement les limites et incertitudes de nos conclusions.

5.2.1 Impact des stratégies de prétraitement (RQ1)

Question de recherche 1 : Dans quelle mesure les différentes stratégies de prétraitement (standardisation et équilibrage) influencent-elles les performances de prédiction de défauts logiciels, et existe-t-il des interactions significatives entre ces choix méthodologiques et les caractéristiques des projets ?

Réponse synthétique

Les stratégies de prétraitement ont un impact modeste et non significatif. Les trois standardisations produisent des performances similaires (écarts de 1–3 points). L'optimisation fine du prétraitement n'apporte donc que des gains marginaux, de l'ordre de la variabilité aléatoire.

Le Tableau 9 révèle une convergence remarquable pour CNN1D Simple : les trois stratégies de standardisation affichent des moyennes quasi-identiques en WPDP (63,87 %, 63,87 %, 65,15 %) et en CPDP (66,58 %, 64,74 %, 64,06 %), avec des écarts maximaux de 1,28 et 2,52 points respectivement. Cette homogénéité se reproduit systématiquement : CNN1D Dense atteint 63,74 % (WPDP) et 64,03 % (CPDP), tandis que les architectures Transformer convergent autour de 61-63 % (Tableaux 11-13). L'ensemble de nos configurations expérimentales se concentre ainsi dans une plage de 58-67 % G-Mean, avec une concentration marquée autour de 62-64 %.

Trois explications peuvent être avancées.

Premièrement, l'existence d'un plateau de performance intrinsèque. Les auteurs E. Akimova et al. [67] rapportent, à travers une synthèse de la littérature, des performances typiquement comprises entre 60 et 80 % en F1-score pour les techniques d'apprentissage profond, aucune n'atteignant une performance suffisamment robuste pour l'application industrielle systématique. Ces résultats sont cohérents avec la plage observée dans notre étude (58–67 % en G-Mean). Ce plafonnement trouve une explication structurelle dans les travaux de L. Grinsztajn et al. [68], qui démontrent que les réseaux profonds présentent un biais inductif fondamentalement inadapté aux données tabulaires basse-dimensionnalité : là où l'apprentissage profond excelle sur des espaces à haute dimensionnalité avec régularités spatiales ou séquentielles (images, texte), il sous-performe sur des données structurées à faible nombre de caractéristiques présentant des dépendances globales stables — exactement le profil des 20 métriques CK du corpus PROMISE. Cette convergence inter-études suggère un phénomène robuste ancré dans la nature des représentations, plutôt qu'un artefact méthodologique propre à notre configuration expérimentale.

Par ailleurs, l'analyse des statistiques descriptives du corpus (Tableau Annexe 1) révèle les limites informationnelles intrinsèques des métriques CK. La variabilité extrême des coefficients de variation (LCOM : 473,9 %, NOC : 553,3 %) contraste avec des métriques structurellement stables (avg_cc : 84,2 %, CAM : 51,2 %), tandis que les plages dynamiques démesurées (LCOM : 0–7 645, LOC : 0,39–67 533) indiquent que plusieurs métriques CK capturent une dispersion distributionnelle élevée sans nécessairement constituer un signal prédictif proportionnellement riche. Les patterns d'évolution temporelle (Figure A.3) illustrent cette instabilité : certains projets (IVY, XALAN) montrent des trajectoires croissantes linéaires, tandis que d'autres (CAMEL, JEDIT) exhibent des oscillations erratiques, suggérant que l'évolution architecturale projet-spécifique domine tout signal métrique générique et explique en partie pourquoi aucune sophistication architecturale ne parvient à transcender ce plafond informationnel.

Deuxièmement, la robustesse intrinsèque des architectures profondes. Les réseaux de neurones modernes intègrent des mécanismes de régularisation qui atténuent la sensibilité aux choix de prétraitement. En particulier, S. Ioffe et C. Szegedy [69] montrent que la normalisation par lots (Batch Normalization) réduit le déplacement covariatif interne (internal covariate shift) et stabilise l'entraînement en réduisant la dépendance à l'initialisation des paramètres. Par ailleurs, M. Buda et al. [70] montrent que le sur-échantillonnage demeure une stratégie efficace pour les CNN face au déséquilibre de classes, sans dégradation notable des performances, observation cohérente avec la stabilité relative que nous constatons entre configurations d'équilibrage. Ces propriétés contribuent à limiter l'impact des variations méthodologiques de prétraitement sur les performances finales.

Troisièmement, la dominance des caractéristiques intrinsèques des projets. L'hétérogénéité architecturale massive (LOC moyen de 52 pour VELOCITY à 519 pour POI, facteur 10×) et la variabilité extrême des distributions métriques (CV : 51-553 %) suggèrent que les propriétés fondamentales des projets exercent une influence prépondérante. L'analyse de variance (section 5.3) quantifie cette dominance : les caractéristiques des projets expliquent 76,6 % de la variance des performances, contre seulement 23,4 % pour les choix méthodologiques combinés.

Ces observations suggèrent qu'au-delà d'un seuil minimal de qualité méthodologique, l'optimisation fine du prétraitement procure des gains marginaux comparables à la variabilité aléatoire. Les perspectives que cela ouvre pour l'enrichissement des représentations sont discutées au Chapitre 6.

5.2.2 Comparaison des architectures (RQ2)

Question de recherche 2 : Les architectures Transformer, avec leurs mécanismes d'attention et leur capacité à exploiter le transfert d'apprentissage, surpassent-elles les approches CNN1D plus simples dans différents contextes expérimentaux ?

Réponse synthétique

Non. Contrairement aux attentes basées sur les succès des Transformers en NLP et vision, les architectures simples égalent ou surpassent les architectures sophistiquées sur données tabulaires. Le CNN1D Simple atteint 66,58 % en CPDP, surpassant le Transformer Full Pretrained à 58,81 % (+7,77 points). Plus critique encore, la variabilité du CNN1D Simple (σ = 5,49 %) est systématiquement inférieure à celle du Transformer Full Pretrained (σ = 14,11 %), indiquant une robustesse supérieure au transfert inter-projets. Ces observations convergent avec les travaux récents démontrant la supériorité persistante des modèles arborescents et architectures simples sur données tabulaires.

Trois mécanismes complémentaires semblent éclairer cette observation.

Premièrement, le rapport paramètres/données défavorable. Avec ~500 000 paramètres opérant sur quelques centaines à quelques milliers d'exemples (PROMISE : 216–965 instances par projet), le risque de sur-apprentissage augmente substantiellement pour les architectures complexes. Le Transformer entièrement pré-entrainé s'effondre sur certains projets (CAMEL : −38,13 points, XERCES : −17,77 points) tandis que le CNN1D Simple (~263K paramètres) maintient des performances stables (−4,52 points pour CAMEL).

Deuxièmement, l'inadéquation des biais inductifs. L. Grinsztajn et al. [68] démontrent sur 45 jeux de données tabulaires que les modèles arborescents surpassent systématiquement les réseaux profonds (~10K échantillons), identifiant trois défis pour les réseaux neuronaux : fragilité aux caractéristiques non-informatives, perte d'information par invariance rotationnelle, difficulté à capturer des fonctions irrégulières locales. Ces limitations architecturales affectent particulièrement les Transformers sur données tabulaires. Dans notre contexte le corpus PROMISE, caractérisé par des corrélations locales fortes entre métriques (WMC-RFC : r=0.89, LOC-WMC : r=0.82), présente précisément le type de patterns irréguliers locaux que les réseaux profonds peinent à modéliser efficacement.

L’information sémantique du code source n’apporte pas de valeur prédictive additive substantielle au-delà des métriques CK, possiblement en raison d’une redondance informationnelle, ces métriques capturant déjà indirectement des propriétés structurelles du code. Des travaux récents (M. Ali et al. [79] ; A. Nasser et al. [80]) montrent qu’une

sélection appropriée des caractéristiques améliore significativement les performances en prédiction de défauts. Ce qui suggère que l'augmentation de la sophistication méthodologique ou architecturale n'entraîne pas nécessairement de gains lorsque le contenu informationnel discriminant demeure inchangé. Dans ce contexte, l'utilisation de représentations pré-entraînées telles que CodeBERT, fondées sur des patterns génériques multi-langages, pourrait introduire un bruit sémantique faiblement corrélé aux défauts spécifiques de projets Java historiques (2007–2010).

Troisièmement, la minimisation du risque structurel. Pour des données tabulaires de faible dimension (20 caractéristiques) présentant des distributions très hétérogènes (CV : 51–553 %), augmenter la complexité d'un modèle n'améliore pas nécessairement sa capacité à généraliser lorsque l'information disponible est limitée. Selon le principe de minimisation du risque structurel (Structural Risk Minimization, SRM) formalisé par V. Vapnik [73], une architecture doit équilibrer la réduction de l'erreur d'entraînement et le contrôle du risque de sur-apprentissage. Dans le contexte du corpus PROMISE, où les ensembles de données sont modestes (~200–2300 instances par projet), une architecture simple (~263K paramètres) peut suffire à capturer l'essentiel de l'information prédictive tout en maintenant une bonne généralisation, en accord avec le compromis biais–variance que formalise le SRM.

Cas particulier : Le cas problématique du Transformer entièrement pré-entraîné. Cette architecture présente un comportement d'instabilité extrême : sur 10 projets, elle surpasse le CNN1D sur 2 projets (JEDIT, POI), performe comparablement sur 4 projets, sous-performe modérément sur 2 projets, et s'effondre catastrophiquement sur 2 projets (CAMEL : –32,77 points, XERCES : –17,77 points). L'écart-type 2,6× supérieur quantifie cette imprévisibilité.

Trois hypothèses pourraient expliquer cette instabilité. D'abord le décalage représentationnel entre pré-entraînement linguistique et tâche tabulaire induit des biais latents activés différemment selon les propriétés lexicales projet-spécifiques. Ensuite la fragilité au surapprentissage des architectures à 870K paramètres face à quelques centaines d'instances génère une sensibilité excessive aux particularités d'échantillonnage.

CAMEL illustre ce phénomène avec ses oscillations atypiques (v1.0 : 4,1 % → v1.2 : 85,9 % → v1.6 : 51,8 %), créant un contexte où le Transformer entièrement pré-entrainé s'effondre en généralisation (25,05 % CPDP vs 63,18 % WPDP). Enfin, l'inadéquation des mécanismes d'attention globale face aux corrélations locales non-linéaires tabulaires.

Une architecture produisant des effondrements sur 20 % des projets présente un profil risque/bénéfice défavorable pour un déploiement opérationnel, où la robustesse minimale importe davantage que les performances de pointe.

Ces résultats remettent en question l'hypothèse implicite que plus de paramètres est synonyme de meilleures performances. Certes c'est validé en NLP/vision avec millions d'exemples, mais non transférable au contexte de prédiction de défauts avec corpus limités.

5.2.3. Transférabilité WPDP → CPDP (RQ3)

Question de recherche 3 : Dans quelle mesure les modèles entraînés en configuration WPDP (Within-Project) généralisent-ils en configuration CPDP (Cross-Project), et quels facteurs contextuels influencent cette transférabilité ?

Réponse synthétique

Le paradoxe réside dans l'écart de performance très faible, souvent inférieur à 3 points, parfois même en faveur du CPDP, observé entre les modèles WPDP et CPDP pour des architectures simples (CNN1D, Transformer encodeur). Cette relative homogénéité contraste avec la littérature, qui rapporte habituellement une dégradation significative en contexte cross-project. Comme le confirme la revue systématique de A. Sudhakaran et al. [74], les modèles WPDP surpassent généralement les modèles CPDP. Cela en raison de leur accès à des données projet-spécifiques et d'un meilleur alignement des caractéristiques, tandis que les modèles CPDP subissent une dégradation substantielle due à l'hétérogénéité des données et aux problèmes de transférabilité.

