Articles

Publicaton source	Title	Year
FSE/ESEC	Improving IR-Based Bug Localization with Context-Aware Query Reformulation	2018
SANER	Combining Query Reduction and Expansion for Text-Retrieval-Based Bug Localization	2021
TOSEM	Precise Learn-to-Rank Fault Localization Using Dynamic and Static Features of Target Program	2019
ICSME	Using Observed Behavior to Reformulate Queries during Text Retrieval-based Bug Localization	2017
TSE	Deep Transfer Bug Localization	2021

BLIZZARD

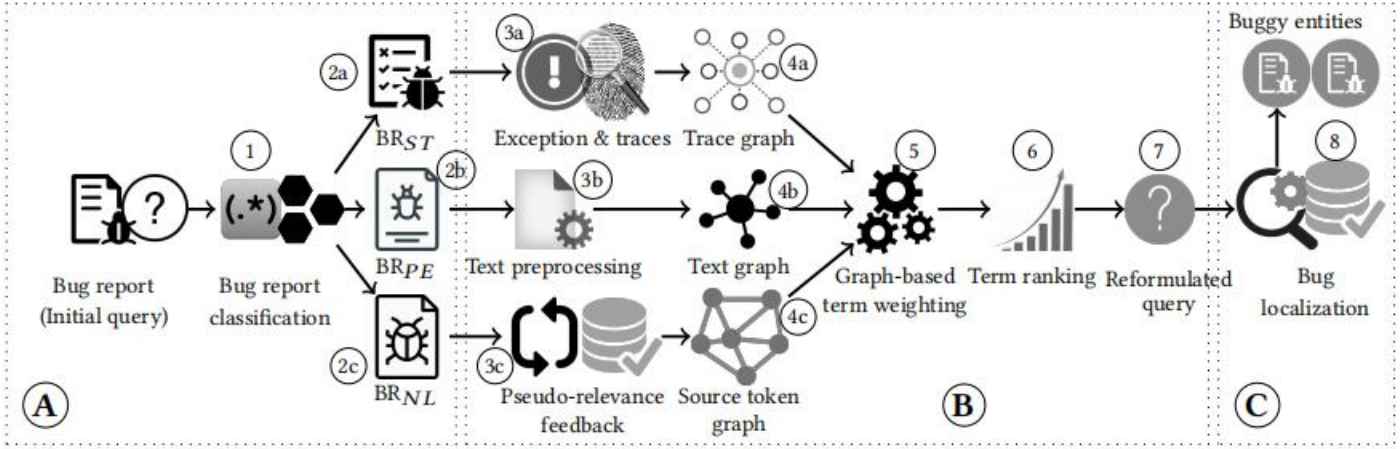
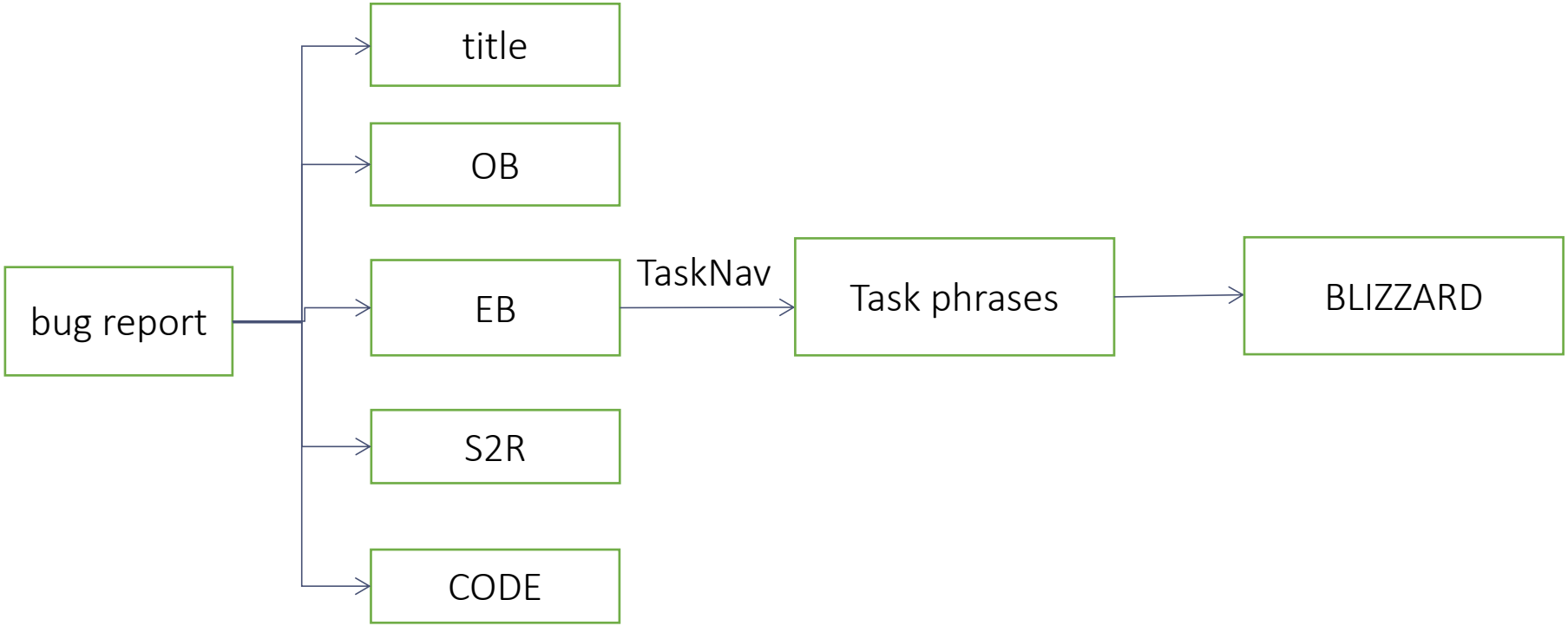


Figure 1: Schematic diagram of the proposed technique: (A) Bug report classification, (B) Query reformulation, and (C) Bug localization

Combining Query Reduction and Expansion for Text-Retrieval-Based Bug Localization



