Multi-modal Machine Learning for Hardening Firmware Binaries

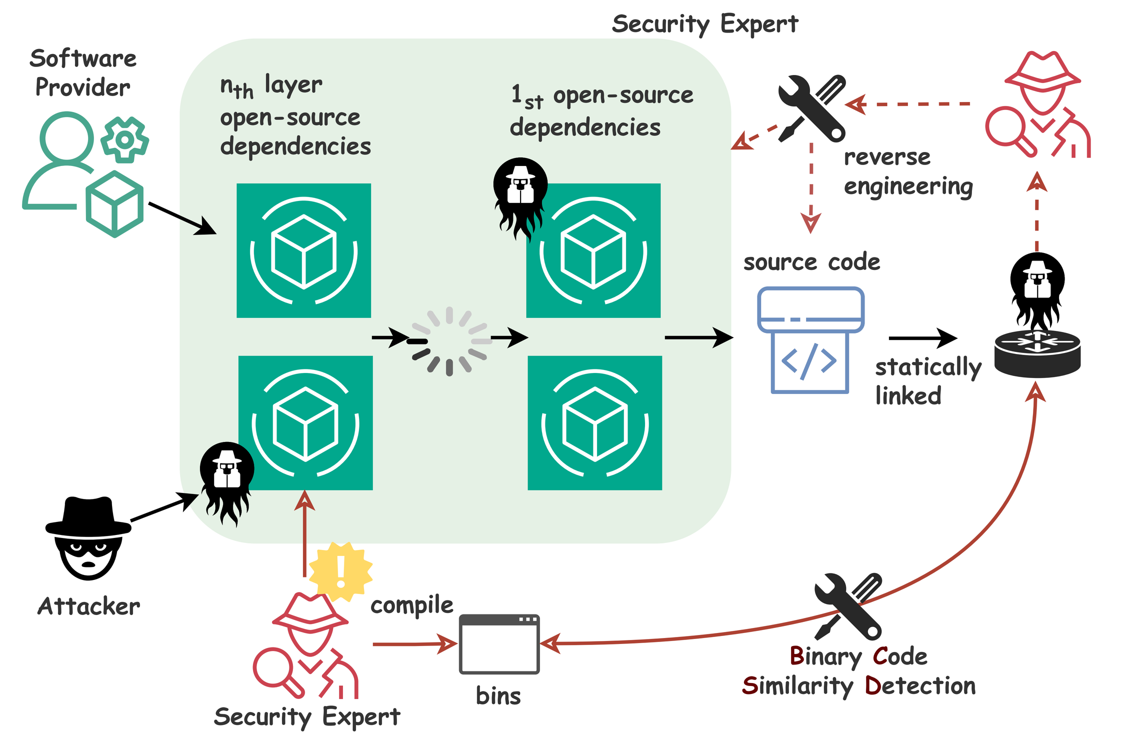
Yunru Wang

yunru.wang@ifi.lmu.de

Supervisors: Johannes Kinder

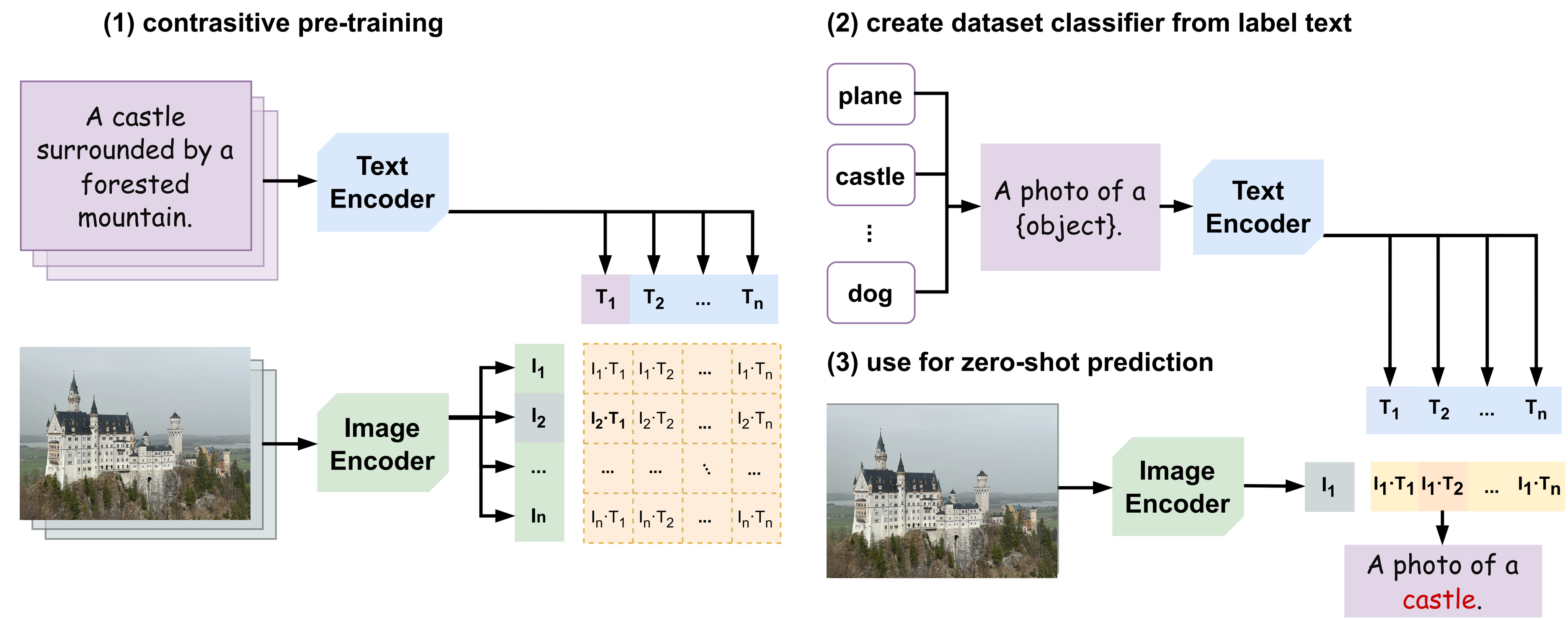
Collaborators: Tristan Benoit

Software Supply Chain