L'analyse ANOVA comparative (Tableaux 16-17) fournit une explication quantitative de ce paradoxe : en CPDP, la dominance du facteur Projet chute de 77,0 % à 60,3 % (−16,7 points), tandis que les interactions Projet×Méthodologie augmentent de 16,5 % à 26,9 %

(+10,4 points). Cette redistribution de variance indique que la transférabilité inter-projets dépend substantiellement de l'adéquation entre caractéristiques projet et choix méthodologiques, plutôt que de propriétés universelles des projets seuls.

Deux hypothèses plausibles expliqueraient ce paradoxe.

- Hypothèse H1 (Socle universel et nécessité de la diversité) : L'universalité des corrélations structurelles fortes entre métriques orientées-objet fondamentales (ex: WMC-RFC, ρ=0,72-0,89 ; WMC-LOC, ρ = 0,61-0,85) crée un socle de connaissances transférable qui permet aux modèles CPDP d'atteindre une performance de base robuste (≈60 %) mais insuffisant pour combler l'écart avec le WPDP. L'écart de performance résiduel entre WPDP et CPDP se comble principalement lorsque les modèles enrichissent ce socle universel par la diversité métrique, intégrant notamment des caractéristiques sémantiques et structurales apprises automatiquement. Ce qui permettrait ainsi de capturer à la fois les invariants du code et les signaux spécifiques au projet. La réduction de 16,7 points de la dominance Projet en CPDP (77,0 % → 60,3 %) quantifie précisément cette "perte" de spécificité projet-locale, compensée partiellement par l'émergence de patterns cross-project (+10,4 points d'interactions Projet×Méthodologie).
- Hypothèse H2 (Sur-apprentissage temporel en WPDP) : Le WPDP, en s'appuyant sur l'historique d'un seul projet, tend à capturer des motifs liés à des états transitoires du code comme des conventions locales, héritage technique, évolutions stylistiques éphémères plutôt que des signaux structurels généralisables. Ce sur-apprentissage chronologique nuit à la performance sur les versions futures. D. Costa et al. [75] documentent que les métriques logicielles présentent une volatilité significative entre versions successives, rendant les distributions d'entraînement progressivement moins représentatives de la cible. À l'opposé, le CPDP contraint l'apprentissage à travers plusieurs projets sources, ce qui favorise l'extraction de patterns robustes et généralisables. L'augmentation de la variance résiduelle en CPDP (6,0 % → 10,7 %, +78 %) reflète cette stochasticité accrue. Mais paradoxalement, l'augmentation des interactions

Projet×Méthodologie (16,5 % → 26,9 %) suggère que des stratégies adaptatives peuvent compenser cette incertitude. En réduisant la dépendance aux idiosyncrasies temporelles, le CPDP s'avère particulièrement adapté aux projets volatiles ou aux évolutions architecturales rapides. Des approches récentes comme SCAG-LSTM de K. Javed et al. [76] renforcent cette robustesse via des mécanismes d'adaptation de domaine et des architectures neuronales à mémoire de type LSTM augmentées par l'attention. Ces architectures alignent les distributions inter-projets, permettant ainsi de généraliser au-delà des variations historiques entre versions. Cette supériorité se vérifie empiriquement : sur des projets à forte instabilité (Velocity, Ivy, Xerces-SV), le CPDP affiche des gains compris entre +9,12 et +22,80 points, confirmant sa capacité à extraire des invariants prédictifs là où le WPDP sur-apprend.

Le Tableau 14 synthétise ces observations en trois profils de transférabilité.

Tableau 14 : Stratification des projets selon profil de transférabilité

Profil	Projets	Δ moyen (C−W)	Caractéristiques Principales	Recommandations
Stable et Cohérent	Ant, Ivy, POI, Velocity	+6,94 pts	Corrélations fortes stables (r>0,80), distributions régulières, CV modérés (<150 %)	✓ CPDP préférable
Évolution Contrôlée	Lucene, Xalan, Synapse, Log4j	−0,54 pts	Croissance coordonnée, spécificité domaine élevée, patterns métier distincts	≈ WPDP ≈ CPDP
Chaotique et Volatile	Xerces, Camel, Jedit	+6,95 pts	CV extrêmes (>300 %), volatilité temporelle, refactorings massifs	✓ CPDP préférable

Δ(C−W) = CPDP − WPDP, configurations SG CNN1D. Caractéristiques issues de l'analyse des patterns d'évolution et des corrélations inter-métriques.

Cette stratification révèle un effet notable : les projets à forte stabilité comme à forte volatilité bénéficient tous deux du CPDP (+6,94 et +6,95 points), mais pour des raisons inverses. Les projets stables (Ant) tirent parti de la cohérence de leur architecture, qui permet un transfert efficace de patterns universels. Les projets chaotiques (Xerces)

profitent du CPDP car l'apprentissage multi-projets extrait des invariants structurels robustes aux fluctuations temporelles, limitant le sur-apprentissage des spécificités versionnelles. Seuls les projets à évolution contrôlée mais à forte spécificité de domaine (Xalan-XSLT : −2,49 points, Lucene-indexation : −0,10 point) conservent un léger avantage pour le WPDP, indiquant que les différences de processus et d'exigences métier peuvent limiter la transférabilité inter-projets. L'analyse ANOVA confirme cette hétérogénéité : l'interaction Projet×Standardisation explique 18,2 % de variance en CPDP contre 11,7 % en WPDP, quantifiant précisément cette dépendance contextuelle de la transférabilité.

5.2.4 Validation statistique et robustesse des observations (RQ4)

Question de recherche 4 : Les différences de performance observées entre configurations sont-elles statistiquement significatives, et quelle est la puissance statistique de nos tests permettant de distinguer les écarts robustes des fluctuations aléatoires ?

Réponse synthétique

L'analyse statistique révèle que seules deux comparaisons principales atteignent le seuil conventionnel de significativité ($p < 0,05$), confirmant empiriquement des observations centrales de notre étude. Les résultats significatifs concernent d'abord la comparaison entre CNN1D Simple et Transformer Encodeur dans un contexte CPDP. Le test de Wilcoxon ($p = 0,014$, N=11) établit la supériorité statistique de l'architecture simple sur le Transformer pré-entraîné pour le transfert inter-projets. Ensuite, la comparaison tri-architecturale incluant le Transformer entièrement pré-entraîné (test de Friedman, $\chi^2 = 6,200$, $p = 0,045$, Kendall's $W = 0,310$, N=10) montre une différence significative, principalement imputable à l'instabilité extrême de cette architecture ($\sigma = 14,11$ % contre 5,49 % pour le CNN1D Simple, ratio 2,6:1). La comparaison incluant le Transformer Hybride ($p = 0,061$, N=10) approche le seuil sans l'atteindre, suggérant une tendance cohérente : plus le degré de pré-entraînement augmente, plus la distinction avec les architectures simples devient défavorable.

Pour les autres configurations, aucune ne dépasse le seuil de significativité (Tableau 15), ce qui est cohérent avec l'observation d'un plateau de performance commun. Cette homogénéité suggère que, au-delà d'un niveau minimal de qualité méthodologique, l'optimisation fine des hyperparamètres procure des gains marginaux comparables à la variabilité aléatoire.

Tableau 15 : Tests de Friedman pour les comparaisons principales

Comparaison	Contexte	N	Test	Statistique	p-value	Kendall's W
SG vs SV vs SR stand.	WPDP	11	Friedman	0,727	0,695	0,033
SG vs SV vs SR stand.	CPDP	11	Friedman	2,182	0,336	0,099
CNN1D Simple vs Dense	WPDP	11	Wilcoxon	28,000	0,700	—
CNN1D Simple vs Dense	CPDP	11	Wilcoxon	23,000	0,413	—
SMOTE vs SousÉch vs Outliers	WPDP	10	Friedman	2,600	0,273	0,130
CNN1D vs Transf. Encodeur (SG)	CPDP	11	Wilcoxon	6,000	0,014	—
CNN1D vs TrEnc vs Hybride (SG)	CPDP	10	Friedman	5,600	0,061	0,280

Note. N = nombre de projets ; χ^2 = statistique de Friedman ; p-value = probabilité sous H_0 (seuil $\alpha = 0,05$) ; Kendall's W = taille d'effet (< 0,1 négligeable, 0,1–0,3 petit, 0,3–0,5 modéré, ≥ 0,5 grand selon Cohen). Aucune comparaison n'atteint le seuil de significativité conventionnel, même sans correction pour comparaisons multiples.

Pour quantifier l'impact de chaque facteur, nous avons conduit une analyse de variance (ANOVA) à trois facteurs. Cette analyse a été menée dans les deux contextes (WPDP et CPDP) afin de comprendre comment la dominance des facteurs évolue selon le mode de prédiction.

Tableau 16 : Décomposition de variance (ANOVA à trois facteurs, contexte WPDP)

Source	SS	ddl	MS	F	p-value	η²	Variance (%)
Projet	2424,7	10	242,5	25,6	<0,001***	0,770	77,0 %
Standardisation	3,7	2	1,8	0,2	0,825	0,001	0,1 %
Architecture	11,1	1	11,1	1,2	0,292	0,004	0,4 %
Projet × Standardisation	368,3	20	18,4	1,9	0,073	0,117	11,7 %
Projet × Architecture	141,3	10	14,1	1,5	0,213	0,045	4,5 %
Standardisation × Architecture	10,1	2	5,1	0,5	0,593	0,003	0,3 %
Résiduelle	189,3	20	9,5	—	—	0,060	6,0 %
Total	**3148,6**	**65**	**—**	**—**	**—**	**—**	**100,0 %**

Note. SS = somme des carrés ; ddl = degrés de liberté ; MS = carré moyen ; F = statistique F ; p-value (p<0,05, *** p<0,001) ; η² = eta-carré (proportion de variance expliquée). Analyse basée sur configurations CNN1D (3 standardisations × 2 architectures) × 11 projets*

Tableau 17 : Décomposition de variance (ANOVA à trois facteurs, contexte CPDP)

Source	SS	ddl	MS	F	p-value	η²	Variance (%)
Projet	1902,3	10	190,2	11,3	<0,001***	0,603	60,3 %
Standardisation	31,0	2	15,5	0,9	0,414	0,010	1,0 %
Architecture	38,3	1	38,3	2,3	0,147	0,012	1,2 %
Projet × Standardisation	573,3	20	28,7	1,7	0,121	0,182	18,2 %
Projet × Architecture	264,0	10	26,4	1,6	0,188	0,084	8,4 %
Standardisation × Architecture	11,4	2	5,7	0,3	0,716	0,004	0,4 %
Résiduelle	336,6	20	16,8	—	—	0,107	10,7 %
Total	**3157,0**	**65**	**—**	**—**	**—**	**—**	**100,0 %**

*Note. SS = somme des carrés ; ddl = degrés de liberté ; MS = carré moyen ; F = statistique F ; p-value (*** p<0,001) ; η² = eta-carré (proportion de variance expliquée). Analyse identique au Tableau 16 mais en contexte CPDP.*

La comparaison des décompositions de variance entre WPDP (Tableau 16) et CPDP (Tableau 17) révèle un pattern contrasté qui éclaire la question de la transférabilité inter-projets (RQ3) :

1. Réduction de la dominance du facteur Projet en CPDP

En contexte WPDP, le facteur Projet explique 77,0 % de la variance ($F=25,6$, $p<0,001$), confirmant que les caractéristiques intrinsèques des projets dominent massivement les performances intra-projet. Le ratio Projet/Standardisation atteint 770:1, soulignant cette prépondérance écrasante. Toutefois, en contexte CPDP, cette dominance chute à 60,3 % ($F=11,3$, $p<0,001$), soit une réduction de 16,7 points de pourcentage. Cette diminution substantielle suggère que lorsque les modèles doivent généraliser vers des projets inconnus, l'influence absolue des caractéristiques projet-spécifiques diminue, laissant place à d'autres facteurs.