Precise Learn-to-Rank Fault Localization Using Dynamic and Static Features of Target Programs

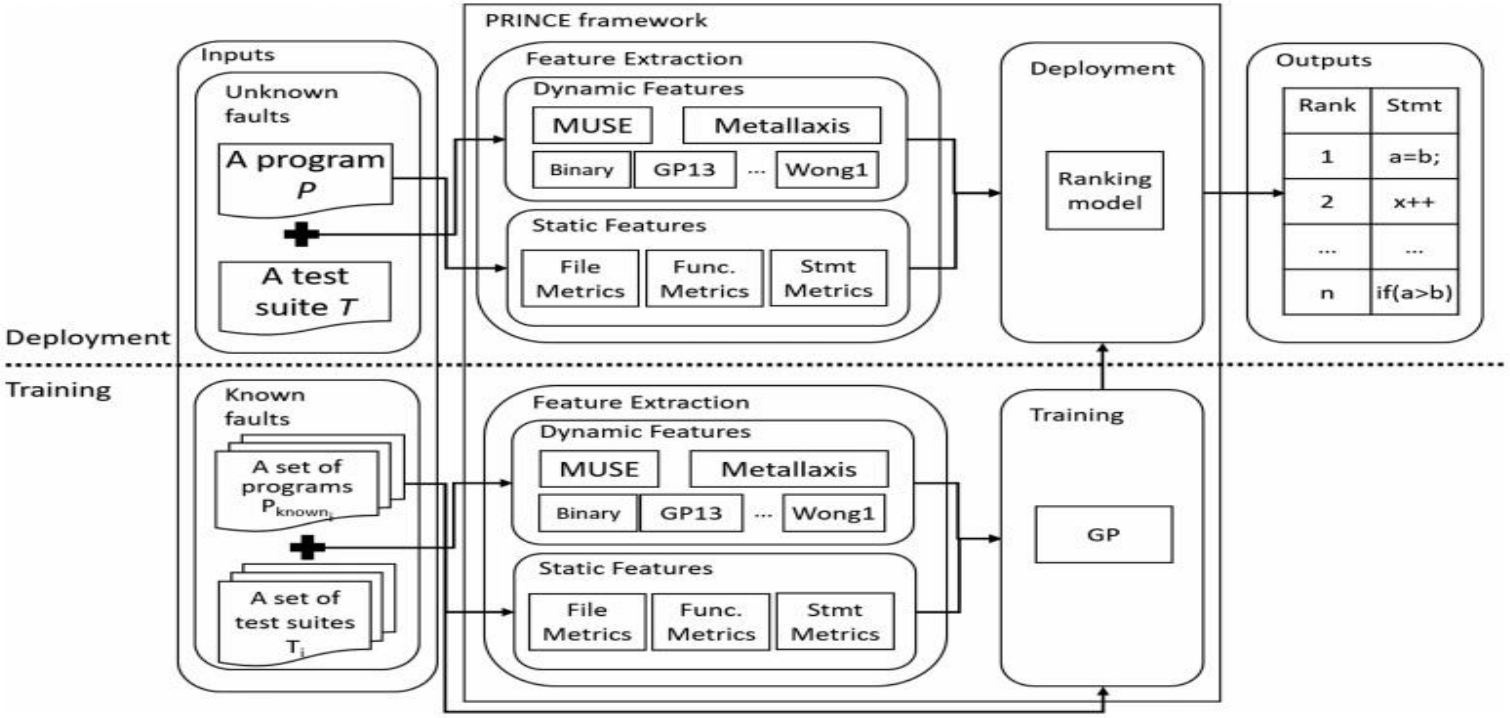
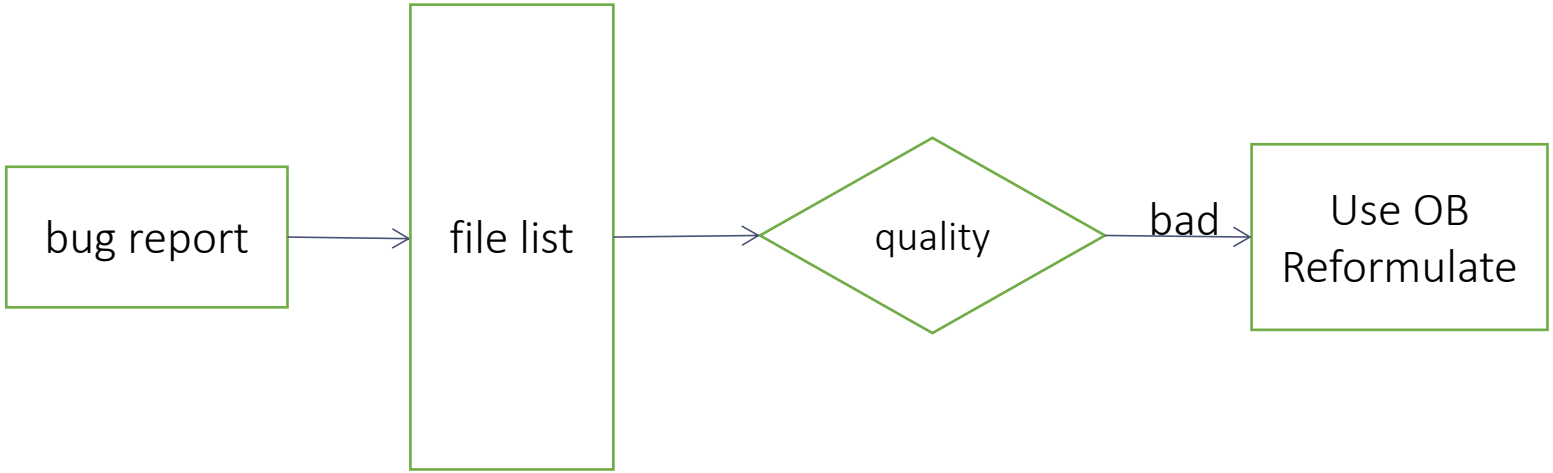


Fig. 1. Overall process of PRINCE.

