USENIX Security '25 Artifact Appendix: BLens: Contrastive Captioning of Binary Functions using Ensemble Embedding

Tristan Benoit*†¶, Yunru Wang*†‡, Moritz Dannehl†‡, Johannes Kinder†‡

† Ludwig-Maximilians-Universität München, Germany
‡ Munich Center for Machine Learning, Germany
¶ Bundeswehr University Munich, Germany

A Artifact Appendix

A.1 Abstract

The artifact provides the datasets, implementation, and evaluation framework for BLens, a novel function name prediction model for stripped binaries. BLens combines COMBO (**CO**ntrastive **M**ulti-modal **B**inary embedding **O**ptimizer) to capture spatial relationships within the binary code and LORD (**L**ikelihood **O**rdered **R**egressive **D**ecoder) to improve function name prediction precision. We conduct ablation studies to validate the effectiveness of both components and compare BLens with several state-of-the-art tools in cross-binary and cross-project settings. Additionally, we introduce a strict setting to further evaluate generalization. This appendix contains the necessary information for reproducing the results and verifying the functionality of BLens

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifact does not disable security mechanisms or process sensitive data, so it poses no security or privacy risks.

A.2.2 How to access

The artifact package can be accessed via GitHub repository, which includes the code, evaluation logs, and supplementary materials for implementation details. The Zenodo record contains all datasets in addition to the items available on GitHub.

A.2.3 Hardware dependencies

The minimum hardware requirements to reproduce the training and evaluation are 30 GB of memory, a GPU with 80 GB of VRAM, and 200 GB of available storage space.

A.2.4 Software dependencies

We recommend using a Python virtual environment, which may require virtualenvwrapper. Enchant is needed for the tokenizer, and Rust is required for VarCLR calculations. Installation instructions are provided in the `INSTALL.md` file in the GitHub repository.

A.2.5 Benchmarks

Pre-computed embeddings, tokenizers, and logs are contained in the data folder from the Zenodo record.

A.3 Set-up

A.3.1 Installation

1) Clone the package from the GitHub repository.
2) Download and extract `data.tar.gz` from the Zenodo record.
3) Follow the instructions in the `INSTALL.md`.

A.3.2 Basic Test

Once the installation is completed, users can do the following tests to check if the tool is properly installed.
1) Test the training work flow.

```
$ workon blens
$ CUDA_VISIBLE_DEVICES=0 nohup python3 RunExp.py -data-
    ↪ dir=DATA-FOLDER --basic-test >
    ↪ basic_test_training.txt &
```

2) Test the evaluation work flow.

```
$ cd evaluation
$ python3 new_experiments_evaluator.py -data-dir=DATA-
    ↪ FOLDER -d=basic_test --cross-project >
    ↪ basic_test_evaluation.txt
```

If the installation is successful, 11 files will be generated in the `DATA-FOLDER/xp/basic_test` directory.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): BLens achieves higher metrics compared to state-of-the-art competitors in the cross-project and strict settings as well as in the easier cross-binary setting. This

is proven by the experiment (E1), whose results are reported in Table 3 and 5, and experiment (E2) reported in Tables 1, 3 and 5.

(C2): COMBO pre-training and the new LORD decoder are shown to be beneficial in terms of F_1 score. This is proven by experiments (E3), (E4), and (E5) reported in Tables 6 and 8 of the ablation subsection (6.4).

A.4.2 Experiments

Given the time constraints of the artifact evaluation, we have structured our evaluation accordingly.

In the paper, (C1) is evaluated by retraining competitors such as XFL, SymLM, and AsmDepictor. Addtionally, BLens is not only trained on the full dataset in both cross-binary and cross-project settings; for better comparison, two variants are further trained on subdatasets generated from SymLM and AsmDepictor preprocessing. In all these settings, we consider F_1 scores as well as RougeL, Bleu, and VarCLR scores.

In the artifact evaluation, we propose only training a new BLens model in the cross-project setting. Other experiments are provided in the form of logs containing precomputed predictions. Lastly, we primarily evaluate our experiments using the F_1 score, which is our central metric.

In the paper, (C2) is evaluated using five models. The first is the full BLens model, trained for 80 epochs, similar to each ablation model. The variations include: BLens without pre-training (BL-NP); BLens with only the pre-training phase; BLens with a simple decoder that uses teacher forcing while maintaining a threshold; and BLens with teacher forcing but without implementing the threshold (SIMPLE-T0).