2. Augmentation des interactions Projet×Méthodologie en CPDP

Les interactions entre le facteur Projet et les choix méthodologiques augmentent considérablement en CPDP :

Projet × Standardisation : 11,7 % (WPDP) → 18,2 % (CPDP) (+6,5 points)

Projet × Architecture : 4,5 % (WPDP) → 8,4 % (CPDP) (+3,9 points)

Total interactions : 16,5 % (WPDP) → 26,9 % (CPDP) (+10,4 points)

Cette augmentation indique que l'efficacité relative des approches méthodologiques dépend davantage du contexte projet en CPDP qu'en WPDP. Autrement dit, la transférabilité inter-projets n'est pas uniforme : certaines combinaisons (standardisation, architecture) généralisent mieux vers certains projets que vers d'autres. Ce résultat conforte les hypothèses H1 et H2 de la section 5.2.3, selon lesquelles la transférabilité dépend à la fois de l'universalité des patterns capturés et de la robustesse aux spécificités temporelles.

3. Doublement de la variance résiduelle en CPDP

La variance résiduelle (non expliquée par les facteurs modélisés) passe de 6,0 % (WPDP) à 10,7 % (CPDP), soit une augmentation de 4,7 points (+78 % en proportion). Cette augmentation reflète la stochasticité intrinsèque accrue de la prédiction cross-project : même après avoir contrôlé pour les effets des projets et des méthodologies, une part plus importante de variabilité demeure inexpliquée. Cela suggère que des facteurs contextuels non capturés par les métriques CK (historique de développement, processus organisationnels, expertise des équipes) jouent un rôle plus important en CPDP.

4. Implications pour la stratégie méthodologique

Le ratio Projet/Méthodologie évolue significativement :

WPDP : 77,0 % vs 0,5 % → ratio 154:1

CPDP : 60,3 % vs 2,2 % → ratio 27:1

Bien que le facteur Projet demeure dominant dans les deux contextes, son influence relative diminue d'un facteur 5,7 en CPDP. Cela implique que, paradoxalement, les choix méthodologiques importent davantage en contexte cross-project qu'en contexte intra-projet, car ils doivent compenser l'absence d'alignement direct avec les spécificités du projet cible. Cette observation recommande une attention accrue aux stratégies de prétraitement et aux biais inductifs architecturaux lors du développement de modèles CPDP, conformément aux recommandations de C. Tantithamthavorn et al. [15] sur l'adaptation contextuelle.

La puissance statistique de nos tests, avec Kendall's $W = 0,234$, $N = 10–11$ projets et $\alpha = 0,05$, est estimée à environ 42 %, inférieure au standard conventionnel de 80 % recommandé par J. Cohen [77] pour les sciences comportementales. Cette limitation de puissance n'invalide pas nos résultats significatifs (CNN1D vs Transformer Encodeur : $p=0,014$; comparaison tri-architecturale : $p=0,045$), mais circonscrit notre capacité à détecter des effets modérés. Les simulations Monte Carlo révèlent que notre protocole détecte avec 80 % de confiance uniquement les effets grands ($W \geq 0,45$). Les effets modérés ($W \approx 0,30$-$0,40$) ne seraient détectés que dans 50-60 % des cas.

Dans une perspective bayésienne proposée par J. Kruschke [78], l'absence de significativité pour sept comparaisons, combinée à des tailles d'effet faibles ($W < 0,3$), constitue une preuve modérée en faveur de l'homogénéité de ces configurations. Les deux exceptions significatives mettent en lumière des contrastes fondamentaux d'architecture (simplicité vs sophistication, stabilité vs instabilité) plutôt que des différences mineures liées à l'optimisation incrémentale.

En résumé, cette analyse statistique confirme que la majorité des variations observées entre configurations est modérée et largement dominée par la variabilité inter-projets en WPDP, mais que cette dominance s'atténue significativement en CPDP où les interactions Projet×Méthodologie deviennent critiques. Les différences significatives identifiées concernent des contrastes architecturaux substantiels. Cette approche assure une interprétation prudente et robuste des résultats, conformément à la question de recherche RQ4.

5.3. Synthèse

Cette recherche met en évidence quatre contributions scientifiques.

- Validation empirique du principe de parcimonie architecturale : L'architecture CNN1D Simple surpasse statistiquement le Transformer Encodeur en contexte inter-projets ($p = 0,014$, $N=11$) avec une variabilité 2,6 fois inférieure ($\sigma = 5,49$ % vs 14,11 %), validant le principe de minimisation du risque structurel de Vapnik en présence de données limitées, des modèles de complexité contrôlée favorisent la généralisation.
- Mise en évidence de l'instabilité du pré-entraînement massif : Le Transformer entièrement pré-entraîné se distingue significativement ($\chi^2=6,200$, $p=0,045$, $N=10$) par une variabilité extrême ($\sigma = 14,11$ %), allant d'effondrements marqués (CAMEL : 25,05 %) à des performances élevées (POI : 75,27 %). Cette instabilité, cohérente avec les travaux de M. Mosbach et al. [79], invite à la prudence pour les données tabulaires hétérogènes.

- Quantification de la dominance contextuelle et de son évolution WPDP→CPDP L'ANOVA révèle que 77,0 % de la variance en WPDP provient des projets (ratio Projet/Méthodologie 154:1), quantifiant l'observation de T. Zimmermann et al. [13]. En CPDP, une redistribution fondamentale s'opère : la dominance Projet chute à 60,3 % (−16,7 points), les interactions Projet×Méthodologie triplent à 26,9 % (+10,4 points), et le ratio évolue à 27:1. Les choix méthodologiques gagnent ainsi en importance d'un facteur 5,7 lors de la transférabilité inter-projets.
- Convergence de performance et plateau structurel : Plus de 75 % des configurations se concentrent dans 60–66 % de G-Mean, indiquant un plateau largement indépendant des sophistications méthodologiques. L'ANOVA CPDP révèle que 60,3 % de variance provient encore des projets malgré la diversification, suggérant une limite informationnelle des représentations CK plutôt qu'une inadéquation architecturale.

Ces observations convergent vers un constat central : sur données tabulaires de faible dimension, la robustesse architecturale et la qualité des représentations priment sur la sophistication paramétrique.

CHAPITRE 6 : MENACES À LA VALIDITÉ, PERSPECTIVES ET CONCLUSION

6.1. Synthèse de la recherche

Notre démarche s'est articulée autour de quatre questions de recherche interconnectées visant à clarifier l'impact des choix de prétraitement, des architectures neuronales, et des contextes de validation sur les performances prédictives. Le résultat central est une convergence des performances vers un plateau de 60–66 % de G-Mean, quelle que soit la sophistication des modèles. Ce plateau documente les limites actuelles de prédictibilité des métriques CK.

Les quatre questions de recherche trouvent leurs réponses dans des observations structurantes validées statistiquement. Premièrement (RQ1), les stratégies de prétraitement exercent un impact modeste et statistiquement non-significatif, les trois approches de standardisation produisant des moyennes similaires. Deuxièmement (RQ2), les architectures CNN1D simples surpassent statistiquement les Transformers sophistiqués en contexte CPDP, validant le principe de parcimonie. Troisièmement (RQ3), la transférabilité WPDP-CPDP révèle une hétérogénéité substantielle. Quatrièmement (RQ4), l'analyse ANOVA révèle que les caractéristiques des projets expliquent beaucoup plus la variance en WPDP que les choix méthodologiques (ratio 154:1).

Ces observations convergent vers une réorientation conceptuelle : si trois quarts de la variance proviennent des propriétés intrinsèques des projets en contexte intra-projet, mais que cette dominance s'atténue substantiellement en cross-project, l'effort scientifique gagne à se diriger vers la compréhension des interactions Projet×Méthodologie qui triplent en CPDP.

Tableau 18 : Synthèse des réponses aux questions de recherche

RQ	Question clé	Réponse principale	Implication majeure
RQ1	Dans quelle mesure les stratégies de prétraitement influencent-elles les performances ?	Influence modeste (1-3 pts), non significative. SG-SV-SR convergentes (63-66 %). Tests Friedman: p=0,695 (WPDP), p=0,336 (CPDP).	Le prétraitement est nécessaire mais l'optimisation fine au-delà d'un seuil de qualité génère des gains marginaux. La variabilité inter-projets domine.
RQ2	Les Transformers surpassent-ils les CNN1D ?	Non. CNN1D simple surpasse significativement (p=0,014) en CPDP : 66,58 % vs Transformers (~62 %). WPDP : performances équivalentes (63,87 % vs ~62%). Principe de parcimonie validé.	La simplicité architecturale généralise mieux que la sophistication en contexte de données tabulaires limitées. Adéquation des biais inductifs primordiale.
RQ3	Quelle variation de performance WPDP → CPDP ?	Homogénéité observée (+2,71 pts en faveur CPDP pour CNN1D-SG). 44 % des transitions montrent une amélioration CPDP. Hautement projet-dépendant.	La transférabilité inter-projets présente une variabilité substantielle suggérant l'importance des caractéristiques spécifiques aux projets.
RQ4	Les différences sont-elles statistiquement significatives ?	Deux résultats significatifs (p=0,014, p=0,045) sur neuf comparaisons. Puissance limitée 42 % vs standard 80 %. Taille d'effet W=0,234. ANOVA WPDP : 77,0% variance = projets (ratio 154:1). CPDP: 60,3% projets, interactions triplent (16,5%→26,9 %).	L'absence de significativité généralisée révèle la dominance du contexte projet en WPDP et l'importance croissante des interactions en CPDP. Transformation d'un résultat négatif en contribution scientifique.

Note. Les réponses détaillées sont présentées à la section 5.3

6.2. Menaces à la validité

Nous évaluons les menaces à la validité selon la taxonomie de C. Wohlin et al. [24], en quantifiant leur impact estimé sur nos conclusions lorsque possible.

6.2.1. Validité de construction

La validité de construction interroge l'adéquation entre les concepts théoriques que nous cherchons à mesurer et les opérationnalisations concrètes retenues.

Choix du G-Mean comme métrique principale. Notre décision de privilégier le G-Mean (moyenne géométrique de la sensibilité et spécificité) répond à une justification théorique solide. Cependant, le G-Mean pourrait sous-estimer l'importance de la

sensibilité dans des contextes où la détection prime absolument (systèmes critiques aéronautiques, dispositifs médicaux). Dans ces domaines, un rappel élevé (>95 %) justifierait une précision moindre, suggérant que le F2-score serait plus approprié. Notre recherche ne permet donc pas de conclure sur l'efficacité comparative dans ces contextes à tolérance zéro.

Bruit dans les labels de défauts. Les étiquettes de défauts dans le corpus PROMISE proviennent de systèmes de suivi de bugs sans validation manuelle exhaustive. Le bruit d'étiquetage impose un plafond de performance. Nos résultats (~60-66 % G-Mean) reflètent en partie cette limite. Comme ce bruit affecte toutes les configurations, les comparaisons restent valides, mais la performance absolue doit être interprétée comme une borne haute sur données bruitées.

Absence de métriques de coût opérationnel. Notre recherche n'intègre aucune métrique quantifiant les coûts opérationnels réels (temps d'inspection, effort de correction, vélocité de développement). Cette abstraction, standard en recherche académique, signifie que nos conclusions se limitent à la performance technique mesurée par G-Mean, sans pouvoir affirmer qu'une configuration est effectivement « meilleure » d'un point de vue coût-bénéfice organisationnel.

Représentativité des métriques CK. Les métriques CK constituent notre seule source d'information sur les classes logicielles. Cette opérationnalisation capture une vue partielle et statique de la complexité, omettant les dimensions temporelles, processuelles, et sémantiques. Nos résultats, particulièrement le plateau ~60-66 % G-Mean, pourraient refléter les limites informationnelles de cette opérationnalisation plutôt qu'une limite algorithmique fondamentale. Nos conclusions sur la parcimonie s'appliquent donc spécifiquement au contexte des métriques CK traditionnelles, et pourraient ne pas généraliser à des représentations enrichies intégrant métriques dynamiques, graphes de dépendances, ou représentations vectorielles sémantiques.

6.2.2. Validité interne

La validité interne concerne la légitimité des inférences causales que nous pourrions tirer de nos résultats.