Using Observed Behavior to Reformulate Queries during Text Retrieval-based Bug Localization

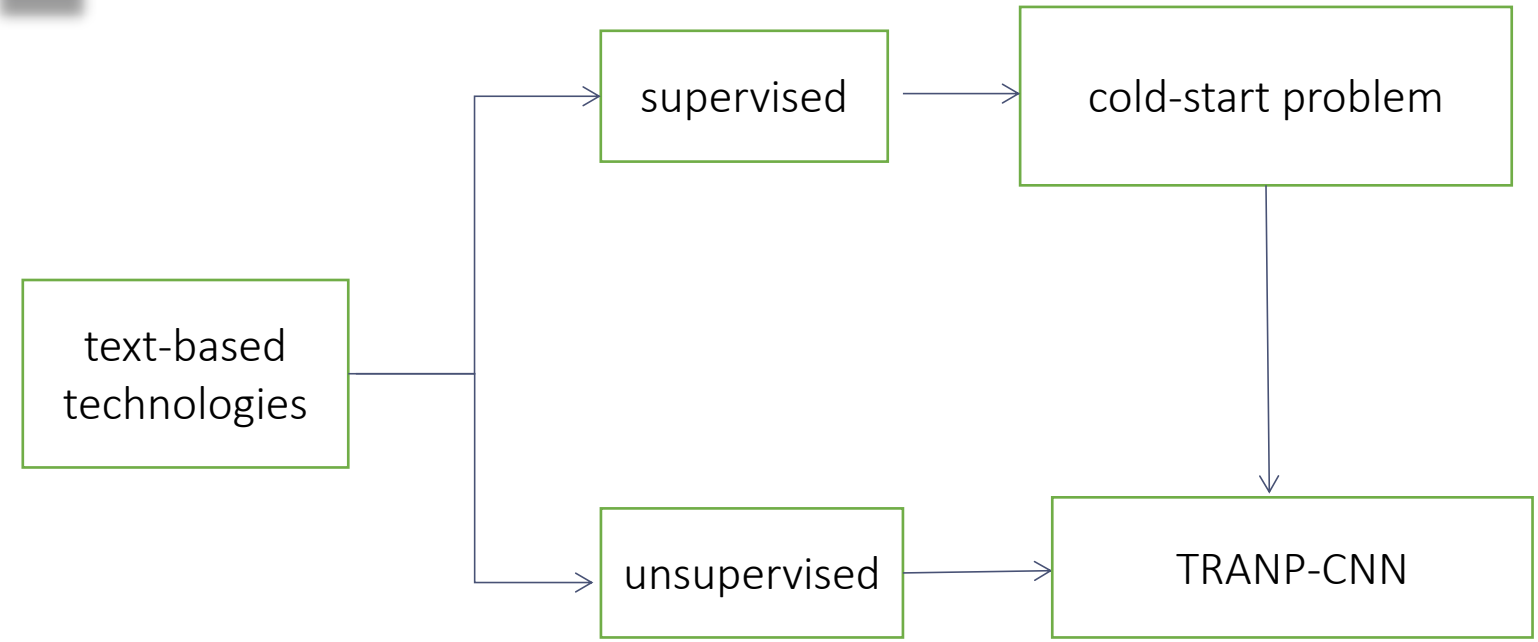


Deep Transfer Bug Localization

汇报人：刘文杰

CONTENTS

Background


supervised
cold-start problem
text-based technologies
unsupervised
TRANP-CNN

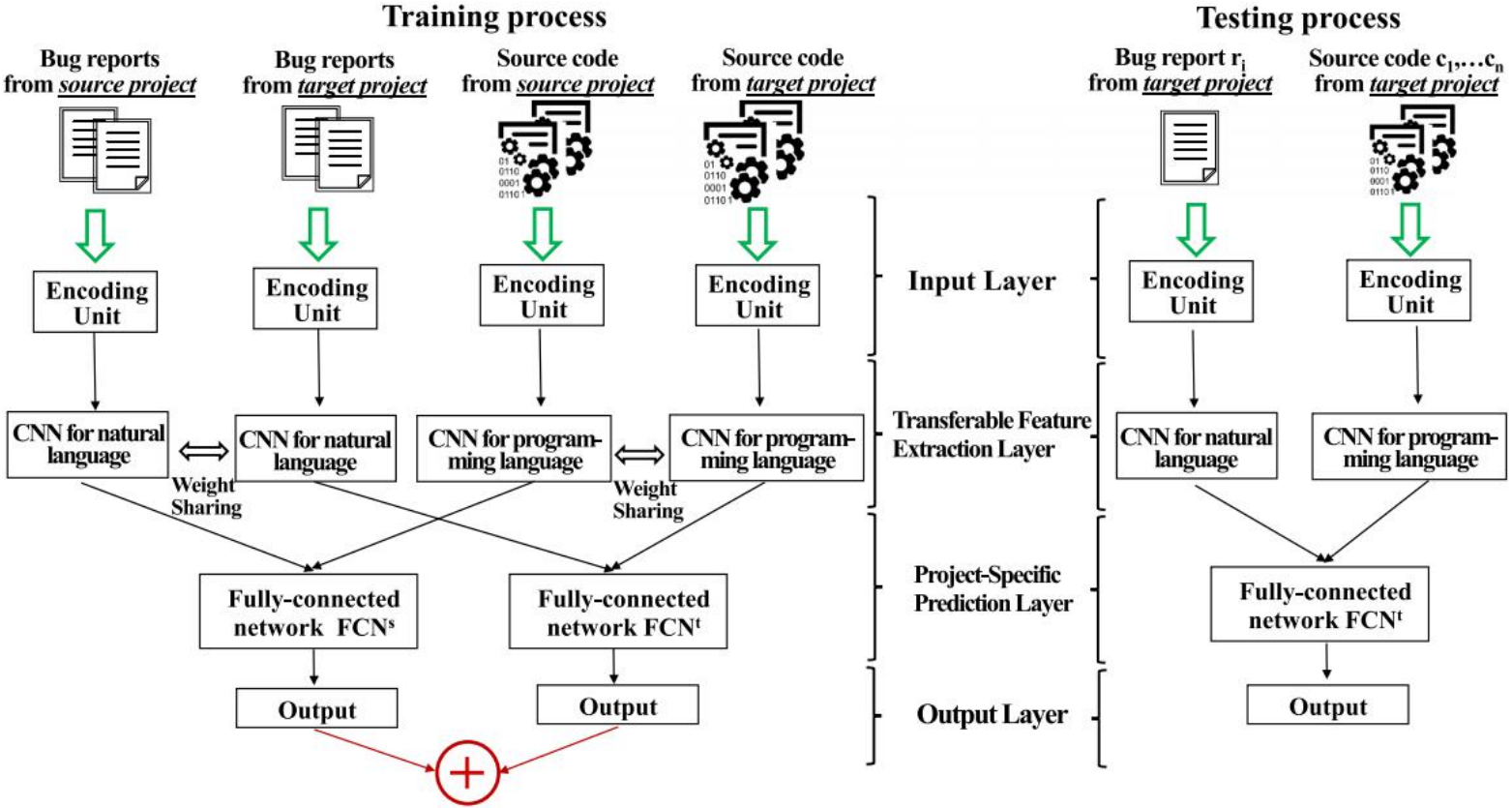
Contribution

aims to learn prediction functions $\mathbf{f} = (f^s, f^t)$, where $f^\alpha : \mathcal{R}^\alpha \times \mathcal{C}^\alpha \mapsto \mathcal{Y}^\alpha$. $y_{ij}^\alpha \in \mathcal{Y}^\alpha = \{+1, -1\}$ indicates whether a source code file $c_j^\alpha \in \mathcal{C}^\alpha$ is relevant to a bug report $r_i^\alpha \in \mathcal{R}^\alpha$, and $\alpha \in \{s, t\}$.

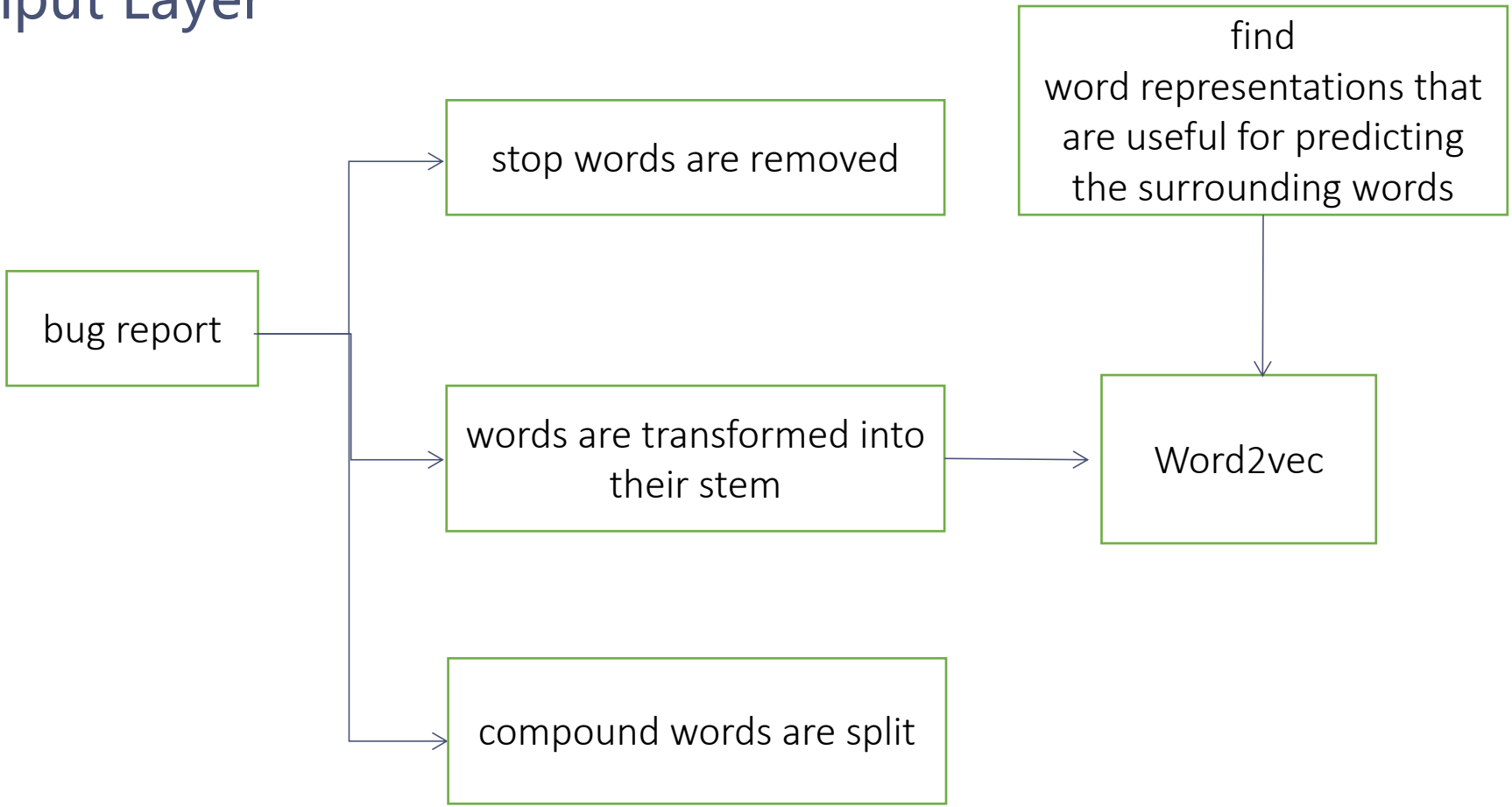
Contribution：

1. We present a novel direction of research on cross-project bug localization
2. We propose a novel deep transfer learning model
3. TRANP-CNN is better performances

