

Making Software Debloating Production Ready

Ahmed Ali-Eldin
Chalmers University of Technology

Mostly work done by Huaifeng (who is not here!)





David, by Michelangelo, 1504

“It’s simple. I just remove everything that is not David.”

—Michelangelo

Debloating Containers

Debloating Containers

Removing

Debloating Containers

Removing

Software Bloat

Debloating Containers

Removing
Software Bloat
in
Containers

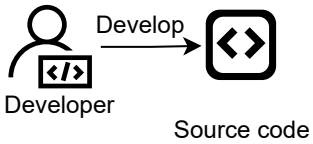
Software Bloat

Software bloat are unnecessary code, files, libraries for an application during run time.

Bloat in Software Development

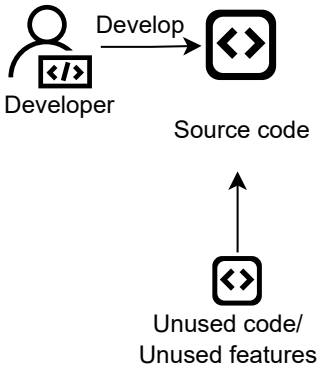
- Each stage of current software development introduces new source of software bloat.

Bloat in Software Development



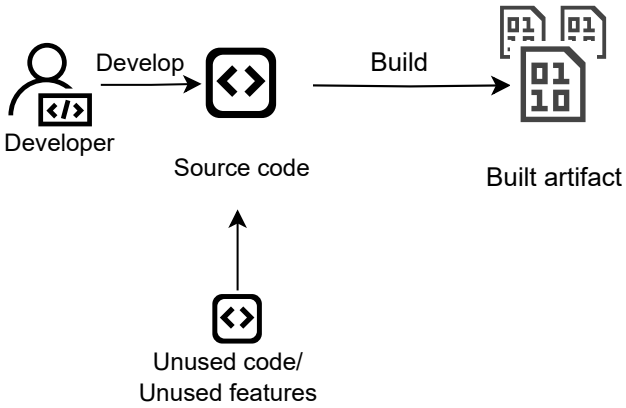
- Each stage of current software development introduces new source of software bloat.

Bloat in Software Development



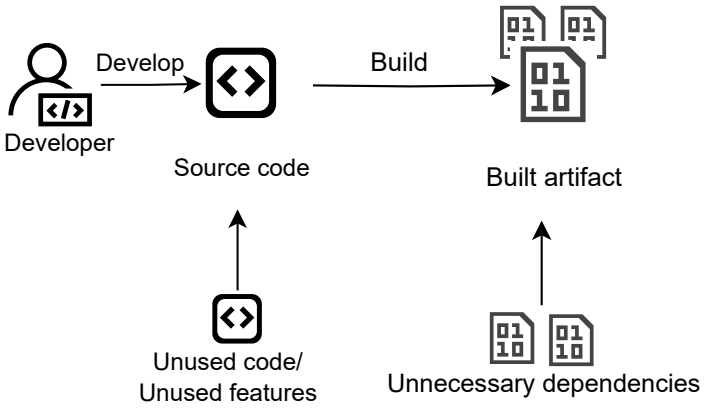
- Each stage of current software development introduces new source of software bloat.

Bloat in Software Development



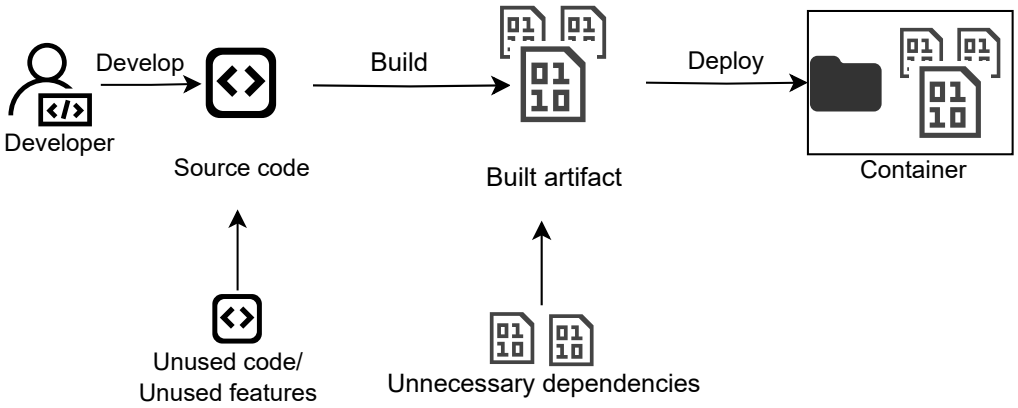
- Each stage of current software development introduces new source of software bloat.

Bloat in Software Development



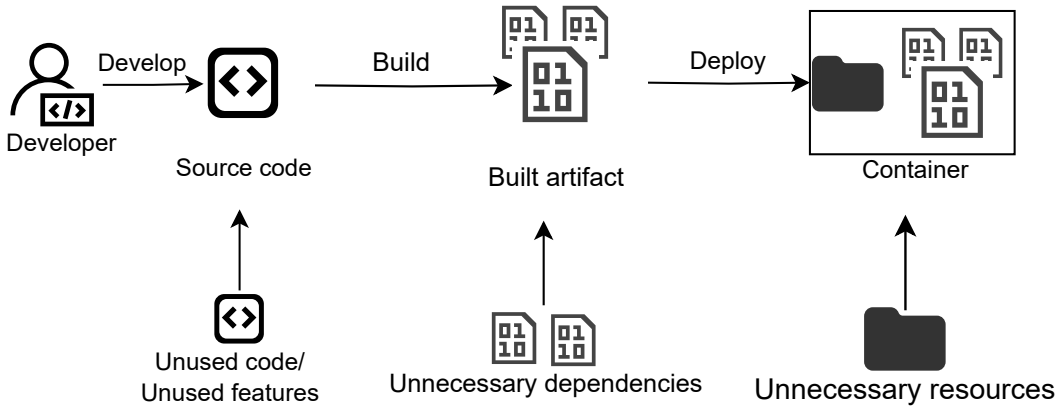
- Each stage of current software development introduces new source of software bloat.

Bloat in Software Development



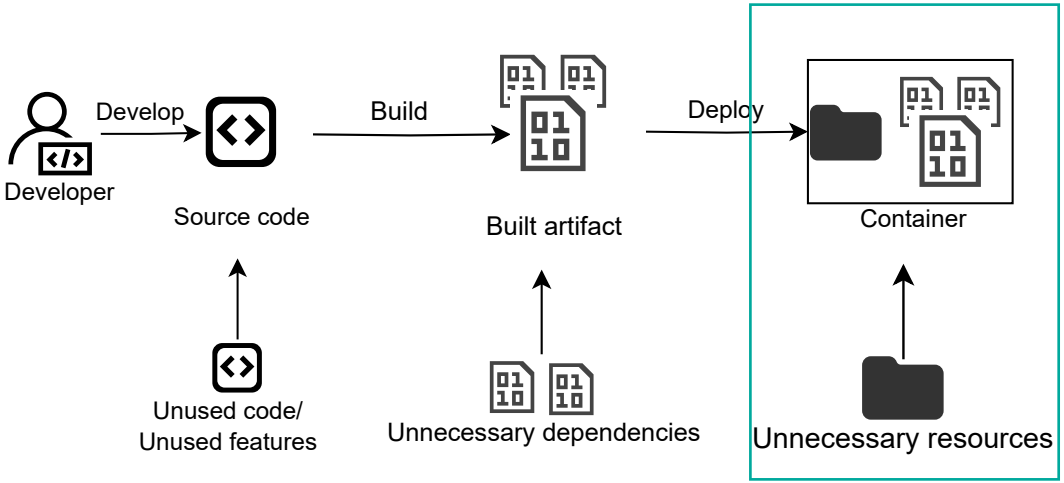
- Each stage of current software development introduces new source of software bloat.

Bloat in Software Development



- Each stage of current software development introduces new source of software bloat.

Bloat in Software Development



- Each stage of current software development introduces new source of software bloat.