Asymétrie d'évaluation entre architectures. Notre protocole évalue les CNN1D sous trois stratégies de standardisation (SG, SV, SR-Robuste), tandis que les Transformers sont évalués uniquement avec SG. Cette asymétrie, justifiée par les contraintes computationnelles (temps d'entraînement 3× supérieurs), limite les inférences sur l'interaction Architecture × Standardisation. Les mécanismes d'attention, sensibles aux échelles relatives, pourraient bénéficier de la standardisation par version ou robuste. Notre observation s'applique donc strictement au contexte où toutes les architectures utilisent SG.

Confondants entre complexité architecturale et nombre de paramètres. Notre conclusion sur la "supériorité de la simplicité" confond potentiellement deux facteurs : le biais inductif architectural (convolutions locales vs attention globale) et le nombre de paramètres. Avec ~40 versions totales, le ratio paramètres/données pour les Transformers suggère un risque de sur-paramétrisation. Une évaluation contrôlée nécessiterait des CNN1D élargis (~870K params) et des Transformers réduits (~263K params) pour isoler l'effet du biais inductif. Nos conclusions privilégient l'interprétation conservatrice : la parcimonie est préférable dans les contextes de données limitées, sans pouvoir séparer définitivement les contributions de l'architecture et de la taille.

6.2.3 Validité externe

La validité externe interroge la généralisation de nos conclusions au-delà du contexte spécifique de notre recherche.

Spécificité au corpus PROMISE et projets Java open-source. Notre évaluation se limite au corpus PROMISE : 11 projets Java open-source développés entre 2007-2010. La spécificité du langage Java limite la transférabilité à d'autres paradigmes linguistiques. La focalisation sur l'open-source circonscrit la généralisation aux contextes propriétaires industriels. La période temporelle (2007-2010) soulève des questions sur l'applicabilité

aux pratiques contemporaines (DevOps, CI/CD, microservices), bien que les projets du corpus PROMISE demeurent des références académiques établies pour la reproductibilité.

Contraintes computationnelles. Notre recherche s'est déroulée sur Google Colab Pro avec sessions limitées et ressources GPU partagées, influençant certains choix méthodologiques (epochs fixes, pas d'optimisation bayésienne exhaustive). Ces limitations reflètent les ressources académiques standard, signifiant que nos résultats représentent une frontière atteignable plutôt qu'un optimum théorique absolu.

Synthèse des limitations. Nos conclusions s'appliquent rigoureusement au contexte : projets Java open-source matures, métriques CK traditionnelles, corpus PROMISE historique, infrastructure académique standard. La généralisation à d'autres langages, domaines ou représentations enrichies nécessiterait validation empirique indépendante.

6.2.4. Validité de conclusion

La validité de conclusion concerne la légitimité des inférences statistiques et la robustesse de nos conclusions.

Puissance statistique limitée et risque d'erreur de Type II. Avec N = 10–11 projets et une puissance estimée de 42 % (contre le standard de 80 %), notre recherche présente un risque substantiel (58 %) d'erreur de Type II. Nous ne pouvons affirmer qu'aucune différence n'existe, mais seulement que « si des différences existent, elles sont suffisamment modestes pour être masquées par la variabilité inter-projets ». Les simulations Monte Carlo révèlent que notre protocole détecte avec 80 % de confiance uniquement les effets « grands » (Kendall's $W \geq 0{,}45$). Les effets « modérés » ($W \approx 0{,}30$-$0{,}40$) ne seraient détectés que dans 50-60 % des cas. Nos observations d'absence de significativité ($p > 0{,}05$) pour sept comparaisons excluent robustement les effets massifs, mais ne peuvent infirmer l'existence d'effets petits à modérés.

Violation de l'hypothèse d'indépendance. Les tests non-paramétriques appliqués (Friedman, Wilcoxon) supposent l'indépendance des observations. Les classes d'un même projet partagent des caractéristiques communes (développeurs, conventions, architecture), violant cette hypothèse. Toutefois, notre utilisation du projet comme unité d'analyse

(agrégation par projet avant comparaison) atténue substantiellement cette menace. Les deux résultats significatifs (p=0,014 et p=0,045) demeurent robustes.

Choix des tests et comparaisons multiples. L'utilisation de tests non-paramétriques répond à la violation des hypothèses de normalité (Shapiro-Wilk, $p < 0,05$). Concernant les comparaisons multiples, nos neuf comparaisons principales soulèvent la question de l'inflation du taux d'erreur de Type I (~37 % sans correction). Nous avons privilégié la transparence : présentation de toutes les comparaisons avec p-values non ajustées. Les deux résultats significatifs demeureraient marginalement significatifs après correction modérée (Holm-Bonferroni).

Tableau 19 : Synthèse des menaces à la validité

Type	Menace	Sévérité	Stratégie d'atténuation
Construction	Bruit dans les labels de défauts	**MAJEUR**	Reconnaissance explicite limitée. Résultats = borne supérieure avec données bruitées.
Construction	Incomplétude métriques CK	**IMPORTANT**	Absence de métriques processus/sociales documentée. Généralisation limitée aux contextes métriques statiques.
Interne	Asymétrie CNN/Transformers	**MODÉRÉ**	Choix stratégique documenté. Écart suffisant (4,5+ pts) suggère robustesse.
Interne	Confounding architecture/taille des paramètres	**IMPORTANT**	Ambiguïté du mécanisme causal documentée. Interprétation conservatrice privilégiée : parcimonie en contexte de données limitées.
Externe	Biais temporel (2007-2010)	**IMPORTANT**	Spécificité temporelle reconnue. Réplication sur corpus récents recommandée.
Externe	Spécificité paradigme OO	**IMPORTANT**	Applicabilité prioritaire aux langages OO. Métriques adaptées nécessaires.
Externe	Effectif limité (N=11)	**IMPORTANT**	Puissance 42% documentée. Détection des effets larges uniquement.
Externe	Contraintes computationnelles	**MODÉRÉ**	Reflète les ressources académiques standard. Résultats = frontière atteignable.
Conclusion	Puissance statistique 42%	**IMPORTANT**	Distinction "absence de preuve" ≠ "preuve d'absence" documentée.
Conclusion	Violation indépendance des observations	**MODÉRÉ**	Agrégation par projet atténue la menace. Résultats significatifs robustes.
Conclusion	Comparaisons multiples (n=9)	**MODÉRÉ**	Transparence privilégiée. Résultats significatifs robustes après correction Holm-Bonferroni.

Note. Impact : MAJEUR (limite portée substantiellement), IMPORTANT (circonscrit généralisation), MODÉRÉ (nuance interprétation).

6.3. Perspectives de recherche futures

Les limitations identifiées et les observations empiriques du Chapitre 5 orientent plusieurs directions de recherche futures, structurées selon trois horizons de faisabilité.

6.3.1. Enrichissement qualitatif des représentations

Le plateau de 60-66 % G-Mean indépendamment des sophistications architecturales suggère que les métriques CK capturent une vue partielle de l'information prédictive disponible. Pour le dépasser, il faudrait enrichir les représentations en exploitant d'autres dimensions.

R. Moser et al. [80] démontrent empiriquement que les métriques de processus présentent un pouvoir prédictif supérieur aux métriques structurelles pour la prédiction de défauts. Leurs expériences sur le projet Eclipse révèlent que parmi 18 métriques de changement testées, trois métriques (nombre de révisions, taille maximale des changements, et corrections de bugs antérieures) contiennent l'essentiel de l'information prédictive.

Une piste est de fusionner les métriques CK statiques avec des caractéristiques temporelles issues de l'historique Git (fréquence des commits, taille des changements, coévolution). L'hypothèse sous-jacente est que la combinaison structure complexe, évolution fréquente et multiples contributeurs fournirait un signal prédictif plus riche que la structure statique seule. Au-delà des métriques numériques agrégées, les graphes de dépendances entre classes et packages capturent la structure relationnelle du système. Cette information est potentiellement critique pour la prédiction.

La revue systématique de G. Giray et al. [18] identifie les Graph Neural Networks (GNN) comme une approche émergente dans le domaine. Les GNN sont conçues pour exploiter la structure et les propriétés des graphes en effectuant des inférences sur des données décrites par des graphes. Dans le contexte de la prédiction de défauts, les GNN permettent d'exploiter la structure arborescente du code source pour acquérir l'information latente de défauts dans les sous-arbres défectueux.

Une direction future consisterait à extraire des graphes multi-relationnels comprenant des arêtes de différents types. On pourrait ensuite appliquer des architectures GNN sophistiquées comme GraphSAGE ou GAT pour apprendre des représentations vectorielles de nœuds encodant le contexte structural. L'hypothèse est que des classes

structurellement similaires présentent des risques de défauts similaires. Cette information n'est pas capturée par les métriques locales CK.

Nos expérimentations avec les architectures multimodales intégrant le code source n'ont pas révélé d'amélioration substantielle. Cela provient possiblement des limitations du pré-entraînement générique. CodeBERT a été entraîné sur un corpus GitHub massif mais n'est pas spécialisé pour la prédiction de défauts.

Les modèles de langage actuels démontrent des capacités remarquables de compréhension sémantique du code. Des représentations vectorielles issues de ces modèles pourraient capturer des patterns de mauvaise qualité invisibles des métriques CK. Ces patterns incluent la logique métier incohérente, la gestion d'erreurs inadéquate et les violations de principes SOLID. L'approche technique consisterait en trois étapes. Premièrement, générer des représentations vectorielles de chaque classe via un modèle de langage de code. Deuxièmement, combiner ces représentations vectorielles avec les métriques CK via fusion multimodale. Troisièmement, entraîner un classifieur sur cette représentation hybride.

6.3.2 Approches contextuellement adaptatives

L'analyse ANOVA révèle que 76,6 % de la variance provient des différences inter-projets. Ce résultat suggère une opportunité stratégique importante. Plutôt que chercher la configuration universellement optimale, qui n'existe probablement pas, il faudrait développer des approches s'adaptant automatiquement aux caractéristiques du projet cible.

Méta-apprentissage pour la sélection de modèles. Le méta-apprentissage offre un cadre pour exploiter les patterns de performance inter-projets. L'approche se déroulerait en trois phases : caractériser chaque projet par un vecteur méta (statistiques CK, taux de défauts, taille, domaine), entraîner un méta-modèle prédisant la configuration optimale pour un nouveau projet, puis appliquer automatiquement cette recommandation. Cette approche transforme la variabilité inter-projets en opportunité. Plutôt que de l'ignorer, on l'exploite explicitement pour guider la sélection. Les travaux de C. Tantithamthavorn et al. [81] sur l'optimisation automatique des hyperparamètres démontrent que l'adaptation

contextuelle améliore significativement les performances par rapport aux configurations fixes.

Adaptation de domaine pour la transférabilité cross-project. Notre observation révèle que 44 % des transitions WPDP vers CPDP montrent des améliorations. Ce résultat suggère que le transfert inter-projets n'est pas uniformément problématique. Les techniques d'adaptation de domaine visent à réduire l'écart de distribution entre projets source et cible. Elles pourraient améliorer systématiquement la transférabilité. Les approches prometteuses incluent plusieurs stratégies. L'alignement des distributions de caractéristiques peut s'effectuer via adversarial training. La sélection intelligente de projets sources devrait cibler ceux similaires au projet cible. La pondération adaptative des instances devrait refléter leur pertinence au domaine cible.

6.3.3 Amélioration de la rigueur méthodologique du domaine

Au-delà des extensions empiriques, notre recherche identifie plusieurs opportunités d'amélioration des standards méthodologiques.

Protocoles standardisés et benchmarks communautaires. La fragmentation méthodologique complique substantiellement les comparaisons inter-études. Il est préconisé l'établissement de protocoles standardisés pour améliorer la reproductibilité et faciliter les comparaisons rigoureuses. Une direction future consisterait à établir un benchmark communautaire comprenant quatre composantes essentielles. Premièrement, un corpus de référence diversifié avec plus de 50 projets couvrant multiples langages et domaines. Deuxièmement, un protocole d'évaluation standardisé incluant validation temporelle, métriques fixes et tests statistiques obligatoires. Troisièmement, une plateforme de soumission centralisée avec code reproductible. Quatrièmement, un leaderboard public documentant clairement les configurations testées. Cette infrastructure transformerait la dynamique du domaine. Elle faciliterait l'identification robuste des avancées réelles versus les gains artificiels.