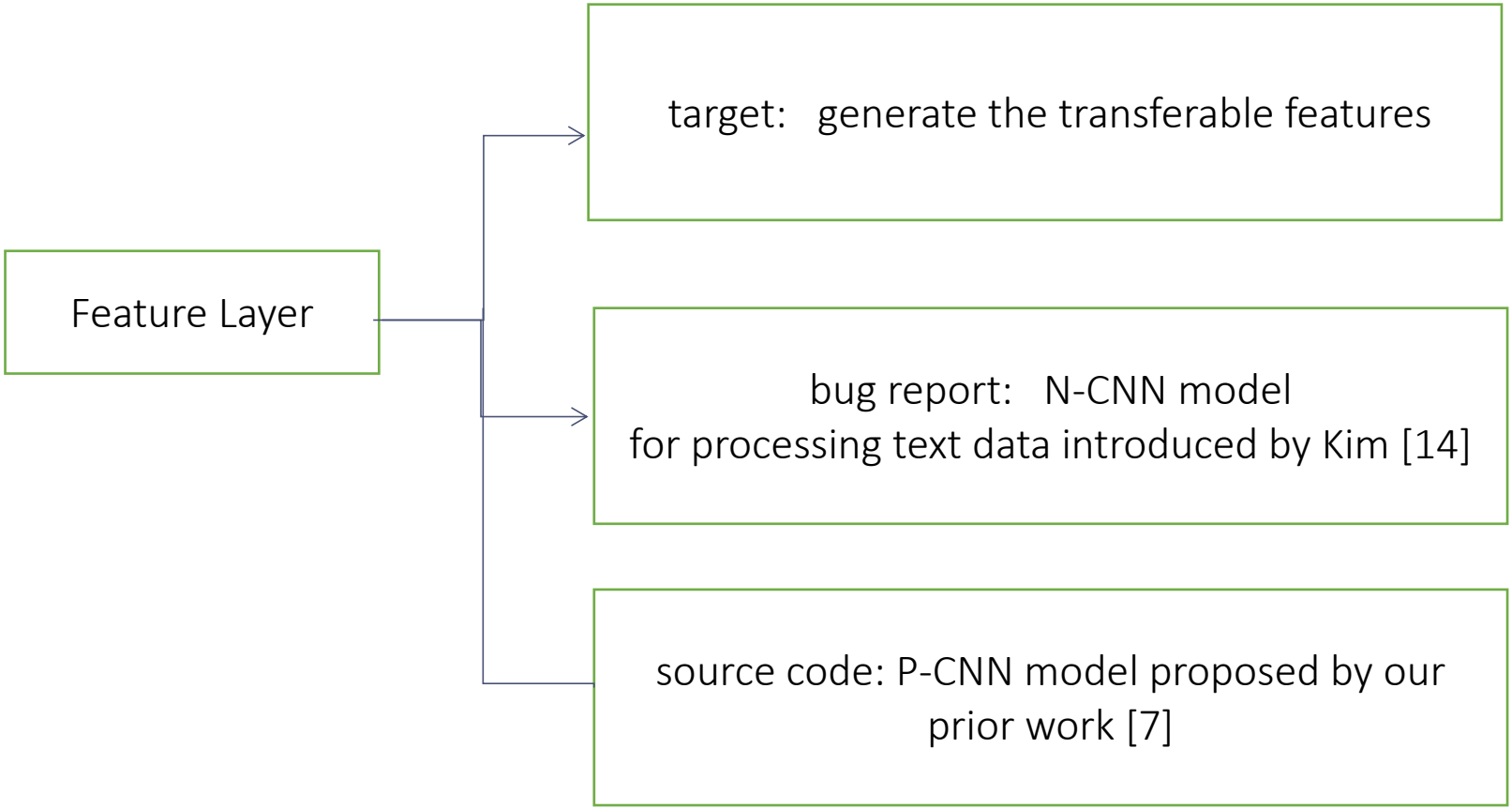
TRANP-CNN Structure



Input Layer


find
word representations that
are useful for predicting
the surrounding words
stop words are removed
bug report
words are transformed into
their stem
Word2vec
compound words are split

Transferable Feature Extraction Layer



N-CNN and P-CNN

（xi ∈ Rk be the k-dimensional word vector corresponding to the i-th word in the sentence.）

$$\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n,$$

$$c_i = f(\mathbf{w} \cdot \mathbf{x}_{i:i+h-1} + b).$$

$$\{\mathbf{x}_{1:h}, \mathbf{x}_{2:h+1}, \ldots, \mathbf{x}_{n-h+1:n}\}$$

$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}],$$

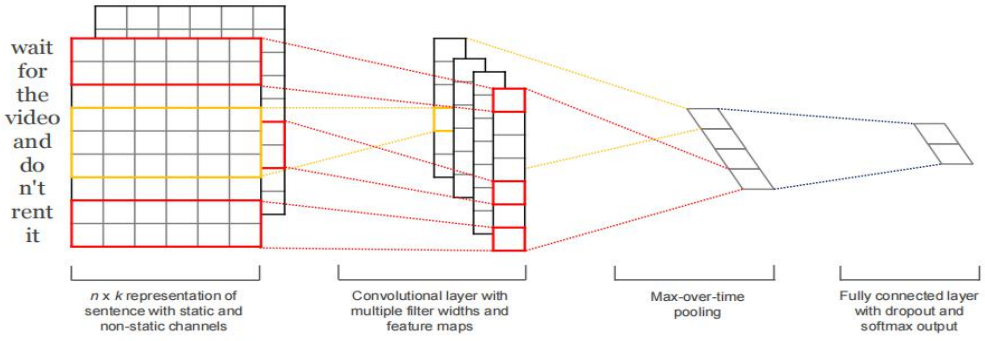


Figure 1: Model architecture with two channels for an example sentence.