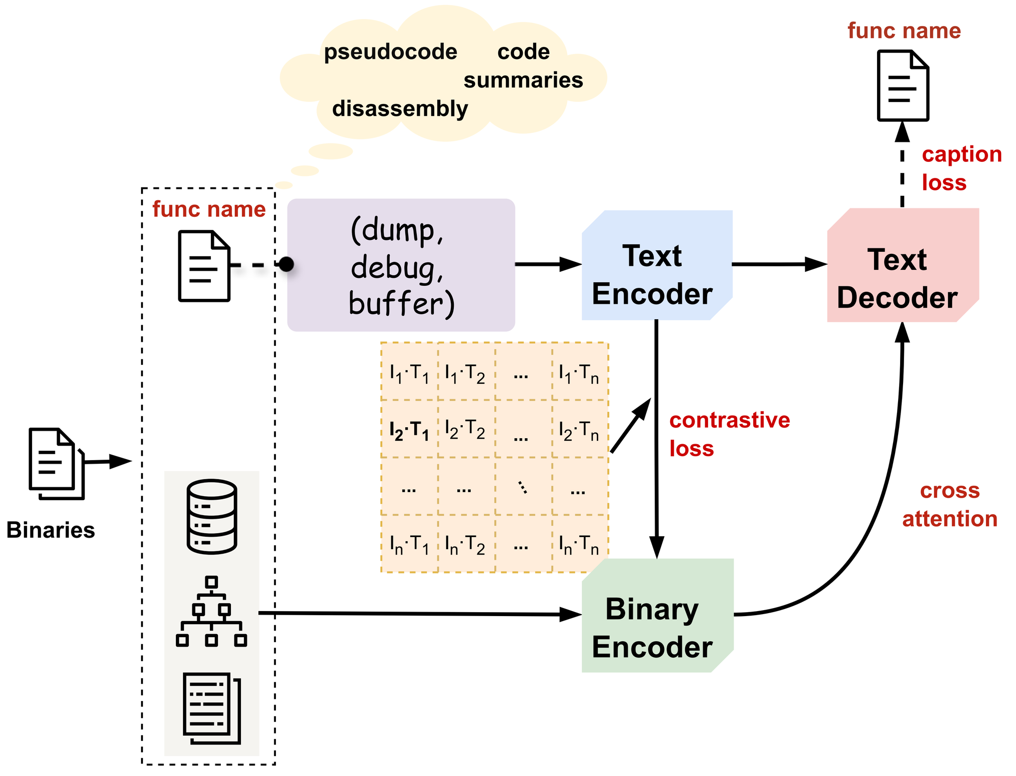
The heavy reliance on **third-party libraries** in embedded firmware heightens software supply chain security risks. **BCSD** addresses known vulnerabilities, while **reverse engineering** reveals unknown ones.