Études de réplication systématiques. Notre recherche documente explicitement l'absence de significativité généralisée. Cette documentation illustre l'importance de la

validation indépendante. Une proportion substantielle de la littérature rapporte des améliorations sans tests statistiques rigoureux. L'absence de documentation des puissances statistiques crée un biais de publication favorisant les résultats positifs. Une autre direction future consisterait à institutionnaliser les réplications selon quatre axes. Premièrement, créer des incitations académiques valorisant les études de réplication. Deuxièmement, adopter des protocoles pré-enregistrés spécifiant hypothèses et analyses avant expérimentation. Troisièmement, assurer le partage systématique de données et code avec packages de reproductibilité complète. Quatrièmement, conduire des méta-analyses périodiques synthétisant résultats multiples pour identifier effets robustes versus artefacts méthodologiques.

6.4. Conclusion générale

6.4.1. Rappel de la problématique et du contexte

Cette recherche est née d'une observation paradoxale. Malgré l'émergence des architectures d'apprentissage profond et leur succès dans des domaines connexes, leur application à la prédiction de défauts révèle une fragmentation méthodologique substantielle. Il existe également une absence de consensus sur les pratiques optimales.

Face à cette fragmentation, notre problématique s'est articulée autour d'une tension centrale. Comment évaluer systématiquement l'impact des choix méthodologiques multidimensionnels sur les performances ? Comment reconnaître explicitement les contraintes techniques réelles qui circonscrivent la portée des conclusions ?

Quatre questions de recherche ont structuré notre démarche. RQ1 examine l'impact des stratégies de prétraitement. RQ2 compare les architectures. RQ3 évalue la transférabilité WPDP-CPDP. RQ4 valide statistiquement les différences observées. Ces questions ont été formulées de manière ouverte. Elles reflètent notre posture épistémologique consistant à privilégier l'exploration rigoureuse sur la confirmation de présupposés et à documenter l'incertitude plutôt que la masquer.

6.4.2 Synthèse des contributions principales

Notre recherche apporte quatre contributions principales qui enrichissent collectivement la compréhension du domaine.

C1 — Validation du principe de parcimonie architecturale. Le CNN1D Simple surpasse statistiquement le Transformer Encodeur en CPDP et présente une robustesse 2,6 fois supérieure. Ce résultat valide, dans le contexte spécifique des données tabulaires hétérogènes, le principe selon lequel des modèles de complexité contrôlée généralisent mieux en présence de données limitées.

C2 — Documentation de l'instabilité du pré-entraînement massif. La variabilité extrême du Transformer entièrement pré-entraîné (σ = 14,11 %) constitue un avertissement pratique : les grands modèles pré-entraînés sur du texte ne transfèrent pas de façon fiable sur des métriques logicielles structurées, invitant à la prudence dans leur adoption industrielle.

C3 — Quantification de la dominance contextuelle. L'ANOVA décompose précisément les sources de variabilité : 76,6 % provient des caractéristiques intrinsèques des projets contre 6,7 % des choix méthodologiques. Cette quantification remet en question l'emphase implicite sur l'optimisation algorithmique et oriente les priorités de recherche vers la modélisation des facteurs contextuels.

C4 — Mise en évidence d'un plateau structurel. La convergence de plus de 75 % des configurations vers 60–66 % de G-Mean, indépendamment des sophistications testées, suggère un plafond lié aux limites informationnelles des métriques CK. Ce « résultat négatif » constitue une contribution positive : il oriente les efforts futurs vers l'enrichissement des représentations plutôt que la seule optimisation algorithmique.

6.4.3. Portée des conclusions et domaine d'applicabilité

L'analyse détaillée des menaces à la validité (section 6.2) circonscrit précisément le domaine d'applicabilité légitime de nos conclusions. Nos résultats s'appliquent rigoureusement à un contexte spécifique défini par quatre caractéristiques.

Premièrement, les projets étudiés sont des projets Java open-source matures de type bibliothèque ou framework. Deuxièmement, nous utilisons les métriques CK traditionnelles. Troisièmement, le corpus provient de PROMISE historique (2007-2010). Quatrièmement, l'infrastructure utilisée est académique standard.

La généralisation à d'autres langages, domaines, organisations ou représentations nécessiterait une validation indépendante. Bien loin de diminuer la valeur de notre travail, cette circonscription est une contribution : elle délimite précisément ce que nos résultats permettent d'affirmer, et ce qu'ils ne permettent pas.

Trois observations émergent avec confiance élevée. Premièrement, le plateau de 60-66 % apparaît robuste. Deuxièmement, les architectures simples égalent ou surpassent les Transformers complexes dans ce contexte. Troisièmement, la variabilité inter-projets domine massivement les différences méthodologiques. Ces observations sont limitées en portée. Elles fournissent néanmoins des insights précieux orientant les recherches futures.

Au terme de ce parcours, nous réalisons que le processus d'apprentissage s'est avéré aussi enrichissant que les résultats eux-mêmes. En débutant ce mémoire, nous portions des attentes typiques. Nous souhaitions identifier la « meilleure » configuration, démontrer la supériorité d'une approche sophistiquée et apporter une contribution immédiatement applicable. La confrontation avec la réalité empirique a progressivement transformé notre compréhension. Le plateau de performance, l'absence de différences significatives et la dominance écrasante des caractéristiques des projets ont redéfini ce qui constitue une contribution valable.

Nous avons appris que documenter rigoureusement ce qui ne fonctionne pas constitue une contribution aussi précieuse que célébrer des succès marginaux potentiellement non-reproductibles. L'honnêteté intellectuelle renforce la crédibilité scientifique plutôt qu'elle ne l'affaiblit. La reconnaissance explicite des frontières de généralisation guide plus efficacement les recherches futures que des affirmations optimistes insuffisamment circonscrites.

Ce parcours nous a fait prendre conscience de la complexité du domaine. Les logiciels sont des artefacts sociotechniques, reflets de décisions et de compromis humains, et non des objets régis par des lois immuables. Cette contingence rend la quête d'une solution universelle probablement vaine, et justifie de privilégier des approches adaptatives.

L'enseignement principal pourrait se condenser ainsi. La maturité scientifique d'un domaine se mesure autant à la précision avec laquelle il reconnaît ses limites qu'à l'ampleur de ses succès. En documentant explicitement les limitations et incertitudes de nos résultats, nous espérons contribuer à une culture de transparence méthodologique dans un domaine où la reproductibilité demeure un défi reconnu.

Nos résultats suggèrent une orientation pratique. Nos quatre résultats convergent vers une recommandation commune : dans le contexte de données tabulaires limitées, la priorité devrait aller à l'enrichissement des représentations et à l'adaptation contextuelle, plutôt qu'à l'optimisation fine de configurations dont l'impact est masqué par la variabilité inter-projets.

Références bibliographiques

[1] Microsoft. (2024, 19 juillet). CrowdStrike issue impacting Windows endpoints causing Azure Virtual Machines and services to be in a non-bootable state [Billet de blogue]. https://techcommunity.microsoft.com/t5/azure-infrastructure-blog/crowdstrike-issue-impacting-windows-endpoints-causing-azure/ba-p/4196140

[2] Krasner, H. (2022). The cost of poor software quality in the US: A 2022 report. Consortium for Information & Software Quality (CISQ). https://www.it-cisq.org/the-cost-of-poor-software-quality-in-the-us-a-2022-report/

[3] Zhang, Y., Lo, D., Xia, X., Xu, B., Sun, J., & Li, S. (2019). Characterizing and identifying misclassified software bug reports. IEEE Transactions on Software Engineering, 47(1), 86-103. https://doi.org/10.1109/TSE.2019.2892959

[4] Alsaeedi, A., & Khan, M. Z. (2018). Software defect prediction using supervised machine learning and ensemble methods: A comparative study. Journal of Software Engineering and Applications, 12(5), 85-100. https://doi.org/10.4236/jsea.2019.125007

[5] Lessmann, S., Baesens, B., Mues, C., & Pietsch, S. (2008). Benchmarking classification models for software defect prediction: A proposed framework and

novel findings. IEEE Transactions on Software Engineering, 34(4), 485-496. https://doi.org/10.1109/TSE.2008.35

[6] Kamei, Y., Shihab, E., Adams, B., Hassan, A. E., Mockus, A., Sinha, A., & Ubayashi, N. (2013). A large-scale empirical study of just-in-time quality assurance. IEEE Transactions on Software Engineering, 39(6), 757-773. https://doi.org/10.1109/TSE.2012.70

[7] Pan, C., Lu, M., & Xu, B. (2021). An empirical study on software defect prediction using CodeBERT model. Applied Sciences, 11(11), 4793. https://doi.org/10.3390/app11114793

[8] Sahar, S., Younas, M., Khan, M. M., & Sarwar, M. U. (2024). DP-CCL: A supervised contrastive learning approach using CodeBERT model in software defect prediction. IEEE Access, 12, 22582-22594. https://doi.org/10.1109/ACCESS.2024.3362896

[9] He, H., & Garcia, E. A. (2009). Learning from imbalanced data. IEEE Transactions on Knowledge and Data Engineering, 21(9), 1263-1284. https://doi.org/10.1109/TKDE.2008.239

[10] Feng, S., Keung, J., Yu, X., Xiao, Y., & Zhang, M. (2021). Investigation on the stability of SMOTE-based oversampling techniques in software defect prediction. Information and Software Technology, 139, Article 106662. https://doi.org/10.1016/j.infsof.2021.106662

[11] Hosseini, S., Turhan, B., & Gunarathna, D. (2019). A systematic literature review and meta-analysis on cross project defect prediction. IEEE Transactions on Software Engineering, 45(2), 111-147. https://doi.org/10.1109/TSE.2017.2770124

[12] McIntosh, S., & Kamei, Y. (2018). Are fix-inducing changes a moving target? A longitudinal case study of just-in-time defect prediction. IEEE Transactions on Software Engineering, 44(5), 412-428. https://doi.org/10.1109/TSE.2017.2693980

[13] Zimmermann, T., Nagappan, N., Gall, H., Giger, E., & Murphy, B. (2009). Cross-project defect prediction: A large-scale experiment on data vs. domain vs. process. Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), 91-100. https://doi.org/10.1145/1595696.1595713

[14] Lin, D., Tantithamthavorn, C., & Hassan, A. E. (2022). The impact of data merging on the interpretation of cross-project just-in-time defect models. IEEE Transactions on Software Engineering, 48(8), 2969-2986. https://doi.org/10.1109/TSE.2021.3073920

[15] Tantithamthavorn, C., Hassan, A. E., & Matsumoto, K. (2018). The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. IEEE Transactions on Software Engineering, 46(11), 1200-1219. https://doi.org/10.1109/TSE.2018.2876537

[16] Vescan, A., Găceanu, R., & Şerban, C. (2024). Exploring the impact of data preprocessing techniques on composite classifier algorithms in cross-project defect

prediction. Automated Software Engineering, 31, 47. https://doi.org/10.1007/s10515-024-00454-9

[17] Pachouly, J., Ahirrao, S., Kotecha, K., Selvachandran, G., & Abraham, A. (2022). A systematic literature review on software defect prediction using artificial intelligence: Datasets, data validation methods, approaches, and tools. Engineering Applications of Artificial Intelligence, 111, 104773. https://doi.org/10.1016/j.engappai.2022.104773

[18] 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, Article 111537. https://doi.org/10.1016/j.jss.2022.111537

[19] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research, 7, 1-30.