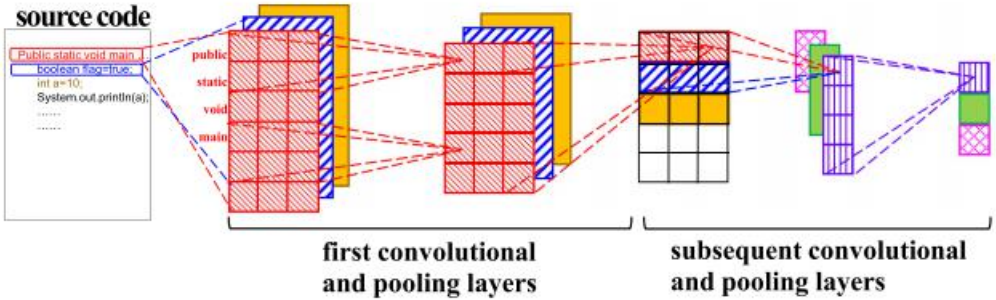
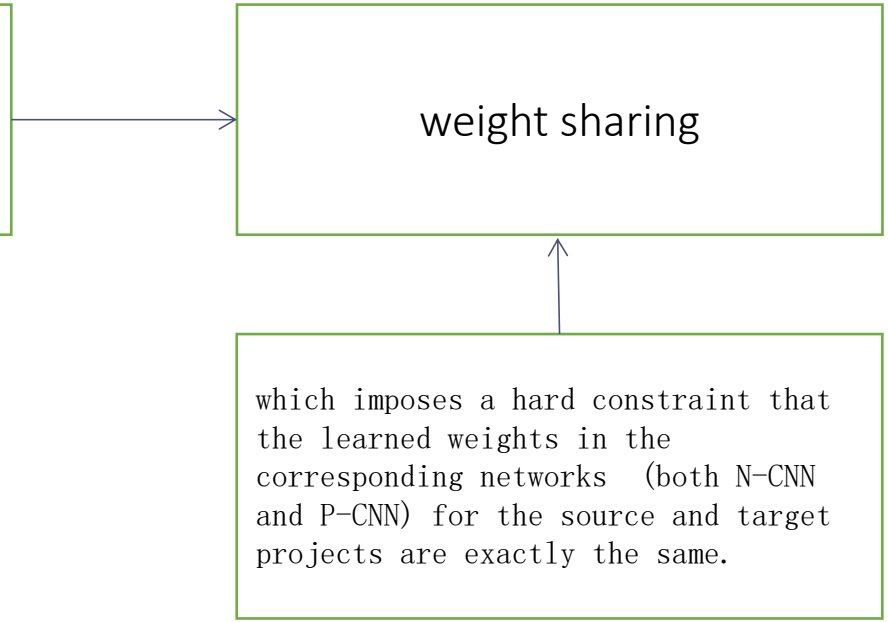


Fig. 2. The structure of convolutional neural network for programming language in the TRANP-CNN model.

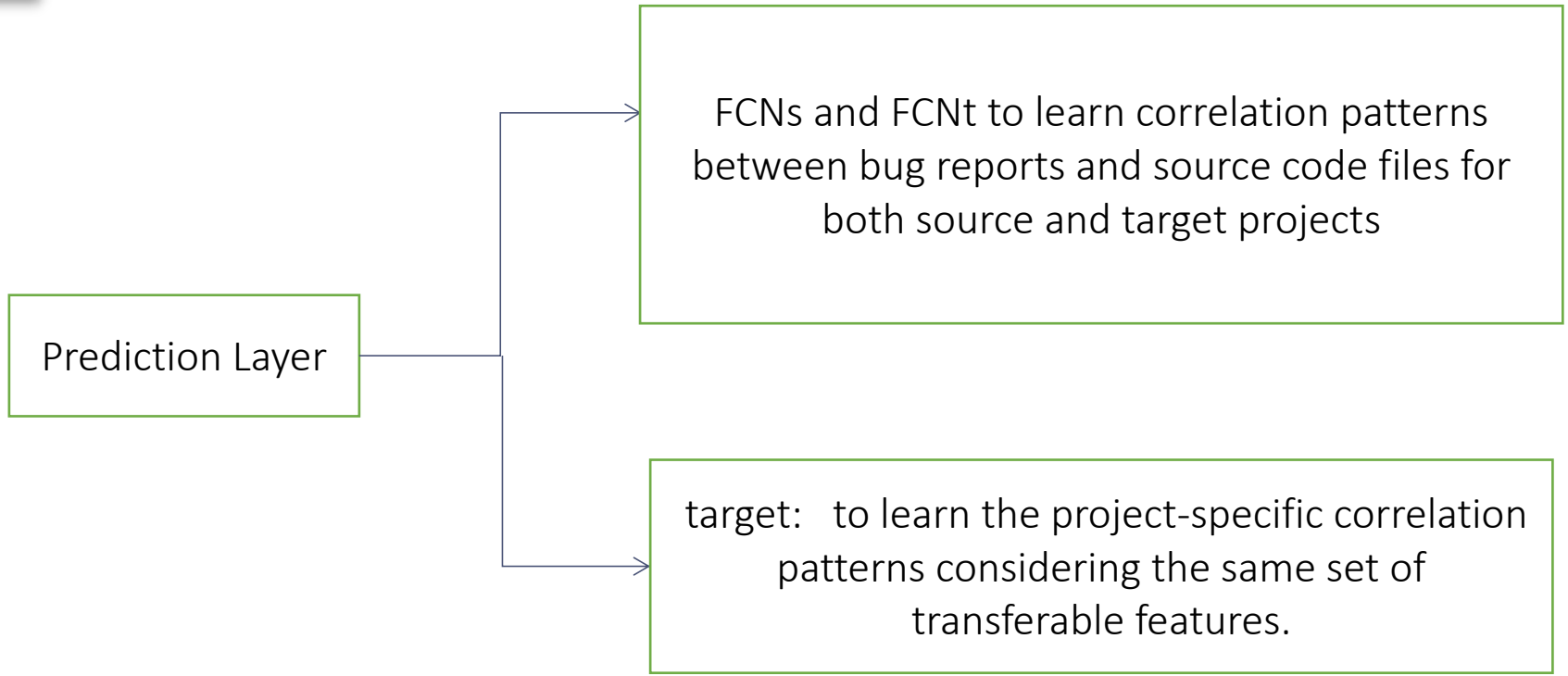
The first layer convert individual statements into intermediate semantic features.
The subsequent layers are used to model interactions between neighboring statements.