In the artifact evaluation, we propose to only train new models for BL-NP, and SIMPLE-T0. Other experiments are provided in the form of logs containing precomputed predictions.

(E1): *[5 human-minutes + 96 compute-hours]* Experiment E1 is to validate Claim C1 regarding the cross-project and strict settings. We validate this claim by training a new model from scratch in the cross-project setting and measuring F_1 scores in the cross-project and strict settings.

Execution: Train the model for around 96 hours:

```
$ workon blens
$ CUDA_VISIBLE_DEVICES=0 nohup python3 RunExp.py -data-
    ↪ dir=DATA-FOLDER -d=new-cross-project -config=
    ↪ main.json --cross-project -pretrain -train -
    ↪ inferBest > c1-cross-project-training.txt &
```

Get metrics for this new model:

```
$ cd evaluation
$ python3 new_experiments_evaluator.py -data-dir=DATA-
    ↪ FOLDER -d=new-cross-project --cross-project >
    ↪ c1-new-cross-project.txt
```

```
Main results
Setting: cross-project
       Bleu     F1  Precision  Recall  RougeL  VarCLR
test  0.242   0.46      0.655   0.354   0.393   0.648

Setting: intermediate strict
       Bleu     F1  Precision  Recall  RougeL  VarCLR
test  0.196  0.404      0.604   0.304   0.307   0.599

Setting: strict
       Bleu     F1  Precision  Recall  RougeL  VarCLR
test  0.094  0.294      0.482   0.211   0.243   0.568
```

Figure 1: Example of `c1-new-cross-project.txt` showing the result of retraining BLens in the cross-project setting.

Results: Review the file `c1-new-cross-project.txt`. Compare the F_1 score of BLens in the cross-project setting in Table 3 from the paper, which is 0.46, with the F_1 score in the table titled `Setting: cross-project` produced by the script. Compare the F_1 score of BLens in the strict setting in Table 5 from the paper, which is 0.294, with the one produced by the script. A difference of around ± 0.02 is expected due to weight initialization.

(E2): *[5 human-minutes + 10 compute-minutes]* Experiment E2 is to validate Claim C1 regarding other models and the cross-binary setting. We validate this claim by measuring F_1 scores from precomputed predictions in logs.

Execution: Compute metrics based on logs:

```
$ cd evaluation
$ python3 evaluator_c1.py -data-dir=DATA-FOLDER > c1-
    ↪ summary.txt
```

Results: Read the output `c1-summary.txt`. Compare metrics scores in Tables 1, 3 and 5 from the paper to the corresponding metrics in tables produced by the script. There should be no difference because this summary is based on precomputed predictions in logs.

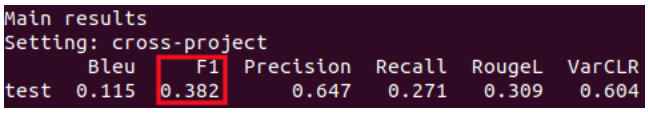
(E3): *[5 human-minutes + 54 compute-hours]* Experiment E3 is to validate (C2). We train a new BL-NP model from scratch and measure F_1 scores for this model.

Execution: Train the model for around 54 hours:

```
$ CUDA_VISIBLE_DEVICES=0 nohup python3 RunExp.py -data-
    ↪ dir=DATA-FOLDER -d=new-np -config=ablation-np.
    ↪ json --cross-project -pretrain -train -
    ↪ inferBest > c2-np-training.txt &
```

Get metrics for this new model:

```
$ cd evaluation
$ python3 new_experiments_evaluator.py -data-dir=DATA-
    ↪ FOLDER -d=new-np --cross-project > c2-new-np.
    ↪ txt
```



```
Main results
Setting: cross-project
       Bleu     F1  Precision  Recall  RougeL  VarCLR
test  0.115  0.382      0.647   0.271   0.309   0.604
```

Figure 2: Example of `c2-new-np.txt` showing the result of re-training BL-NP.

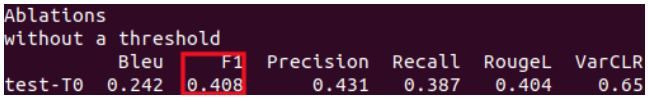
Results: Review the file `c2-new-np.txt`. Compare the F_1 score of BL-NP from Table 6 in the paper, which is 0.287, with the one found in the table titled `Setting : cross-project` generated by the script. Expect the score to be around 0.287 ± 0.10 due to the instability without COMBO pre-training phase. It should remain well below 0.445.