[20] Heil, M. M., Mattson, C. A., Erickson, L. M., Christensen, A. A., Yeh, C. I., & Leishman, D. J. (2021). Reproducibility standards for machine learning in the life sciences. Nature Methods, 18(10), 1132-1135. https://doi.org/10.1038/s41592-021-01256-7

[21] Dodge, J., Gururangan, S., Card, D., Schwartz, R., & Smith, N. A. (2019). Show your work: Improved reporting of experimental results. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, 2185-2194. https://doi.org/10.18653/v1/D19-1224

[22] Kapoor, S., Cantrell, E. M., Peng, K., Pham, T. H., Bail, C. A., Gundersen, O. E., Hofman, J. M., Hullman, J., Lones, M. A., Malik, M. M., Nanayakkara, P., Poldrack, R. A., Raji, I. D., Roberts, M., Salganik, M. J., Serra-Garcia, M., Stewart, B. M., Vandewiele, G., & Narayanan, A. (2024). REFORMS: Consensus-based recommendations for machine-learning-based science. Science Advances, 10(18), eadk3452. https://doi.org/10.1126/sciadv.adk3452

[23] Semmelrock, H., Ross-Hellauer, T., Kopeinik, S., Theiler, D., Haberl, A., Thalmann, S., & Kowald, D. (2025). Reproducibility in machine learning-based research: Overview, barriers, and drivers. AI Magazine, 46, e70002. https://doi.org/10.1002/aaai.70002

[24] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). Experimentation in software engineering. Springer Science & Business Media. https://doi.org/10.1007/978-3-642-29044-2

[25] Avizienis, A., Laprie, J.-C., Randell, B., & Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing, 1(1), 11-33. https://doi.org/10.1109/TDSC.2004.2

[26] Herzig, K., Just, S., & Zeller, A. (2013). It's not a bug, it's a feature: How misclassification impacts bug prediction. Proceedings of the 35th International Conference on Software Engineering (ICSE), 392-401. https://doi.org/10.1109/ICSE.2013.6606585

[27] Fenton, N. E., & Bieman, J. M. (2014). Software metrics: A rigorous and practical approach (3e éd.). CRC Press.

[28] Zhang, H. (2009). An investigation of the relationships between lines of code and defects. Proceedings of the 2009 IEEE International Conference on Software Maintenance (ICSM), 274-283. https://doi.org/10.1109/ICSM.2009.5306338

[29] McCabe, T. J. (1976). A complexity measure. IEEE Transactions on Software Engineering, SE-2(4), 308-320. https://doi.org/10.1109/TSE.1976.233837

[30] Watson, A. H., & McCabe, T. J. (1996). Structured testing: A testing methodology using the cyclomatic complexity metric (NIST Special Publication 500-235). National Institute of Standards and Technology.

[31] Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. IEEE Transactions on Software Engineering, 20(6), 476-493. https://doi.org/10.1109/32.295895

[32] Basili, V. R., Briand, L. C., & Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering, 22(10), 751-761. https://doi.org/10.1109/32.544352

[33] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 16, 321-357. https://doi.org/10.1613/jair.953

[34] Balogun, A. O., Odejide, B. J., Bajeh, A. O., Alanamu, Z. O., Usman-Hamza, F. E., Adeleke, H. O., Mabayoje, M. A., & Yusuff, S. R. (2022). Empirical analysis of data sampling-based ensemble methods in software defect prediction. Dans O. Gervasi, B. Murgante, S. Misra, A. M. A. C. Rocha, & C. Garau (dir.), *Computational Science and Its Applications – ICCSA 2022 Workshops* (pp. 367–384). Springer. https://doi.org/10.1007/978-3-031-10548-7_27

[35] Li, J., He, P., Zhu, J., & Lyu, M. R. (2017). Software defect prediction via convolutional neural network. Dans 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS) (p. 318-328). IEEE. https://doi.org/10.1109/QRS.2017.42

[36] Pan, C., Lu, M., Xu, B., & Gao, H. (2019). An improved CNN model for within-project software defect prediction. Applied Sciences, 9(10), 2138. https://doi.org/10.3390/app9102138

[37] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. Dans I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (dir.), Advances in Neural Information Processing Systems (Vol. 30, p. 5998-6008). Curran Associates, Inc.

[38] Tantithamthavorn, C., McIntosh, S., Hassan, A. E., & Matsumoto, K. (2017). An empirical comparison of model validation techniques for defect prediction models.

IEEE Transactions on Software Engineering, 43(1), 1-18. https://doi.org/10.1109/TSE.2016.2584050

[39] Chicco, D., & Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. BMC Genomics, 21(1), 6. https://doi.org/10.1186/s12864-019-6413-7

[40] Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. Proceedings of the IEEE, 68(9), 1060-1076. https://doi.org/10.1109/PROC.1980.11805

[41] Herraiz, I., Rodriguez, D., Robles, G., & Gonzalez-Barahona, J. M. (2013). The evolution of the laws of software evolution: A discussion based on a systematic literature review. ACM Computing Surveys, 46(2), Article 28. https://doi.org/10.1145/2543581.2543595

[42] Mahmood, Z., Bowes, D., Hall, T., Lane, P. C. R., & Petrić, J. (2018). Reproducibility and replicability of software defect prediction studies. Information and Software Technology, 99, 148-163. https://doi.org/10.1016/j.infsof.2017.10.009

[43] Jiarpakdee, J., Tantithamthavorn, C., & Hassan, A. E. (2021). The impact of correlated metrics on the interpretation of defect models. IEEE Transactions on Software Engineering, 47(2), 320-331. https://doi.org/10.1109/TSE.2019.2891758

[44] Wang, S., Liu, T., & Tan, L. (2016). Automatically learning semantic features for defect prediction. Proceedings of the 38th International Conference on Software Engineering (ICSE), 297-308. https://doi.org/10.1145/2884781.2884804

[45] Farid, A. B., Fathy, E. M., Eldin, A. S., & Abd-Elmegid, L. A. (2021). Software defect prediction using hybrid model (CBIL) of convolutional neural network (CNN) and bidirectional long short-term memory (Bi-LSTM). PeerJ Computer Science, 7, e739. https://doi.org/10.7717/peerj-cs.739

[46] Shepperd, M., Bowes, D., & Hall, T. (2014). Researcher bias: The use of machine learning in software defect prediction. IEEE Transactions on Software Engineering, 40(6), 603–616. https://doi.org/10.1109/TSE.2014.2322358

[47] Bhat, N. A., & Farooq, S. U. (2023). An empirical evaluation of defect prediction approaches in within-project and cross-project context. Software Quality Journal, 31, 917-946. https://doi.org/10.1007/s11219-023-09615-7

[48] Wongpheng, K., & Visutsak, P. (2020). Software defect prediction using convolutional neural network. Dans 2020 35th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC) (p. 240-243). IEEE. https://doi.org/10.1109/ITC-CSCC46805.2020.9182919

[49] Qiu, S., Xu, H., Deng, J., Jiang, S., & Lu, L. (2019). Transfer CNN for cross-project defect prediction. Applied Sciences, 9(13), 2660. https://doi.org/10.3390/app9132660

[50] Deng, J., Lu, L., Qiu, S., & Ou, Y. (2020). A suitable AST node granularity and multi-kernel transfer CNN for cross-project defect prediction. IEEE Access, 8, 66647-66661. https://doi.org/10.1109/ACCESS.2020.2985780

[51] Zhang, Q., & Wu, B. (2020). Software defect prediction via transformer. ITNEC, 1, 874-879. https://doi.org/10.1109/ITNEC48623.2020.9084745

[52] Ahmad, W., Chakraborty, S., Ray, B., & Chang, K. W. (2021). Unified pre-training for program understanding and generation. Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT), 2655-2668. https://doi.org/10.18653/v1/2021.naacl-main.211

[53] Song, L., & Minku, L. L. (2023). A procedure to continuously evaluate predictive performance of just-in-time software defect prediction models during software development. IEEE Transactions on Software Engineering, 49(2), 646-666. https://doi.org/10.1109/TSE.2022.3158831

[54] Uddin, M. N., Li, B., Ali, Z., Kefalas, P., Khan, I., & Zada, I. (2022). Software defect prediction employing BiLSTM and BERT-based semantic feature. Soft Computing, 26(16), 7877-7891. https://doi.org/10.1007/s00500-022-06830-5

[55] Liang, H., Yu, Y., Jiang, L., & Xie, Z. (2019). Seml: A semantic LSTM model for software defect prediction. IEEE Access, 7, 83812-83824. https://doi.org/10.1109/ACCESS.2019.2925313

[56] Yao, J., & Shepperd, M. (2020). Assessing software defection prediction performance: Why using the Matthews correlation coefficient matters. Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering (EASE '20), 120-129. https://doi.org/10.1145/3383219.3383232

[57] Rakha, M. S., Miranskyy, A., & Alencar da Costa, D. (2025). Contrasting the hyperparameter tuning impact across software defect prediction scenarios. IEEE Transactions on Software Engineering. https://doi.org/10.1109/TSE.2025.3624631

[58] Herbold, S., Trautsch, A., & Grabowski, J. (2018). A comparative study to benchmark cross-project defect prediction approaches. IEEE Transactions on Software Engineering, 44(9), 811-833. https://doi.org/10.1109/TSE.2017.2724538

[59] Jureczko, M., & Madeyski, L. (2010). Towards identifying software project clusters with regard to defect prediction. Dans Proceedings of the 6th International Conference on Predictive Models in Software Engineering (p. 9:1-9:10). ACM. https://doi.org/10.1145/1868328.1868342

[60] Abdu, A., Zhai, Z., Abdo, H. A., Algabri, R., Al-masni, M. A., Muhammad, M. S., & Gu, Y. H. (2024). Semantic and traditional feature fusion for software defect prediction using hybrid deep learning model. Scientific Reports, 14(1), 14771. https://doi.org/10.1038/s41598-024-65639-4

[61] Nam, J., Pan, S. J., & Kim, S. (2013). Transfer defect learning. Dans Proceedings of the 2013 35th International Conference on Software Engineering (p. 382-391). IEEE. https://doi.org/10.1109/ICSE.2013.6606584

[62] Rousseeuw, P. J., & Van Driessen, K. (1999). A fast algorithm for the minimum covariance determinant estimator. Technometrics, 41(3), 212-223. https://doi.org/10.1080/00401706.1999.10485670

[63] Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). Isolation forest. Dans Proceedings of the 8th IEEE International Conference on Data Mining (p. 413-422). IEEE. https://doi.org/10.1109/ICDM.2008.17

[64] Zain, Z. M., Sakri, S., Asmak Ismail, N. H., & Parizi, R. M. (2022). Software defect prediction harnessing on multi 1-dimensional convolutional neural network structure. Computers, Materials & Continua, 71(1), 1521–1546. https://doi.org/10.32604/cmc.2022.022085

[65] Ekanayake, J., Tappolet, J., Gall, H. C., & Bernstein, A. (2012). Time variance and defect prediction in software projects. Empirical Software Engineering, 17(4-5), 348-389. https://doi.org/10.1007/s10664-011-9180-x

[66] Kubat, M., & Matwin, S. (1997). Addressing the curse of imbalanced training sets: One-sided selection. Proceedings of the 14th International Conference on Machine Learning, 179-186.

[67] Akimova, E. N., Bersenev, A. Y., Deikov, A. A., Kobylkin, K. S., Konygin, A. V., Mezentsev, I. P., & Misilov, V. E. (2021). A survey on software defect prediction using deep learning. Mathematics, 9(11), 1180. https://doi.org/10.3390/math9111180

[68] Grinsztajn, L., Oyallon, E., & Varoquaux, G. (2022). Why do tree-based models still outperform deep learning on typical tabular data? In Advances in Neural Information Processing Systems (NeurIPS 2022), 35, 507-520. https://arxiv.org/abs/2207.08815

[69] Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the 32nd International Conference on Machine Learning (ICML), 37, 448-456.