Negative Effects of Software Bloat

Negative Effects of Software Bloat



Consumes unnecessary resources

Negative Effects of Software Bloat



Consumes unnecessary resources



Slows down system performance

Negative Effects of Software Bloat



Consumes unnecessary resources

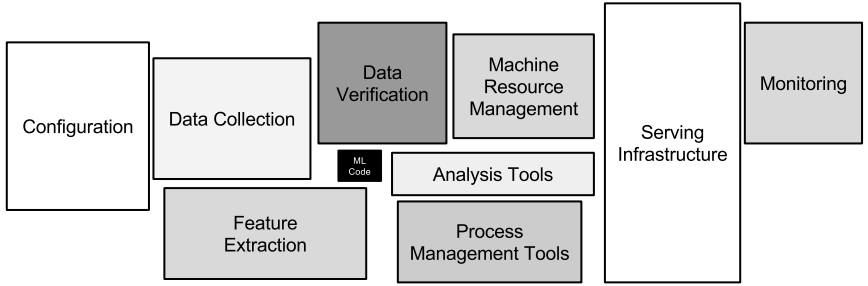


Slows down system performance



Expands the attack surface

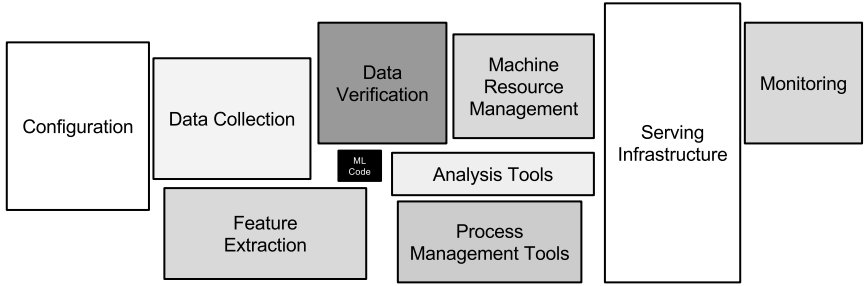
Machine Learning Deployments are Easier to be Bloated



Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Machine Learning Deployments are Easier to be Bloated

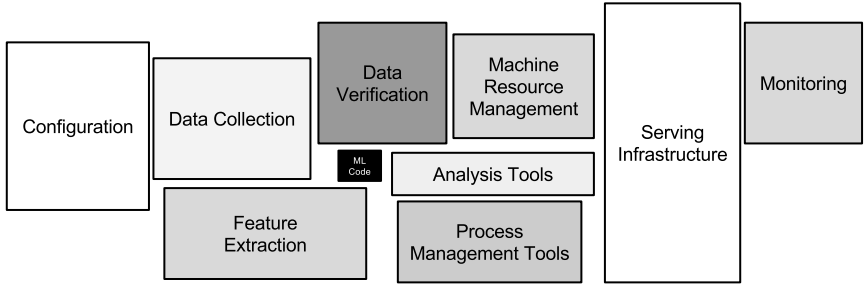
- Machine Learning (ML) systems support the entire process of ML/AI applications, such as ChatGPT, from training to inference.



Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Machine Learning Deployments are Easier to be Bloated

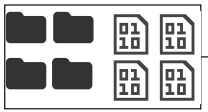
- Machine Learning (ML) systems support the entire process of ML/AI applications, such as ChatGPT, from training to inference.
- ML systems have all the maintenance challenges of traditional software system along with a unique set of ML-specific issues (Sculley et al., 2016).



Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Our Tools

Our Tools



RQ1

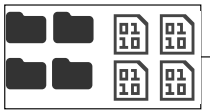
Measure bloat in ML containers

Research Question 1

To what extent are machine learning systems affected by bloat? (Measure bloat in ML containers)

Our Tools

SIGMETRICS'24



RQ1

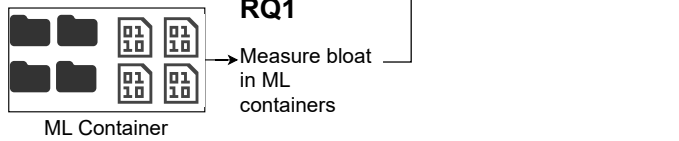
Measure bloat in ML containers

Research Question 1

To what extent are machine learning systems affected by bloat? (Measure bloat in ML containers)

Our Tools

SIGMETRICS'24

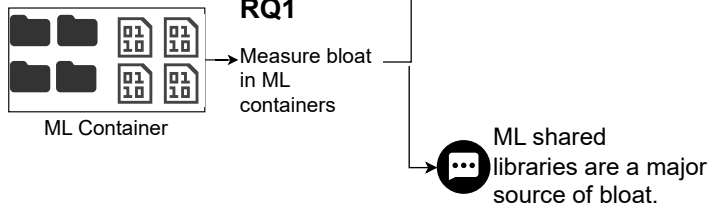


Research Question 1

To what extent are machine learning systems affected by bloat? (Measure bloat in ML containers)

Our Tools

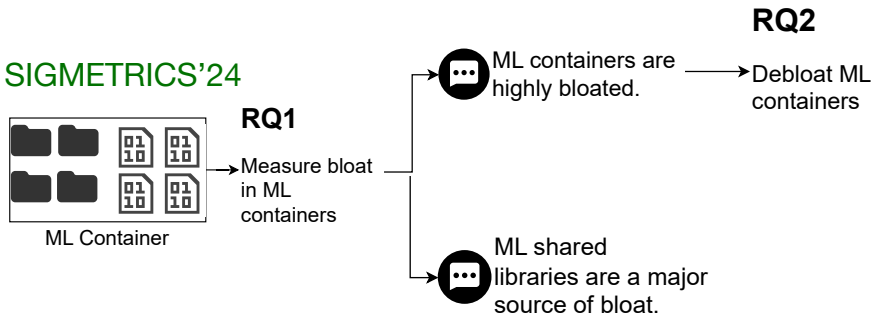
SIGMETRICS'24



Research Question 1

To what extent are machine learning systems affected by bloat? (Measure bloat in ML containers)

Our Tools



Research Question 2

Why do existing debloating approaches fail for ML containers and how can novel approaches be developed to address these limitations? (Debloat ML containers)

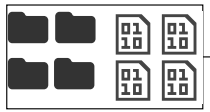
Our Tools

SoCC'25
GitHub: 600+ stars

RQ2

SIGMETRICS'24

RQ1



Measure bloat in ML containers

ML containers are highly bloated. → Debloat ML containers

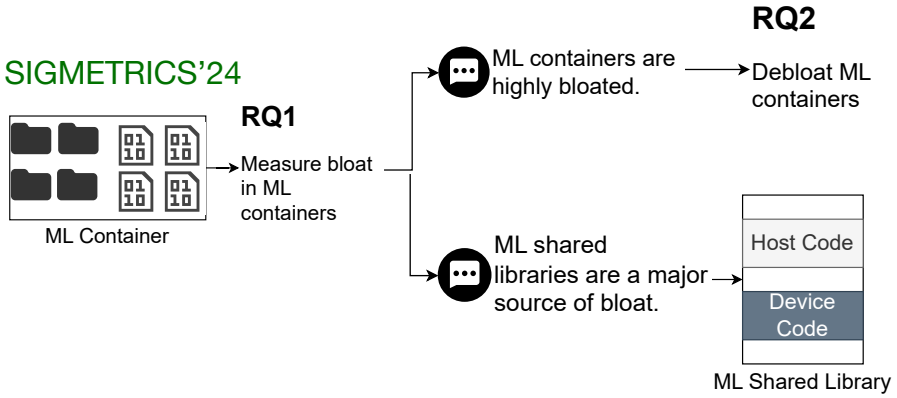
ML shared libraries are a major source of bloat.