CLIP: a SotA MMML Architecture [1]



CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset's classes.

MMML for Binary Code



- **Align** binary encoding with function names in latent space to generalize to **zero-shot learning**.
- Reconstruct **high-level structures** from binaries to assist in reverse engineering.
- **Generalize** to binaries across domains and different downstream tasks.

Loss Functions

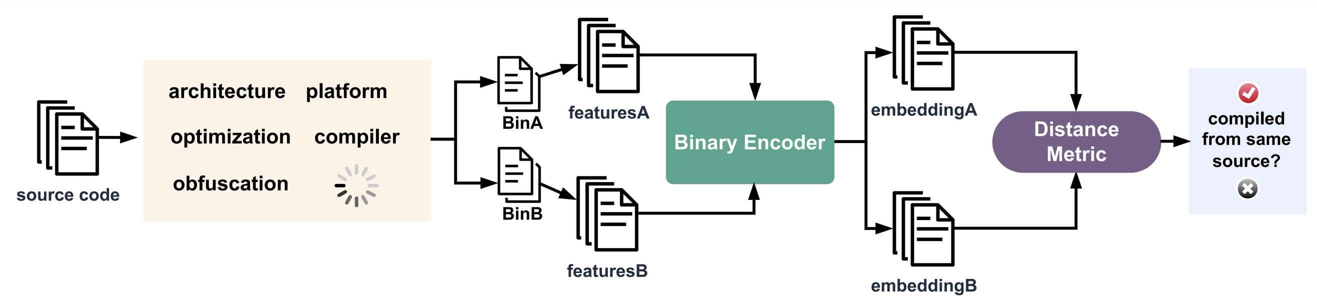
$$\mathcal{L}_{\text{Contrastive}} = -\frac{1}{N}\left(\sum_{i=1}^{N}\log\frac{\exp(x_i^\top y_i/\sigma)}{\sum_{j=1}^{N}\exp(x_i^\top y_j/\sigma)} + \sum_{i=1}^{N}\log\frac{\exp(y_i^\top x_i/\sigma)}{\sum_{j=1}^{N}\exp(y_i^\top x_j/\sigma)}\right),$$

$$\mathcal{L}_{\text{Caption}} = -\sum_{t=1}^{T}\log P_\theta(y_t|y_{<t}, x),$$

$$\mathcal{L}_{\text{Sum}} = \lambda_{\text{Contrastive}}\cdot\mathcal{L}_{\text{Contrastive}} + \lambda_{\text{Caption}}\cdot\mathcal{L}_{\text{Caption}}.$$

Here, x_i and y_j denote binary and function name embeddings in the i-th and j-th pairs. N represents the batch size, and σ is the temperature to scale the logits.