[70] Buda, M., Maki, A., & Mazurowski, M. A. (2018). A systematic study of the class imbalance problem in convolutional neural networks. Neural Networks, 106, 249-259. https://doi.org/10.1016/j.neunet.2018.07.011

[71] Ali, M., Mazhar, T., Al-Rasheed, A., Shahzad, T., Ghadi, Y. Y., & Khan, M. A. (2024). Enhancing software defect prediction: A framework with improved feature selection and ensemble machine learning. PeerJ Computer Science, 10, e1860. https://doi.org/10.7717/peerj-cs.1860

[72] Nasser, A., Ghanem, W., Saad, A., Abdul-Qawy, A., Ghaleb, S., Alduais, N., Din, F., & Ghetas, M. (2024). Depth linear discrimination-oriented feature selection method based on adaptive sine cosine algorithm for software defect prediction. Expert

Systems with Applications, 253, Article 124266. https://doi.org/10.1016/j.eswa.2024.124266

[73] Vapnik, V. N. (2000). The nature of statistical learning theory (2e éd.). Springer. https://doi.org/10.1007/978-1-4757-3264-1

[74] Sudhakaran, A., Ali, A., McClean, S., Abu-Tair, M., & Hussain, T. (2025). Dynamics of software defect prediction: A systematic literature review on within-project and cross-project approaches. SSRN. https://doi.org/10.2139/ssrn.5354581

[75] Costa, D. A., McIntosh, S., Kulesza, U., Hassan, A. E., & Abdalkareem, R. (2023). An empirical study of software metrics volatility. Empirical Software Engineering, 28(2), Article 35. doi.org

[76] Javed, K., Shengbing, R., Asim, M., & Wani, M. A. (2024). Cross-project defect prediction based on domain adaptation and LSTM optimization. Algorithms, 17(5), 175. https://doi.org/10.3390/a17050175

[77] Cohen, J. (1988). Statistical power analysis for the behavioral sciences (2e éd.). Lawrence Erlbaum Associates.

[78] Kruschke, J. K. (2015). Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan (2e éd.). Academic Press.

[79] Mosbach, M., Andriushchenko, M., & Klakow, D. (2021). On the stability of fine-tuning BERT: Misconceptions, explanations, and strong baselines. Dans Proceedings of the International Conference on Learning Representations (ICLR). https://openreview.net/forum?id=nzpLWnVAyah

[80] Moser, R., Pedrycz, W., & Succi, G. (2008). A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. Proceedings of the 30th International Conference on Software Engineering, 181-190. https://doi.org/10.1145/1368088.1368114

[81] Tantithamthavorn, C., McIntosh, S., Hassan, A. E., & Matsumoto, K. (2019). The impact of automated parameter optimization on defect prediction models. IEEE Transactions on Software Engineering, 45(7), 683-711. https://doi.org/10.1109/TSE.2018.2794977

Annexes

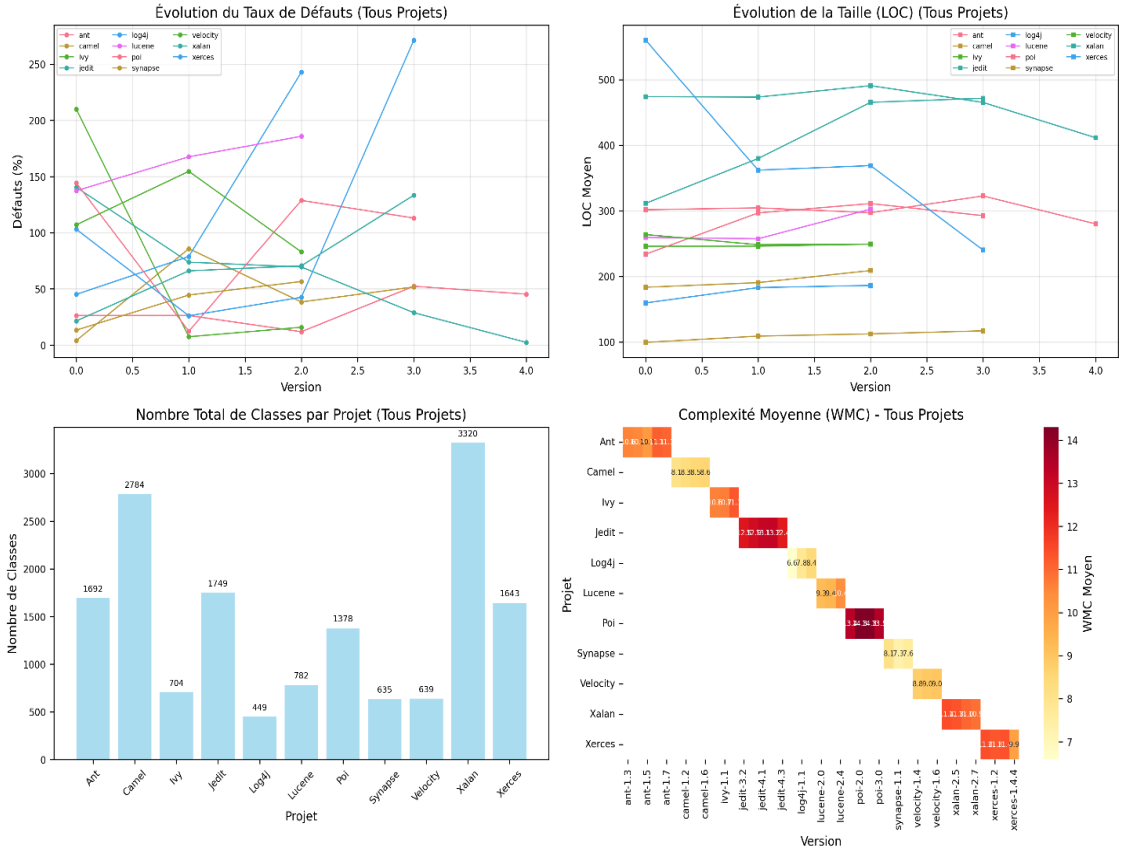
Annexe 1 : Statistiques descriptives

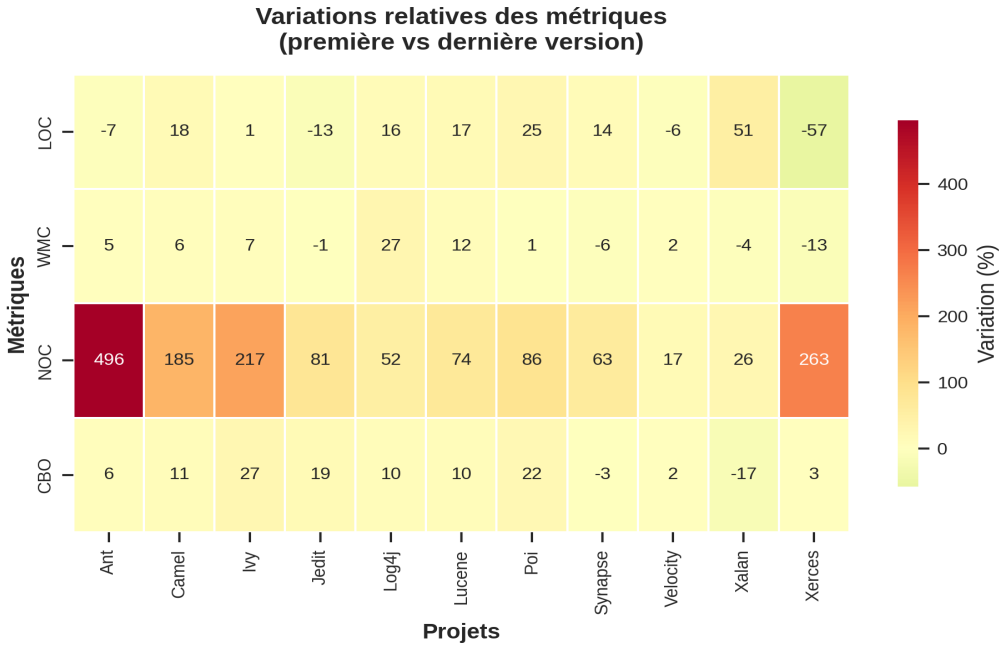
Projet	Version	Classes	Défauts	Taux Déf.	LOC Moy	WMC Moy	CBO Moy
Ant	ant-1.3	125	33	26.4	302	10.6	10.4
Ant	ant-1.4	178	47	26.4	304	10.5	10.8
Ant	ant-1.5	293	35	11.9	297	10.1	10.6
Ant	ant-1.6	351	184	52.4	323	11.1	11.5
Ant	ant-1.7	745	338	45.4	280	11.1	11.0
Camel	camel-1.0	339	14	4.1	99	8.1	10.0
Camel	camel-1.2	608	522	85.9	109	8.3	10.1
Camel	camel-1.4	872	335	38.4	112	8.5	10.8
Camel	camel-1.6	965	500	51.8	117	8.6	11.1
Ivy	ivy-1.0	111	233	209.9	246	10.6	10.4
Ivy	ivy-1.1	241	18	7.5	246	10.7	11.3
Ivy	ivy-1.2	352	56	15.9	249	11.3	13.2
Jedit	jedit-3.2	272	382	140.4	474	12.5	12.0
Jedit	jedit-4.0	306	226	73.9	473	12.9	12.4
Jedit	jedit-4.1	312	217	69.6	491	13.1	13.0
Jedit	jedit-4.2	367	106	28.9	465	13.2	14.1
Jedit	jedit-4.3	492	12	2.4	411	12.4	14.3
Log4j	log4j-1.0	135	61	45.2	160	6.6	6.9
Log4j	log4j-1.1	109	86	78.9	183	7.8	6.9
Log4j	log4j-1.2	205	498	242.9	186	8.4	7.6
Lucene	lucene-2.0	195	268	137.4	259	9.3	9.8
Lucene	lucene-2.2	247	414	167.6	257	9.4	10.0
Lucene	lucene-2.4	340	632	185.9	303	10.4	10.8
Poi	poi-1.5	237	342	144.3	234	13.4	8.3
Poi	poi-2.0	314	39	12.4	297	14.3	8.7
Poi	poi-2.5	385	496	128.8	311	14.3	9.0
Poi	poi-3.0	442	500	113.1	293	13.5	10.1
Synapse	synapse-1.0	157	21	13.4	183	8.1	13.2
Synapse	synapse-1.1	222	99	44.6	191	7.3	12.5
Synapse	synapse-1.2	256	145	56.6	209	7.6	12.8
Velocity	velocity-1.4	196	210	107.1	264	8.8	10.6
Velocity	velocity-1.5	214	331	154.7	248	9.0	10.6
Velocity	velocity-1.6	229	190	83.0	249	9.0	10.8
Xalan	xalan-2.4	723	156	21.6	311	11.4	14.5
Xalan	xalan-2.5	803	531	66.1	380	11.3	12.9
Xalan	xalan-2.6	885	625	70.6	465	11.0	12.1
Xalan	xalan-2.7	909	1213	133.4	471	10.9	12.0

Projet	Version	Classes	Défauts	Taux Déf.	LOC Moy	WMC Moy	CBO Moy
Xerces	xerces-1.1	162	167	103.1	560	11.4	6.1
Xerces	xerces-1.2	440	115	26.1	362	11.3	4.9
Xerces	xerces-1.3	453	193	42.6	369	11.4	5.1
Xerces	xerces-1.4.4	588	1596	271.4	240	9.9	6.3