Problem

a key challenge is to identify what information is useful and transferable and what is useful but may not be transferable



Project -Specific Prediction Layer



hybrid loss function

$$\mathcal{L}(\Theta, W_f^s, W_f^t) = \sum_{i,j} \ell(\mathbf{x}_{i,j}^s, \mathbf{x}_{i,j}^t, y_{i,j}^s, y_{i,j}^t) + \lambda||W||^2, \quad (2)$$

where

$$\ell(\mathbf{x}_{i,j}^s, \mathbf{x}_{i,j}^t, y_{i,j}^s, y_{i,j}^t) = \hat{\ell}(\mathbf{x}_{i,j}^s, y_{i,j}^s) + \tilde{\ell}(\mathbf{x}_{i,j}^t, y_{i,j}^t)$$
$$\hat{\ell}(\mathbf{x}_{i,j}^s, y_{i,j}^s) = \ell_c(\mathbf{r}_i^s, \mathbf{c}_j^s, y_{i,j}^s)$$
$$\tilde{\ell}(\mathbf{x}_{i,j}^t, y_{i,j}^t) = \ell_c(\mathbf{r}_i^t, \mathbf{c}_j^t, y_{i,j}^t).$$

In the above equation, $W = [\Theta, W_f^s, W_f^t]$ is the vector of all the weights on the network connections to be learned for the entire TRANP-CNN, where Θ is the weights in the convolutional neural networks, W_f^s and W_f^t are weights in fully-connected networks for the source project and the target project, respectively. $(\mathbf{r}_i^s, \mathbf{c}_i^s)$ and $(\mathbf{r}_i^t, \mathbf{c}_i^t)$ are report-code

third term is a regularization term controlled by a hyper-parameter λ to minimize the risk of over-fitting. To effi-

Algorithm 1. TRANP-CNN for bug Localization

Require: the set of training modules from source project $\mathcal{X}^s$, the set of training modules from target project $\mathcal{X}^t$, a new bug report r_k^t in the target project for testing

Process:

1: Encode bug reports and source code from source and target projects into vector representation $\mathbf{x}_{i,j}^s = (\mathbf{r}_i^s, \mathbf{c}_j^s)$, $\mathbf{x}_{i,j}^t = (\mathbf{r}_i^t, \mathbf{c}_j^t)$.
2: **repeat**
3: Randomly pick pairs of bug reports and source code from source and target projects without replacement to create a training batch.
4: Forward propagate the source project data and calculate its training loss $\hat{\ell}(\mathbf{x}_{i,j}^s, y_{i,j}^s)$ by Eq. (2).
5: Forward propagate the target project data and calculate the training loss $\tilde{\ell}(\mathbf{x}_{i,j}^t, y_{i,j}^t)$ by Eq. (2).
6: Calculate the derivatives $\partial\mathcal{L}/\partial W_f^s$, $\partial\mathcal{L}/\partial W_f^t$, $\partial\mathcal{L}/\partial\Theta$. Back propagate and update the network parameters W_f^s, W_f^t, Θ.
7: **until** the max-iteration is reached or convergence
8: Given a bug report r_k in the target project, encode feature representation and use the trained model to generate a potentially buggy source code files list.

Research Questions

RQ1: Is there a need for a specialized technique for cross-project bug localization?
RQ2: Can TRANP-CNN outperform other traditional bug localization methods?
RQ3: Do the weight sharing strategy and the project-specific prediction layer in TRANP-CNN improve bug localization performance?

EXPERIMENTS

Datasets:
63 bug reports from HTTPClient (H), 534 bug reports from Jackrabbit (J), and 196 bug reports from Lucene-solr (L), which are provided by Kochhar et al,AspectJ(A) is 371 links between bug reports and source code files by investigating commit logs

Evaluation Metrics:
TOP-N(N sets 1,5,10),
Mean Average Precision (MAP),
Mean Reciprocal Rank(MRR)

Baselines：
VSM the baseline method
BugLocator,AmaLgam, state-of-the-art traditional methods
DNNLOC,NP-CNN, state-of-the-art deep learning
Burak,TCA+,TCA+P,TCA+D, for cross-project software mining problems

RQ1

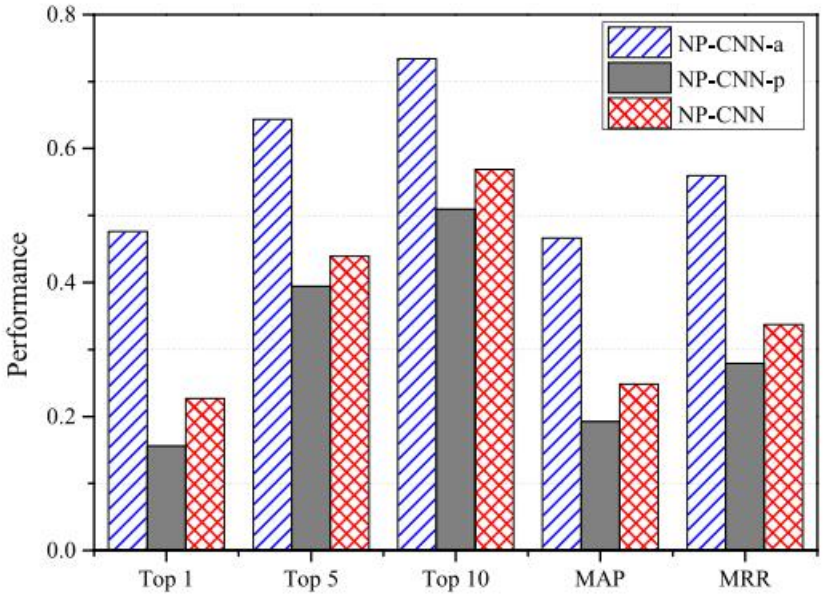


Fig. 3. Performance comparisons between within-project and cross-project bug localization.