Research Question 2

Why do existing debloating approaches fail for ML containers and how can novel approaches be developed to address these limitations? (Debloat ML containers)

Our Tools

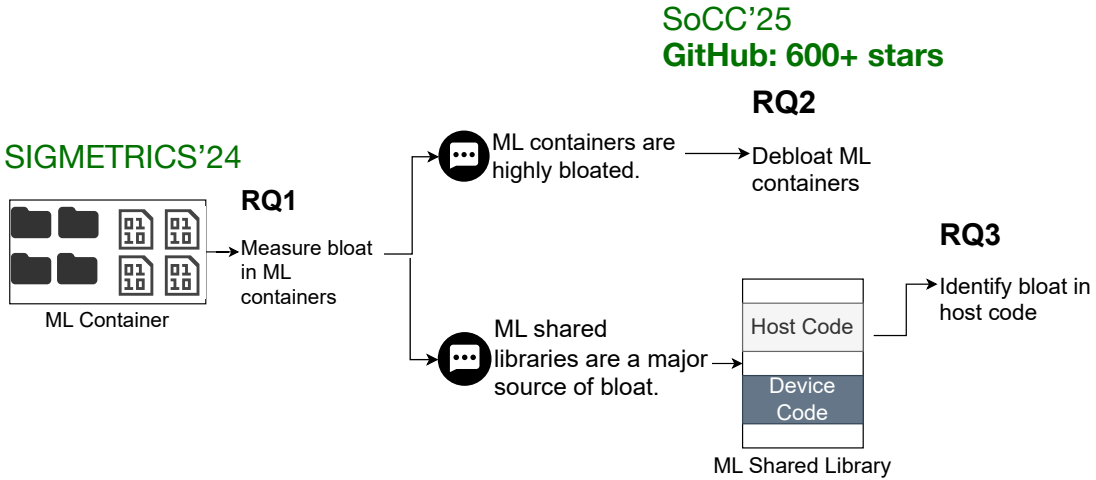
SoCC'25
GitHub: 600+ stars



Research Question 2

Why do existing debloating approaches fail for ML containers and how can novel approaches be developed to address these limitations? (Debloat ML containers)

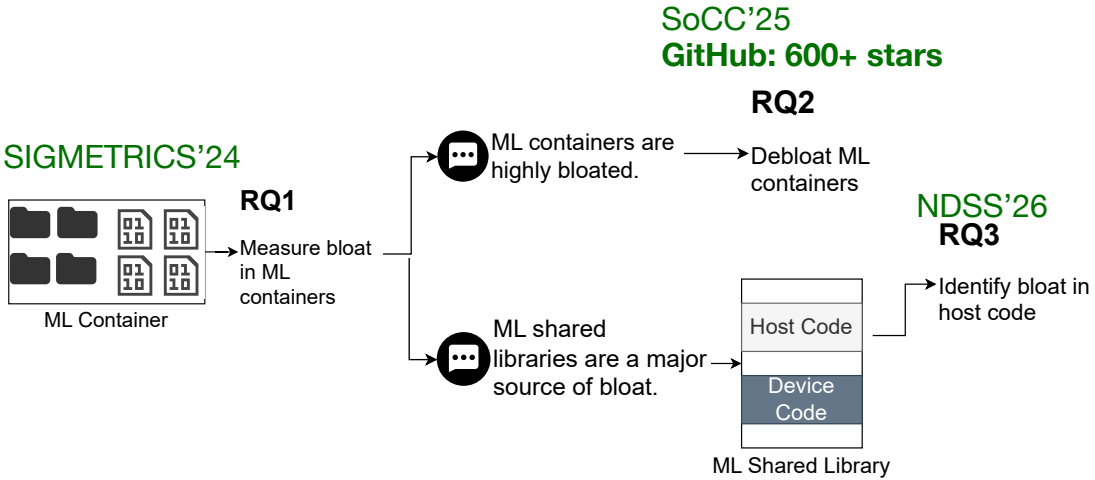
Our Tools



Research Question 3

How can bloat in the host code of ML shared libraries be identified? (Identify bloat in host code)

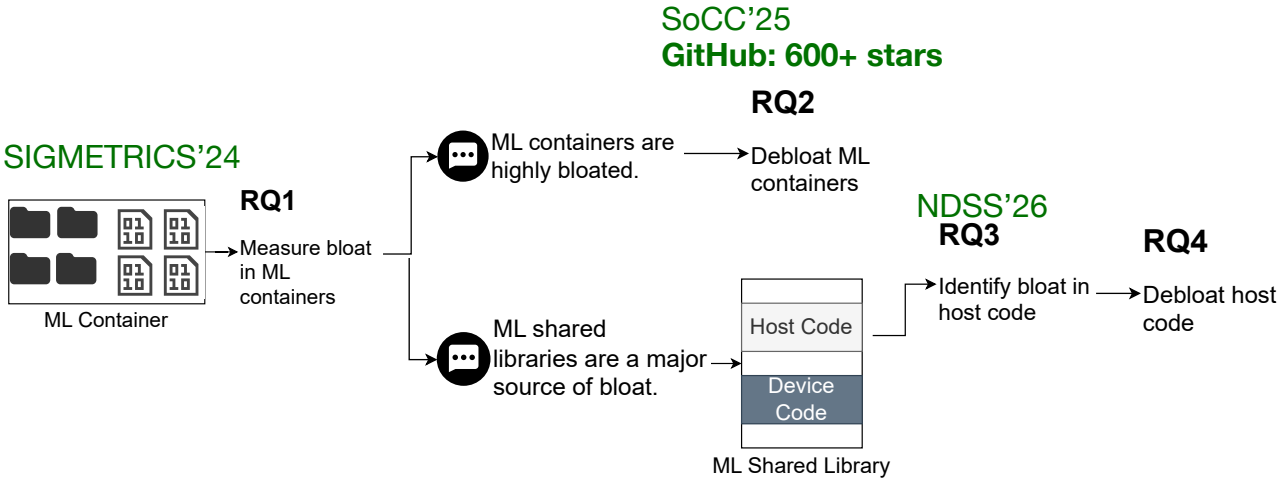
Our Tools



Research Question 3

How can bloat in the host code of ML shared libraries be identified? (Identify bloat in host code)

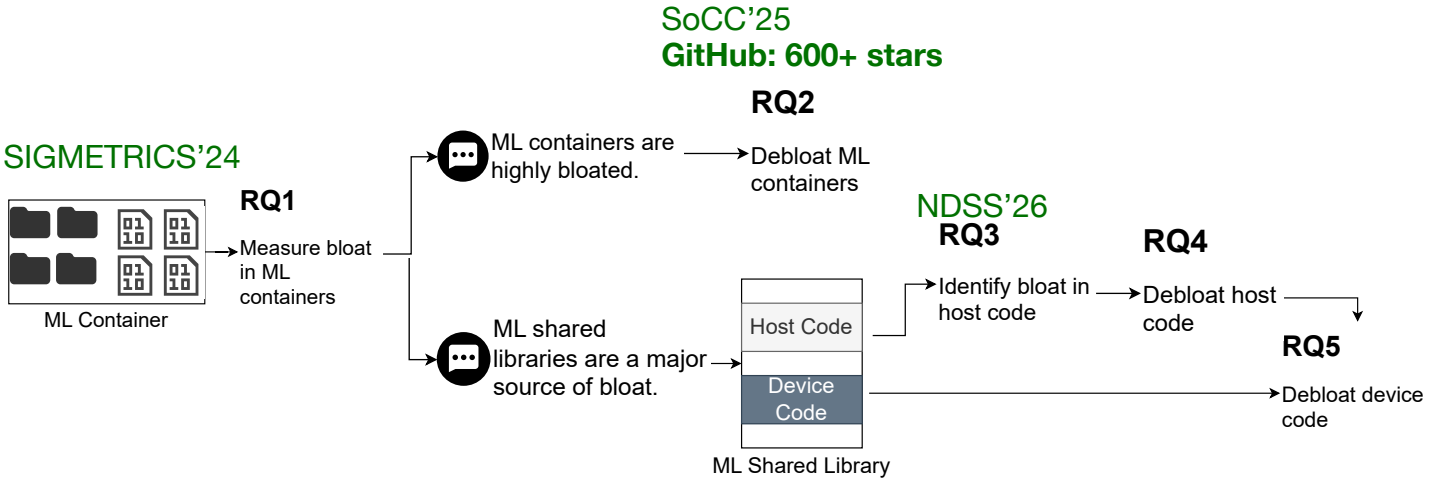
Our Tools



Research Question 4

How can host code in ML shared libraries be debloated? (Debloat host code)