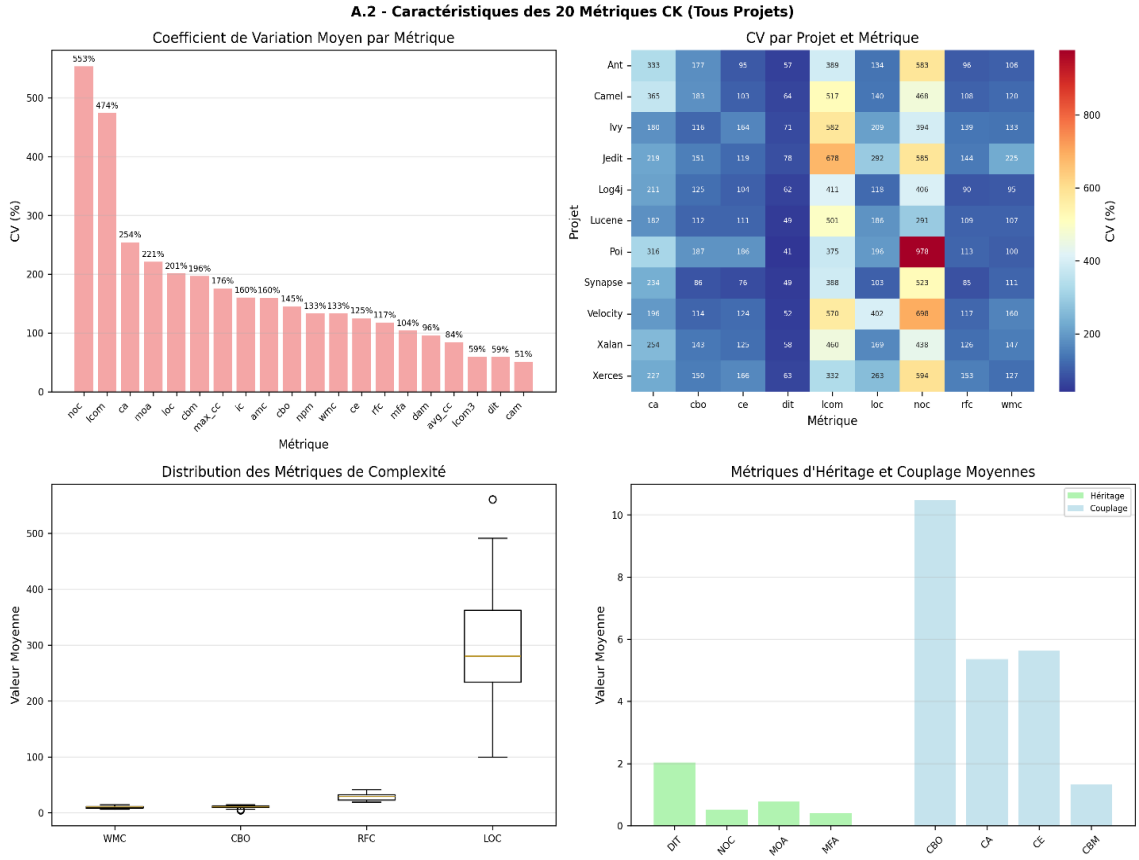
Métrique	Moyenne	Écart-type	CV (%)	Min	Max
	mean	***std***	***mean***	***mean***	***mean***
amc	25.713	13.228	159.875	0.0	532.239
avg_cc	1.363	0.296	84.23	0.008	12.491
ca	5.36	1.417	254.035	0.0	148.561
cam	0.468	0.031	51.226	0.008	1.0
cbm	1.321	0.806	196.469	0.0	15.659
cbo	10.468	2.431	145.17	0.146	159.756
ce	5.629	1.596	125.048	0.0	57.537
dam	0.503	0.148	96.22	0.0	1.0
dit	2.037	0.39	59.249	0.902	6.073
ic	0.508	0.188	160.165	0.0	3.512
lcom	98.797	60.839	473.938	0.0	7645.829
lcom3	1.081	0.145	59.394	0.0	2.0
loc	292.303	114.688	201.199	0.39	6735.171
max_cc	4.33	1.489	175.851	0.024	94.585
mfa	0.393	0.1	104.29	0.0	1.0
moa	0.773	0.235	221.398	0.0	15.195
noc	0.506	0.142	553.309	0.0	40.585
npm	7.948	1.728	133.196	0.024	97.146
rfc	28.411	6.147	117.465	0.268	299.146
wmc	10.472	1.993	133.069	0.268	142.341

A.1 - Caractéristiques du Corpus PROMISE (11 Projets)

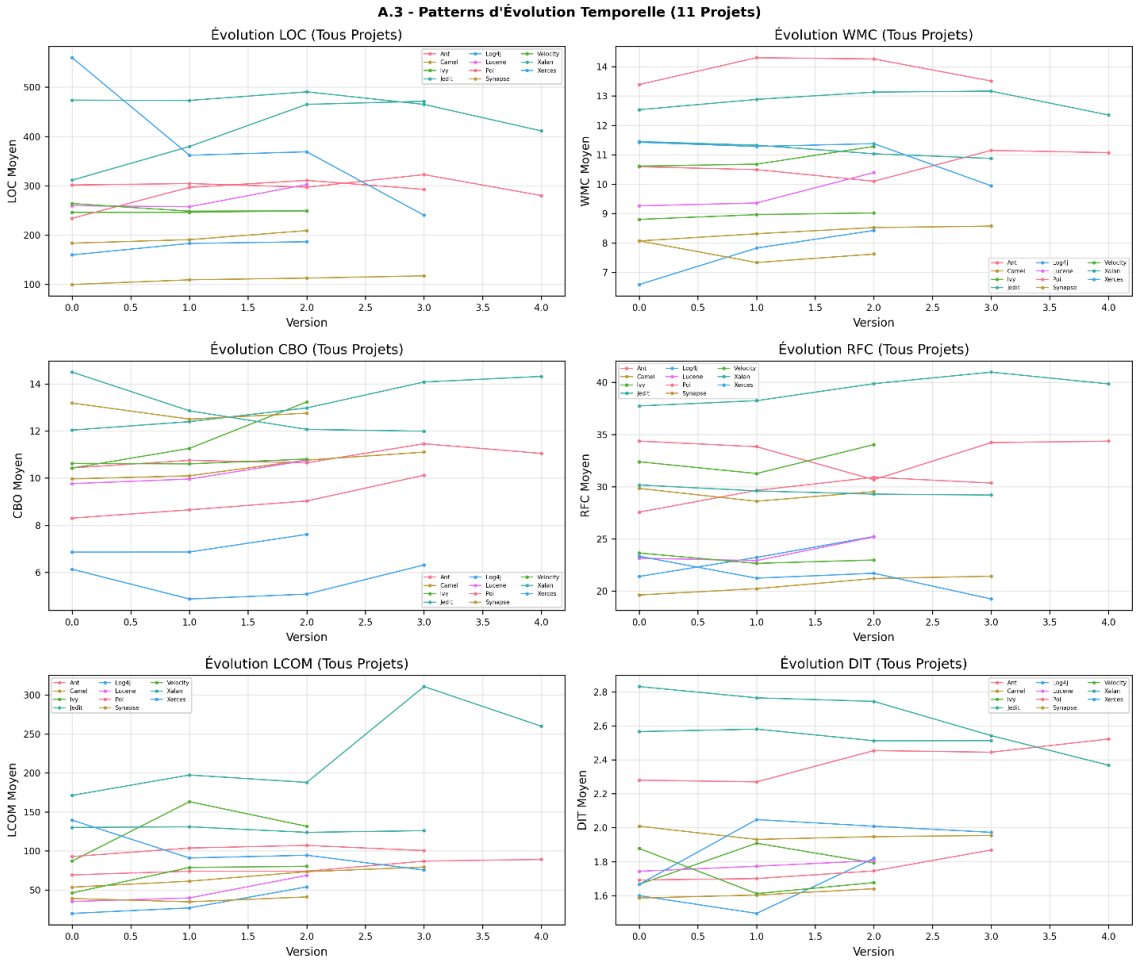




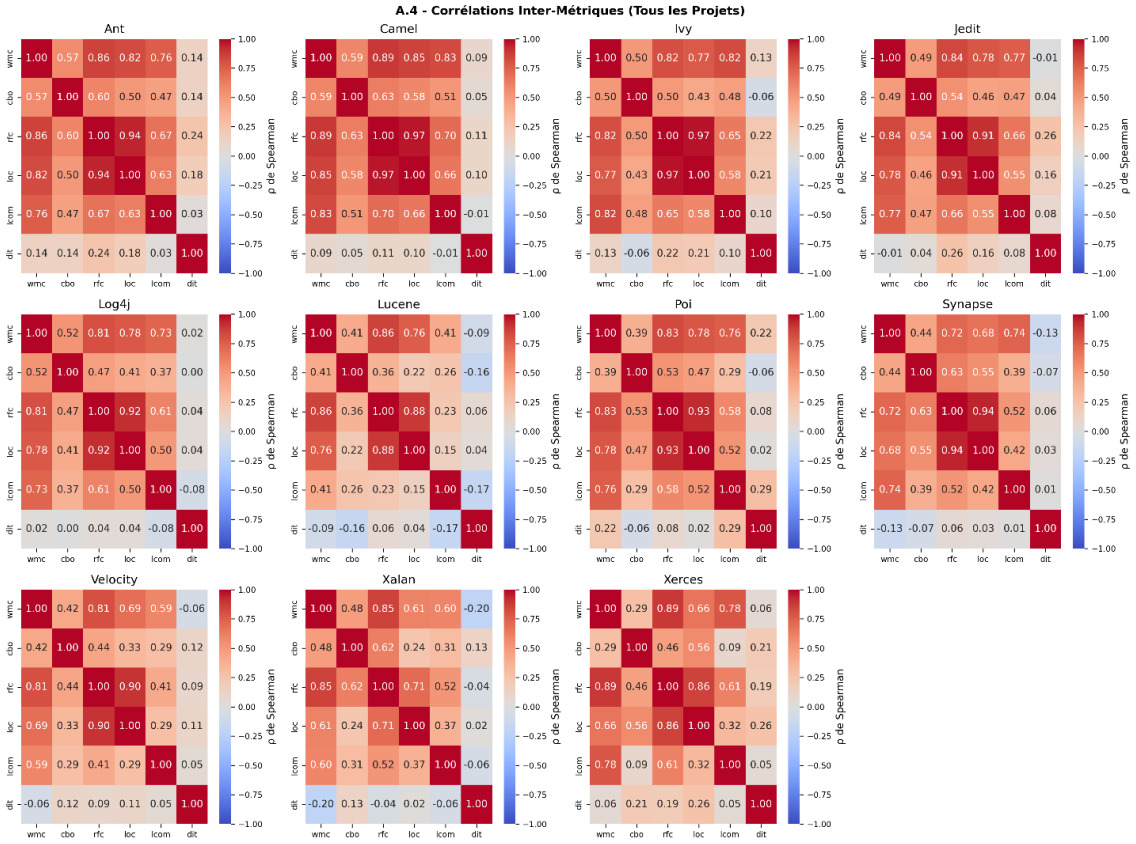
Annexe 2 :


A.2 - Caractéristiques des 20 Métriques CK (Tous Projets)
Coefficient de Variation Moyen par Métrique
CV (%)
Métrique
noc
lcom
ca
moa
loc
cbm
max_cc
ic
amc
cbo
npm
wmc
ce
rfc
mfa
dam
avg_cc
lcom3
dit
cam
CV par Projet et Métrique
Projet
Ant
Camel
Ivy
Jedit
Log4j
Lucene
Poi
Synapse
Velocity
Xalan
Xerces
Distribution des Métriques de Complexité
Valeur Moyenne
WMC
CBO
RFC
LOC
Métriques d'Héritage et Couplage Moyennes
Héritage
Couplage
DIT
NOC
MOA
MFA
CBO
CA
CE
CBM

Annexe 3 :


A.3 - Patterns d'Évolution Temporelle (11 Projets)
Évolution LOC (Tous Projets)
LOC Moyen
Version
Évolution WMC (Tous Projets)
WMC Moyen
Version
Évolution CBO (Tous Projets)
CBO Moyen
Version
Évolution RFC (Tous Projets)
RFC Moyen
Version
Évolution LCOM (Tous Projets)
LCOM Moyen
Version
Évolution DIT (Tous Projets)
DIT Moyen
Version
Ant
Camel
Ivy
JedIt
Log4j
Lucene
Poi
Synapse
Velocity
Xalan
Xerces

Annexe 4 :


A.4 - Corrélations Inter-Métriques (Tous les Projets)
Ant
Camel
Ivy
Jedit
Log4j
Lucene
Poi
Synapse
Velocity
Xalan
Xerces
wmc
cbo
rfc
loc
lcom
dit
ρ de Spearman