BCSD Scenario



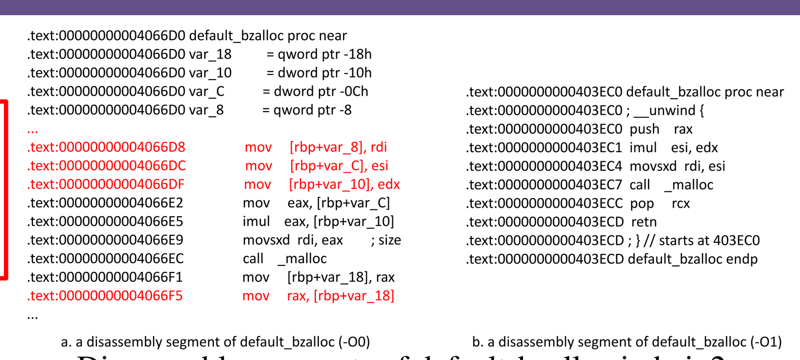
Evaluate binary similarity by computing the cosine distance between embeddings generated by the trained binary encoder.

Optimization Levels

Binaries compiled with different configurations vary significantly. For instance, in O0 optimization, call arguments are pushed onto the stack, whereas they are optimized in O1.

```
static void* default_bzalloc ( void* opaque, Int32 items, Int32 size )
{
   void* v = malloc ( items * size );
   return v;
}
```

Source code of default_bzalloc in bzip2.



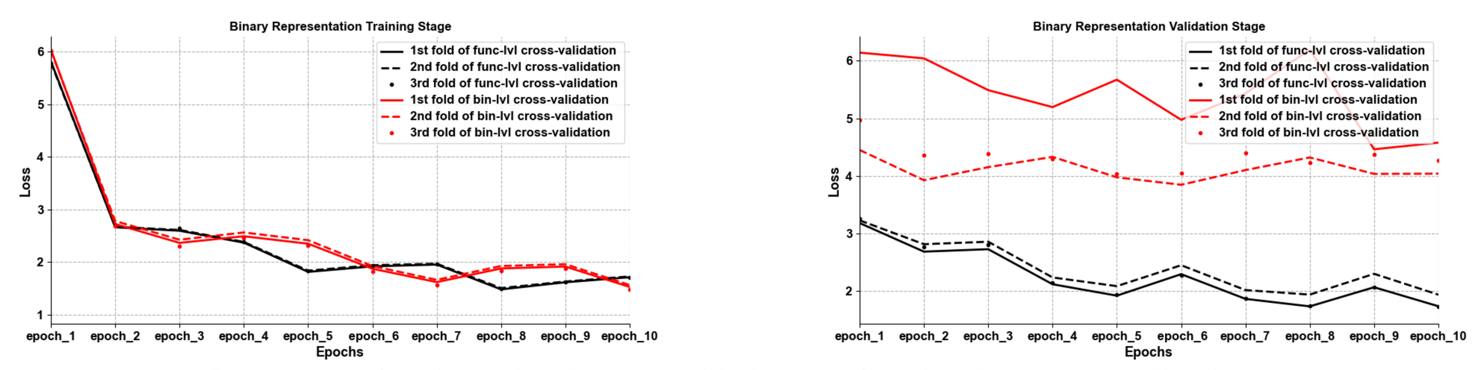
Disassembly segments of default_bzalloc in bzip2.

Binary Distribution

In small-scale function name generation experiments, we observed significant differences in results based on the splitting strategy:

- F1 score averaged **0.6646** when splitting by functions
- F1 score averaged **0.4708** when splitting by binaries

We also noticed poor generalization between binaries when evaluating other state-of-the-art approaches.



Loss curves for binary-level cross-validation and function-level cross-validation.

References

[1] A. Radford et al. "Learning transferable visual models from natural language supervision". In: *International Conference on Machine Learning*. ICML. 2021, pp. 8748–8763.