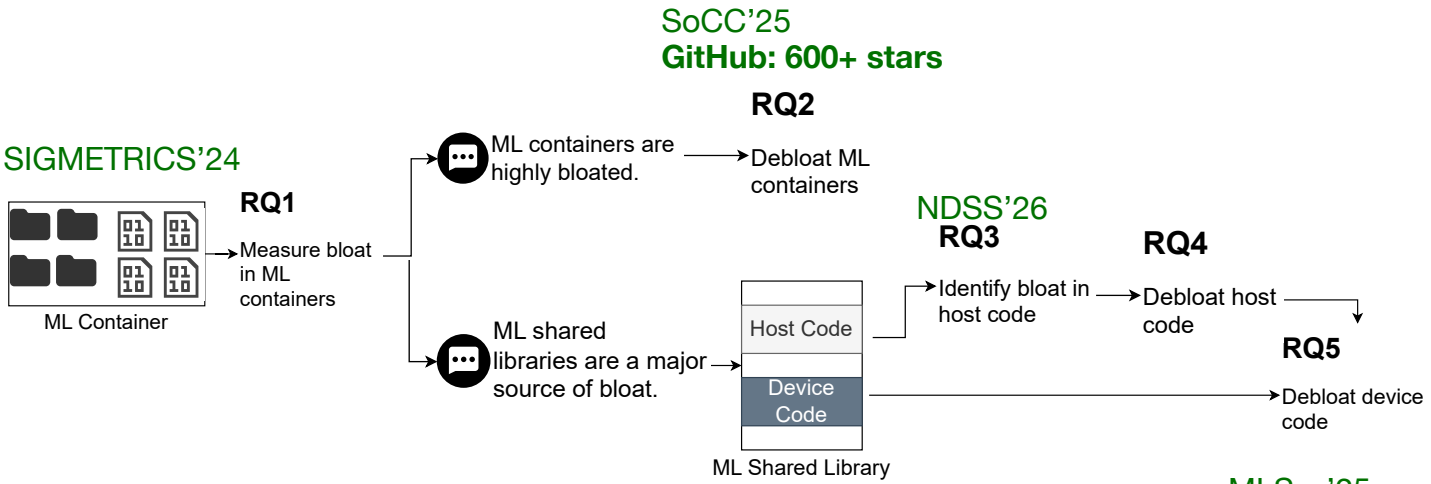
Our Tools



Research Question 5

How can device code in ML shared libraries be debloated? (Debloat device code)

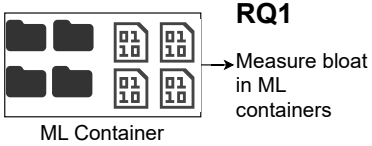
Our Tools



Research Question 5

How can device code in ML shared libraries be debloated? (Debloat device code)

RQ1: Measure Bloat in Machine Learning Containers



- Zhang, H., Alhanahnah, M., Ahmed, F. A., Fatih, D., Leitner, P., & Ali-Eldin, A. (2024). Machine learning systems are bloated and vulnerable. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 8(1), 1-30.

Why Machine Learning Containers?

Why Machine Learning Containers?

- Containers are supposed to be **lightweight virtualization.**

Why Machine Learning Containers?

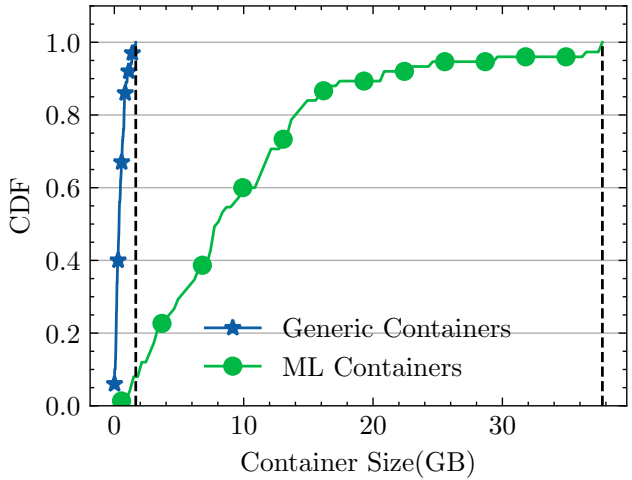
- Containers are supposed to be **lightweight virtualization.**
- ML deployments are the probably now dominating many datacenter

Why Machine Learning Containers?

- Containers are supposed to be **lightweight virtualization.**
- ML deployments are the probably now dominating many datacenter
- For their ease of use, they are the most dominant ML deployment method.

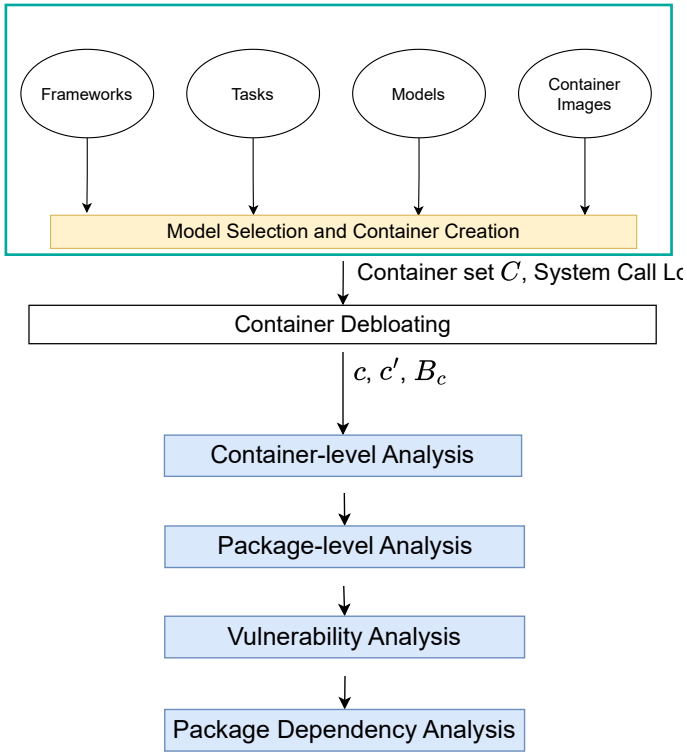
Why Machine Learning Containers?

- Containers are supposed to be **lightweight virtualization.**
- ML deployments are the probably now dominating many datacenter
- For their ease of use, they are the most dominant ML deployment method.
- So how bad is bloat there?



Cumulative distribution function of ML container sizes and generic container sizes.