NP-CNN-a: Directly employ the NP-CNN model to locate buggy source code files in the target project.(data rich)
NP-CNN-p: Directly employ the NP-CNN model to locate buggy source code files in the target project.(data rare)
NP-CNN: using some labeled data from a source project for training,

RQ2

TABLE 2
Performance Comparisons with Traditional Bug Localization Models

Tasks	Method	Top 1	Top 5	Top 10	MAP	MRR	Tasks	Method	Top 1	Top 5	Top 10	MAP	MRR
H→J	TRANP-CNN	**0.270** •	**0.478** •	**0.589** •	**0.328** •	**0.418** •	**H→L**	TRANP-CNN	**0.308** •	**0.547** •	**0.598**	**0.293** •	**0.418** •
	DNNLOC	0.199	0.398	0.493	0.289	0.306		DNNLOC	0.220	0.472	0.583	0.277	0.379
	VSM	0.163	0.287	0.466	0.234	0.220		VSM	0.164	0.468	0.518	0.242	0.334
	BugLocator	0.182	0.327	0.523	0.260	0.273		BugLocator	0.160	0.435	0.523	0.242	0.347
	AmaLgam	0.201	0.325	0.535	0.277	0.291		AmaLgam	0.168	0.431	0.536	0.251	0.349
	Burak	0.175	0.292	0.426	0.214	0.216		Burak	0.181	0.472	0.546	0.263	0.356
	TCA+	0.212	0.316	0.442	0.237	0.240		TCA+	0.241	0.424	0.581	0.271	0.386
	TCA+P	0.212	0.314	0.443	0.235	0.240		TCA+P	0.237	0.425	0.585	0.271	0.383
	TCA+D	0.214	0.321	0.451	0.239	0.243		TCA+D	0.246	0.435	0.589	0.278	0.393
L→J	TRANP-CNN	**0.257** •	**0.465** •	**0.567** •	**0.351** •	**0.438** •	**J→L**	TRANP-CNN	0.257	0.486	**0.583**	**0.262**	**0.386**
	DNNLOC	0.123	0.349	0.448	0.216	0.269		DNNLOC	**0.258**	**0.494**	0.580	0.257	0.373
	VSM	0.155	0.273	0.406	0.213	0.248		VSM	0.140	0.412	0.515	0.236	0.348
	BugLocator	0.193	0.301	0.425	0.230	0.286		BugLocator	0.157	0.421	0.523	0.239	0.350
	AmaLgam	0.191	0.321	0.430	0.241	0.284		AmaLgam	0.168	0.422	0.527	0.245	0.361
	Burak	0.140	0.290	0.410	0.228	0.263		Burak	0.188	0.422	0.535	0.233	0.358
	TCA+	0.118	0.295	0.436	0.221	0.246		TCA+	0.142	0.429	0.560	0.246	0.350
	TCA+P	0.115	0.293	0.447	0.220	0.245		TCA+P	0.147	0.426	0.551	0.242	0.348
	TCA+D	0.119	0.296	0.445	0.223	0.248		TCA+D	0.154	0.438	0.564	0.247	0.352
A→J	TRANP-CNN	**0.348** •	**0.512** •	**0.596** •	**0.376** •	**0.469** •	**A→L**	TRANP-CNN	**0.293** •	**0.490** •	**0.592** •	**0.295** •	**0.410** •
	DNNLOC	0.180	0.371	0.462	0.272	0.290		DNNLOC	0.245	0.460	0.568	0.262	0.362
	VSM	0.158	0.333	0.480	0.258	0.254		VSM	0.181	0.376	0.486	0.195	0.286
	BugLocator	0.180	0.352	0.472	0.265	0.283		BugLocator	0.173	0.398	0.531	0.227	0.319
	AmaLgam	0.191	0.347	0.452	0.257	0.286		AmaLgam	0.178	0.389	0.527	0.230	0.331
	Burak	0.160	0.380	0.480	0.248	0.273		Burak	0.189	0.419	0.556	0.237	0.336
	TCA+	0.162	0.379	0.483	0.241	0.246		TCA+	0.164	0.421	0.512	0.196	0.291
	TCA+P	0.155	0.390	0.487	0.239	0.243		TCA+P	0.167	0.427	0.515	0.196	0.288
	TCA+D	0.145	0.327	0.490	0.215	0.229		TCA+D	0.169	0.433	0.521	0.196	0.293

RQ3

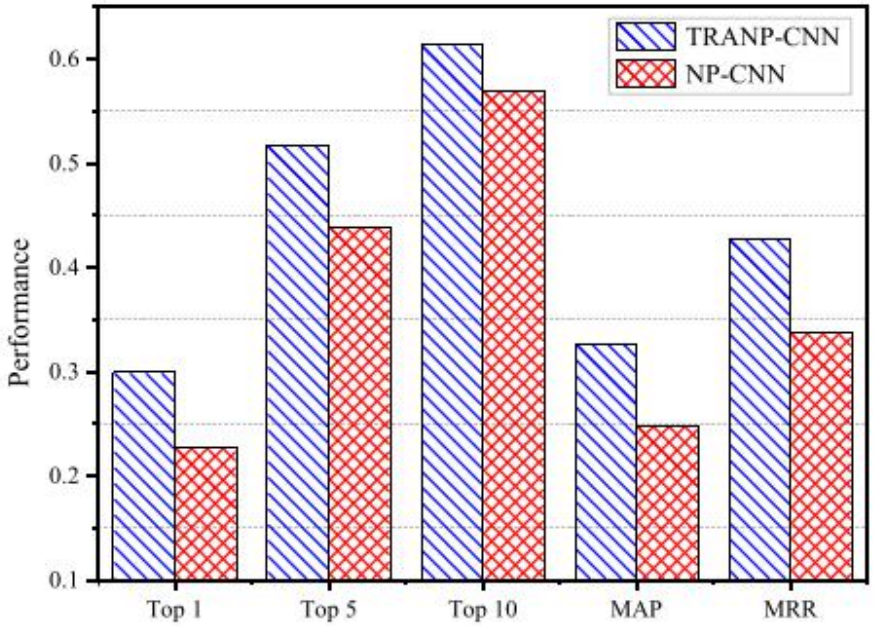


Fig. 5. Performance comparisons with existing deep models.

The main differences between TRANP-CNN and NP-CNN are

(1) the use of the weight sharing strategy (i.e., in the transferable feature extraction layer)

(2) the two fully-connected net-works that combine deep features from source and target projects

谢谢观看