(E4): *[5 human-minutes + 32 compute-hours]* Experiment E4 is to validate (C2). We train a new SIMPLE-T0 model from scratch and measure F_1 scores for this new model.
Execution: Train the model for around 32 hours:

```
$ CUDA_VISIBLE_DEVICES=0 nohup python3 RunExp.py -data-
    ↪ dir=DATA-FOLDER -d=new-simple-t0 -config=
    ↪ ablation-simple.json  --cross-project -pretrain
    ↪  -train -inferT0 > c2-simple-t0-training.txt &
```

Get metrics for this new model:

```
$ cd evaluation
$ python3 new_experiments_evaluator.py -data-dir=DATA-
    ↪ FOLDER -d=new-simple-t0 --cross-project --
    ↪ evaluateNoTreshold > c2-new-simple-t0.txt
```



```
Ablations
without a threshold
          Bleu     F1  Precision  Recall  RougeL  VarCLR
test-T0  0.242  0.408      0.431   0.387   0.404    0.65
```

Figure 3: Example of `c2-new-simple-t0.txt` showing the result of retraining SIMPLE-T0.

Results: Review the file `c2-new-simple-t0.txt`. Compare the F_1 score of SIMPLE-T0 from Table 8 in the paper, which is 0.409, with the one found in the table titled `Setting: cross-project` generated by the script. Expect the score to be around 0.409 ± 0.01 due to weight initialization. Again, it should remain well below 0.445.

(E5): *[5 human-minutes + 5 compute-minutes]* Experiment E5 is to validate Claim C2 regarding other LORD ablation models. We validate this claim by measuring F_1 scores from precomputed predictions in logs.
Execution: Compute metrics based on logs:

```
$ cd evaluation
$ python3 evaluator_c2.py -data-dir=DATA-FOLDER > c2-
    ↪ summary.txt
```

Results: Read the output `c2-summary.txt`. Compare F_1 scores in Tables 8 from the paper to corresponding scores in the script-produced tables. No differences are expected since this is based on logs.

(E6): *Optional full re-training [5 human-minutes + 47 compute-days]* Experiment E6 is an optional experiment to re-train each BLens model and to measure their metrics. It requires around 47 days of computations with one GPU card.
Preparation: Modify line 3 of the `retrainAll.sh` script to set the correct `DATA-FOLDER` path.
Execution: Run the script:

```
$ nohup bash retrainAll.sh &
```

Results: Read outputs in the `evaluation` folder such as `cb-bl.txt` (for the BLens model in the cross-binary setting). Compare new tables with tables from the paper. Due to weight initialization, expect the F_1 score to differ by around ± 0.02 except in BL-NP case, in which it differs by ± 0.10.

A.5 Notes on Reusability

To employ a different dataset, you should begin by creating a dataset summary pickle file. This file should contain three lists categorizing functions into training, validation, and test sets, and each function within these lists should be represented by an array of the form [binary path, virtual address, real name, untangled name, placeholder, binary identifier, function identifier]. The binary and function identifiers are specifically required for DEXTER embeddings due to its precise database schema. Then, use `Tokenizers.py` to save a tokenizer tailored to your dataset. Subsequently, for each function, generate an embedding using the appropriate pre-processing tools provided by the creators of PALMTREE, DEXTER, and CLAP. These embeddings should be saved in dictionaries with keys as (binaryPath, virtual address) or (binary identifier, function identifier) for DEXTER embeddings, and values should be PyTorch tensors. Once these are saved using pickle, adjust the `RunExp.py` file to load these embeddings and the related dataset summary file and tokenizer.

To extend Blens with additional embeddings, you should modify the `COMBO.py` constructor to incorporate another module based on either `Dexter2Seq.py` for function embeddings or `PalmTree2Seq.py` for basic block embeddings. Update the `combo_input` function within `COMBO.py` to integrate this new module. Additionally, create a corresponding pickle dictionary with keys as (binary path, virtual address). Adjustments should be made to `RunExp.py`, `builder.py` and `COMBO.py` to load

new embeddings and to read the function encoder's new hyperparameters added to the configuration file `configs/main.json`.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2025/.