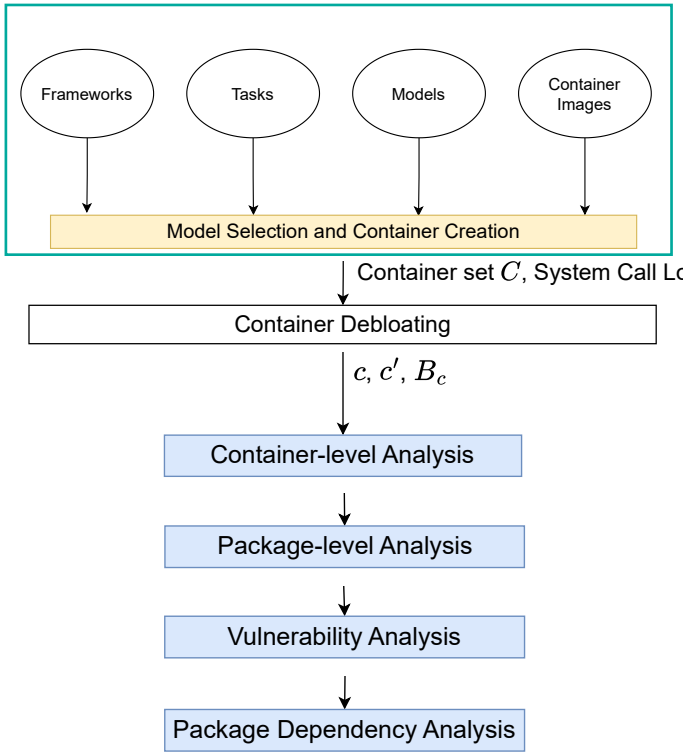
Measure Bloat in ML Containers



Container	Framework	Task	Model	Image	Publisher	Application
c_1	TensorFlow-2.4.0	tuning	Bert	tensorflow/tensorflow:2.4.0-gpu	TF	NLP
c_2	PyTorch-1.12.0	tuning	Bert	pytorch/pytorch:latest	PT	NLP
c_3	PyTorch-1.11.0	tuning	Bert	bert:latest	NGC	NLP
c_4	TensorFlow-2.4.0	tuning	Bert	bert_tf2:latest	NGC	NLP
c_5	PyTorch-1.11.0	serving	Bert	pytorch/torchserve:latest-gpu	PT	NLP
c_6	TensorFlow-2.4.0	serving	Bert	tensorflow/serving:2.4.0-gpu	TF	NLP
c_7	TensorFlow-2.4.0	training	Transformer	tensorflow/tensorflow:2.4.0-gpu	TF	NLP
c_8	PyTorch-1.12.0	training	Transformer	pytorch/pytorch:latest	PT	NLP
c_9	TensorFlow-2.6.0	training	EfficientNet	efficientnet_v2_tf2:latest	NGC	IC
c_{10}	TensorFlow-2.4.0	training	ResNet	tensorflow/tensorflow:2.4.0-gpu	TF	IC
c_{11}	PyTorch-1.12.0	training	ResNet	pytorch/pytorch:latest	PT	IC
c_{12}	PyTorch-1.11.0	serving	ResNet	pytorch/torchserve:latest-gpu	PT	IC
c_{13}	TensorFlow-2.4.0	serving	ResNet	tensorflow/serving:2.4.0-gpu	TF	IC
c_{14}	TensorFlow-2.4.0	training	MaskRCNN	tensorflow/tensorflow:2.4.0-gpu	TF	IS
c_{15}	TensorFlow-2.4.0	training	MaskRCNN	nvidia_mrcnn_tf2:latest	NGC	IS

NLP = Natural Language Processing; IC = Image Classification; IS = Image Segmentation.

Measure Bloat in ML Containers

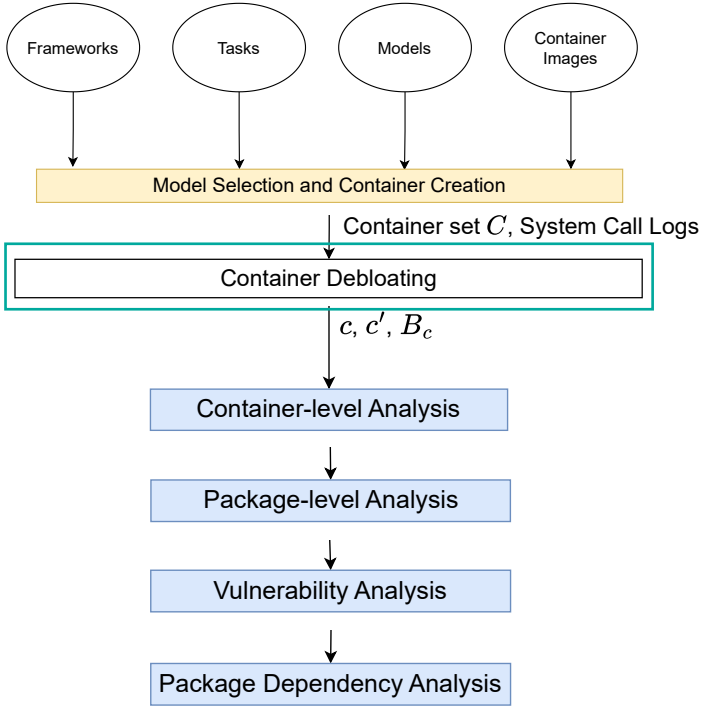


Container	Framework	Task	Model	Image	Publisher	Application
c_1	TensorFlow-2.4.0	tuning	Bert	tensorflow/tensorflow:2.4.0-gpu	TF	NLP
c_2	PyTorch-1.12.0	tuning	Bert	pytorch/pytorch:latest	PT	NLP
c_3	PyTorch-1.11.0	tuning	Bert	bert:latest	NGC	NLP
c_4	TensorFlow-2.4.0	tuning	Bert	bert_tf2:latest	NGC	NLP
c_5	PyTorch-1.11.0	serving	Bert	pytorch/torchserve:latest-gpu	PT	NLP
c_6	TensorFlow-2.4.0	serving	Bert	tensorflow/serving:2.4.0-gpu	TF	NLP
c_7	TensorFlow-2.4.0	training	Transformer	tensorflow/tensorflow:2.4.0-gpu	TF	NLP
c_8	PyTorch-1.12.0	training	Transformer	pytorch/pytorch:latest	PT	NLP
c_9	TensorFlow-2.6.0	training	EfficientNet	efficientnet_v2_tf2:latest	NGC	IC
c_{10}	TensorFlow-2.4.0	training	ResNet	tensorflow/tensorflow:2.4.0-gpu	TF	IC
c_{11}	PyTorch-1.12.0	training	ResNet	pytorch/pytorch:latest	PT	IC
c_{12}	PyTorch-1.11.0	serving	ResNet	pytorch/torchserve:latest-gpu	PT	IC
c_{13}	TensorFlow-2.4.0	serving	ResNet	tensorflow/serving:2.4.0-gpu	TF	IC
c_{14}	TensorFlow-2.4.0	training	MaskRCNN	tensorflow/tensorflow:2.4.0-gpu	TF	IS
c_{15}	TensorFlow-2.4.0	training	MaskRCNN	nvidia_mrcnn_tf2:latest	NGC	IS

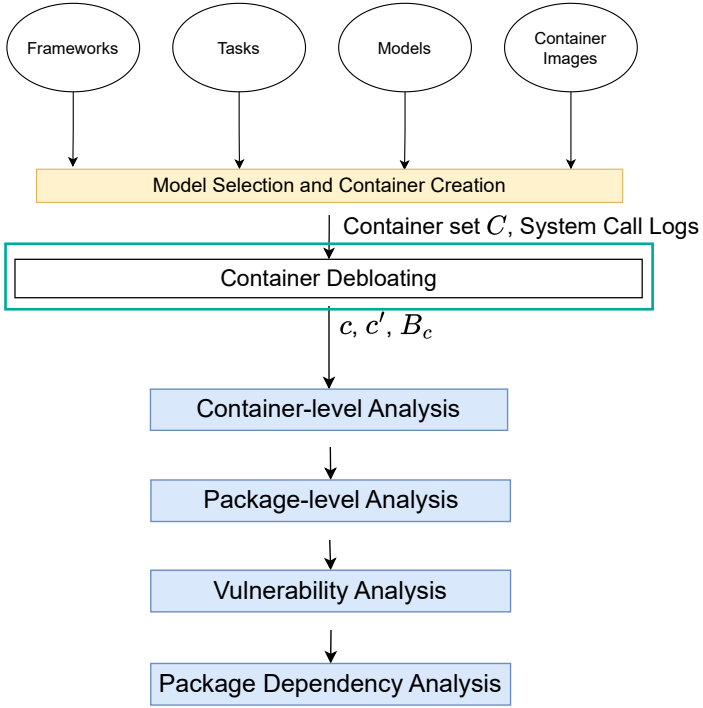
NLP = Natural Language Processing; IC = Image Classification; IS = Image Segmentation.

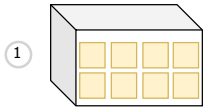
- Tested with ML containers from different **publishers** running different **frameworks**, **tasks**, **models** and **applications**.
- Cover as many aspects of ML as possible.

Measure Bloat in ML Containers

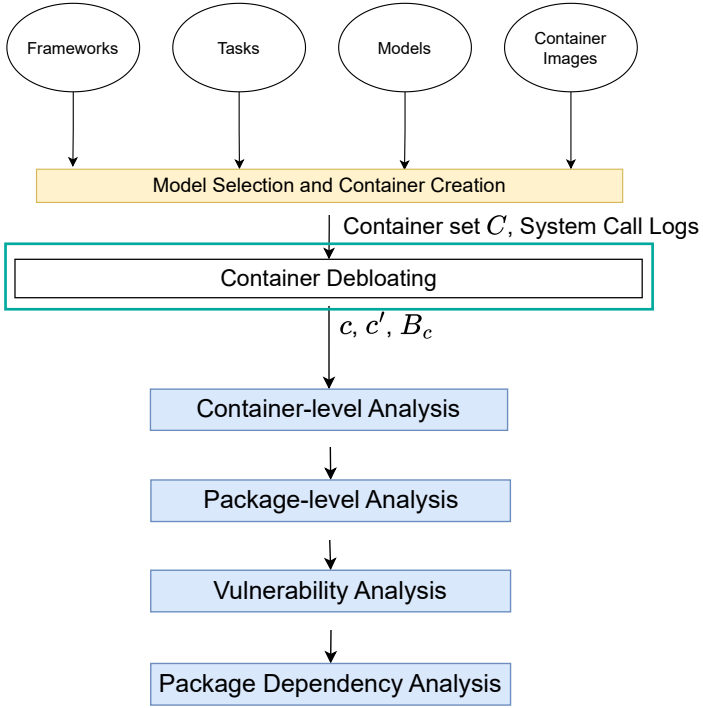


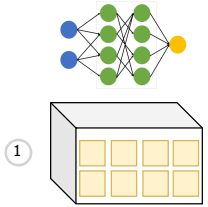
Measure Bloat in ML Containers



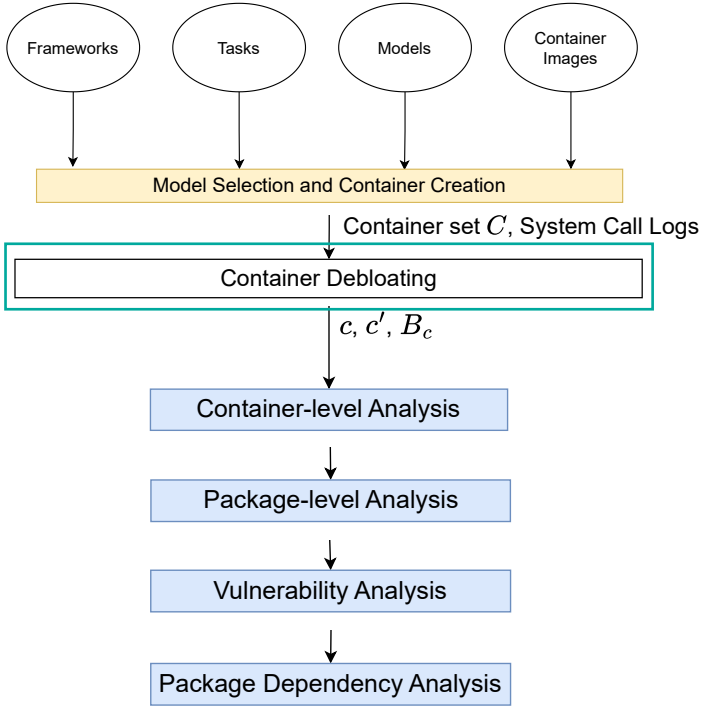


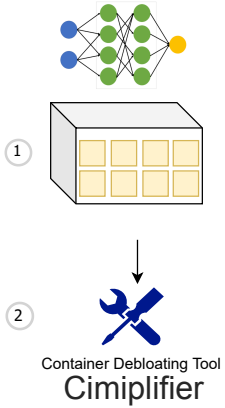
Measure Bloat in ML Containers



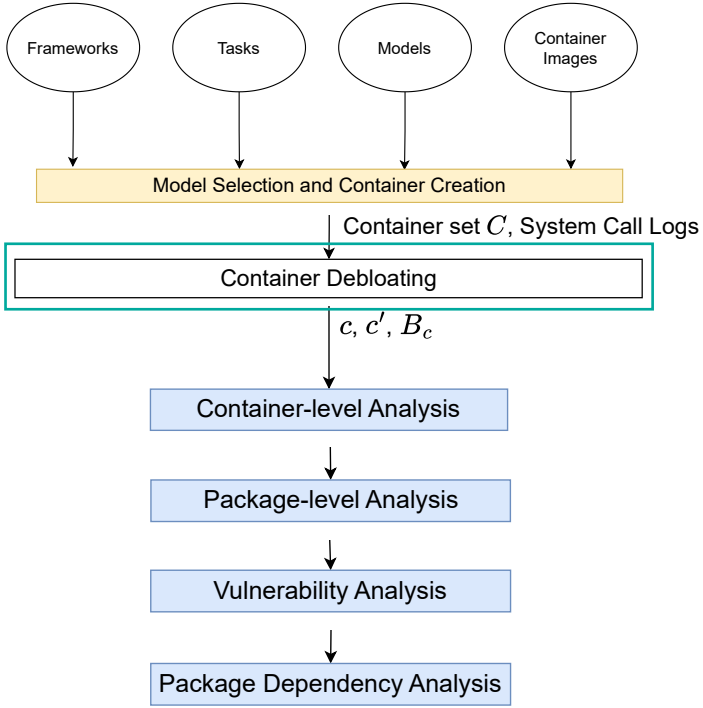


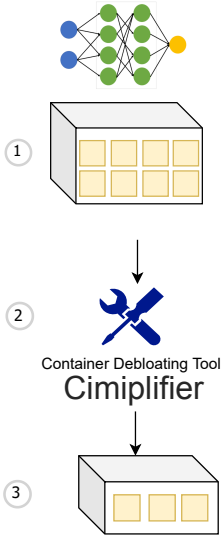
Measure Bloat in ML Containers



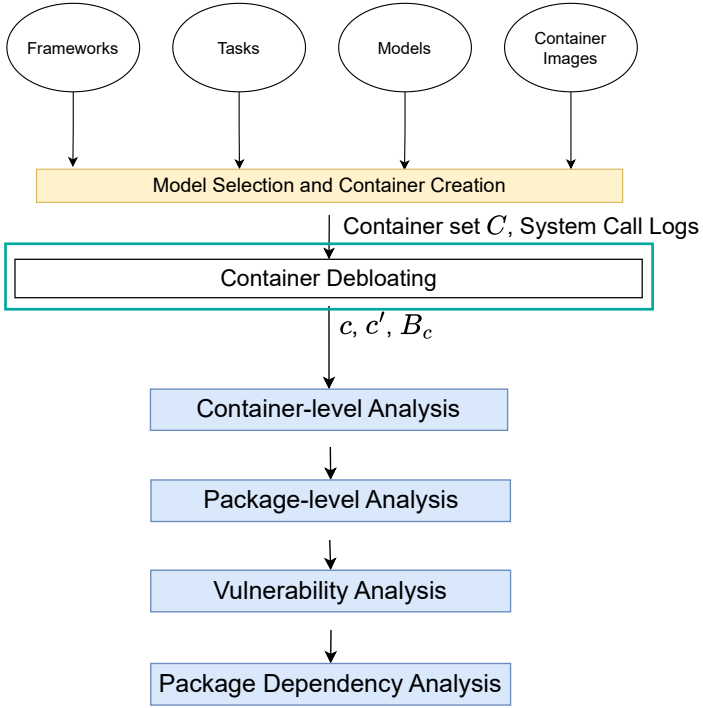


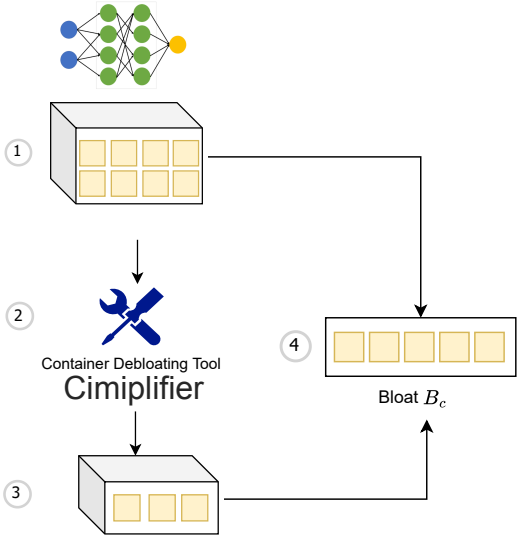
Measure Bloat in ML Containers



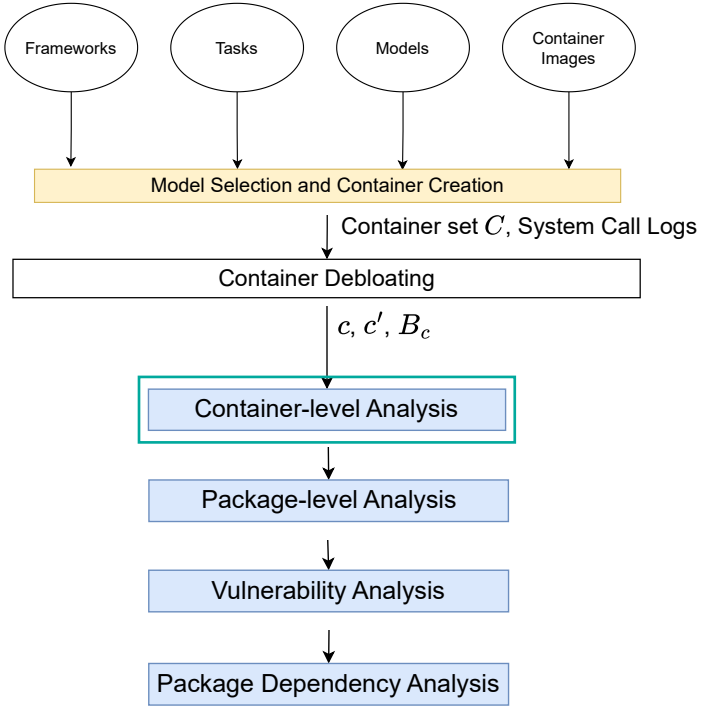


Measure Bloat in ML Containers





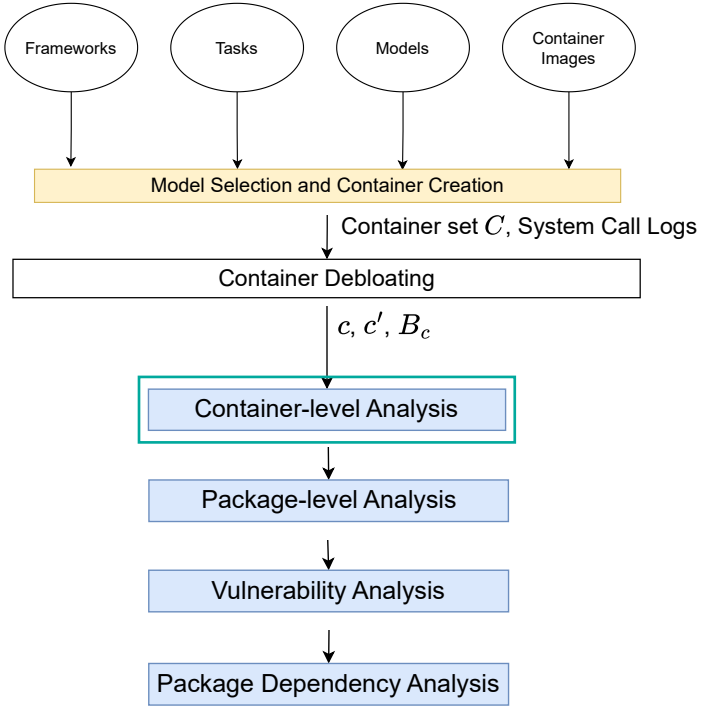
Measure Bloat in ML Containers



Sizes of original and debloated ML containers.

Container c	Original Size	Debloated Size	Reduction
c_1	6.34GB	2.22GB	65%
c_2	8.52GB	1.74GB	**80%**
c_3	15.10GB	3.28GB	78%
c_4	11.30GB	2.74GB	76%
c_5	4.54GB	1.90GB	58%
c_6	8.43GB	2.25GB	73%
c_7	6.34GB	2.21GB	65%
c_8	8.52GB	1.79GB	79%
c_9	12.00GB	5.86GB	51%
c_{10}	6.34GB	3.90GB	39%
c_{11}	8.52GB	1.79GB	79%
c_{12}	4.49GB	1.94GB	57%
c_{13}	8.43GB	3.92GB	53%
c_{14}	6.34GB	4.04GB	**36%**
c_{15}	11.50GB	4.05GB	65%

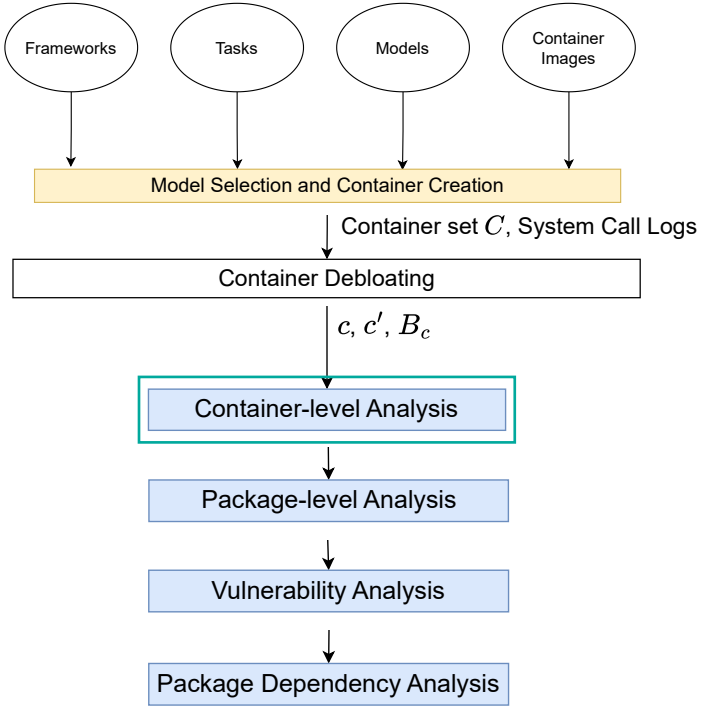
Measure Bloat in ML Containers



Sizes of original and debloated ML containers.

Container c	Original Size	Debloated Size	Reduction
c_1	6.34GB	2.22GB	65%
c_2	8.52GB	1.74GB	**80%**
c_3	15.10GB	3.28GB	78%
c_4	11.30GB	2.74GB	76%
c_5	4.54GB	1.90GB	58%
c_6	8.43GB	2.25GB	73%
c_7	6.34GB	2.21GB	65%
c_8	8.52GB	1.79GB	79%
c_9	12.00GB	5.86GB	51%
c_{10}	6.34GB	3.90GB	39%
c_{11}	8.52GB	1.79GB	79%
c_{12}	4.49GB	1.94GB	57%
c_{13}	8.43GB	3.92GB	53%
c_{14}	6.34GB	4.04GB	**36%**
c_{15}	11.50GB	4.05GB	65%

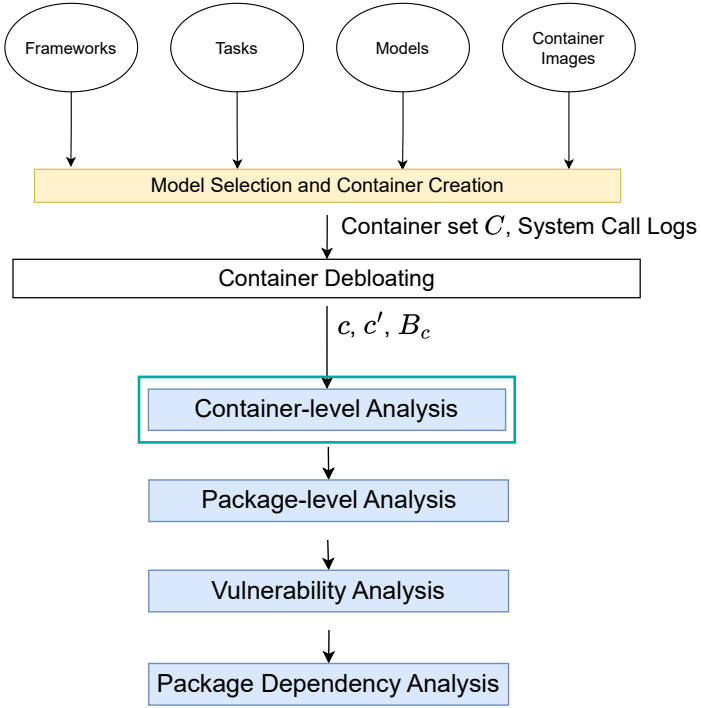
Measure Bloat in ML Containers



Sizes of original and debloated ML containers.

Container c	Original Size	Debloated Size	Reduction
c_1	6.34GB	2.22GB	65%
c_2	8.52GB	1.74GB	**80%**
c_3	15.10GB	3.28GB	78%
c_4	11.30GB	2.74GB	76%
c_5	4.54GB	1.90GB	58%
c_6	8.43GB	2.25GB	73%
c_7	6.34GB	2.21GB	65%
c_8	8.52GB	1.79GB	79%
c_9	12.00GB	5.86GB	51%
c_{10}	6.34GB	3.90GB	39%
c_{11}	8.52GB	1.79GB	79%
c_{12}	4.49GB	1.94GB	57%
c_{13}	8.43GB	3.92GB	53%
c_{14}	6.34GB	4.04GB	**36%**
c_{15}	11.50GB	4.05GB	65%

Measure Bloat in ML Containers

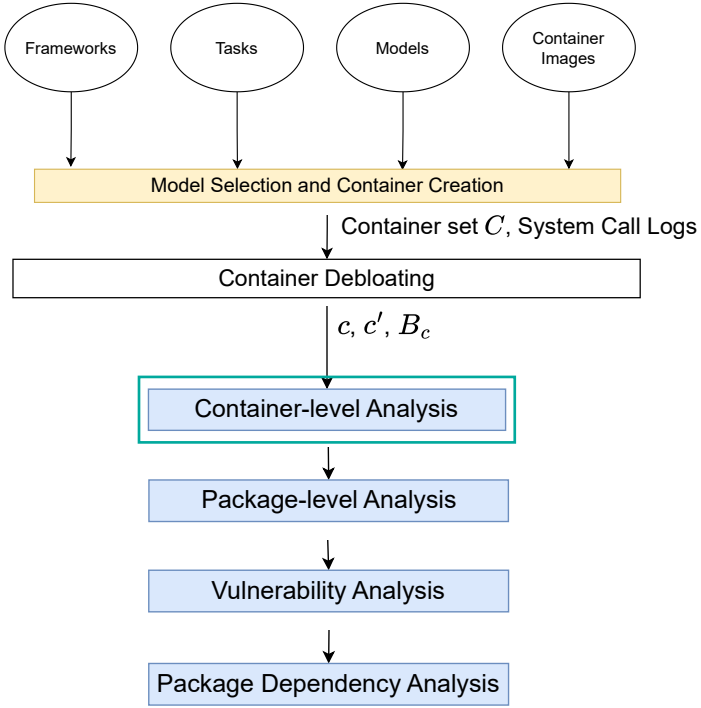


- All these larges files within the debloated ML containers are shared libraries from ML packages.

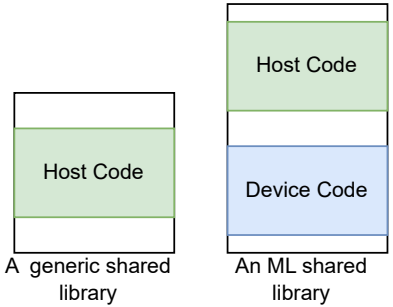
Summary of the large files in debloated ML containers.

File Name	Package Name	Package Manager	Package Functionality	Size(MB)
libtorch_cuda.so	Torch	PIP	ML	1,253
libcudnn_cnn_infer.so	libcudnn	APT	ML	1,133
_pywrap_tensorflow_internal.so	TensorFlow	PIP	ML	1,038
libdali_kernels.so	nvidia-dali-cuda	APT	ML	700
libcusolver.so	libcusolver	APT	ML	565
libtorch_cpu.so	Torch	PIP	ML	469
libdali_operators.so	nvidia-dali-cuda	APT	ML	428
libcudnn_ops_infer.so	libcudnn	APT	ML	417
libcufft.so	libcufft	APT	ML	353
libcublasLt.so	libcublas	APT	ML	314

Measure Bloat in ML Containers

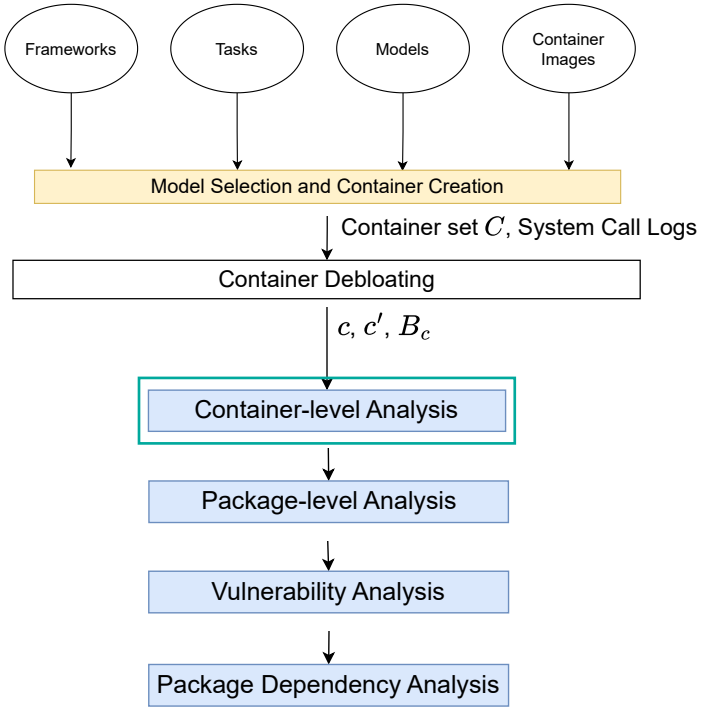


- The uniqueness of ML shared libraries is that they contain both host code and device code.
- Host code runs on CPUs.
- Device code runs on GPUs.

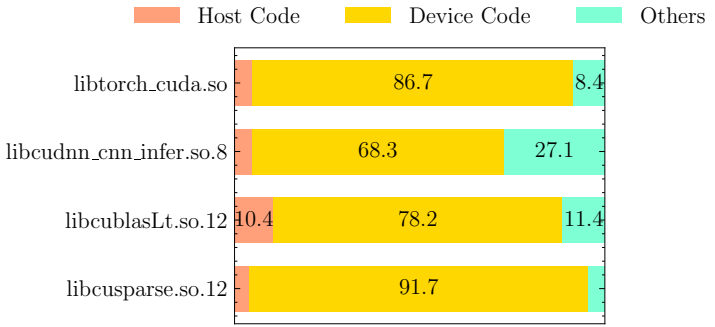


Difference between generic shared library and ML shared library,

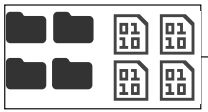
Measure Bloat in ML Containers



- In PyTorch, device code accounts for between 68% to more than 91% of the size of each shared library.
- In contrast, host code represents only a small fraction of these libraries.



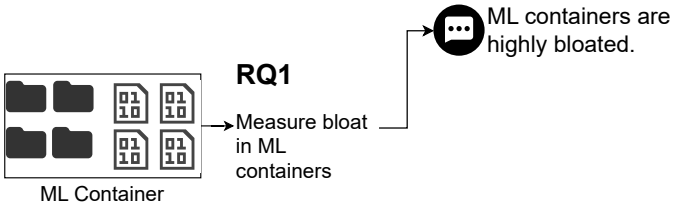
Size distribution of host code and device code in the top 4 largest shared libraries in PyTorch.

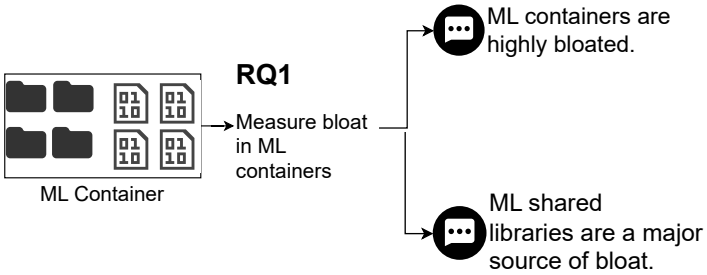


ML Container

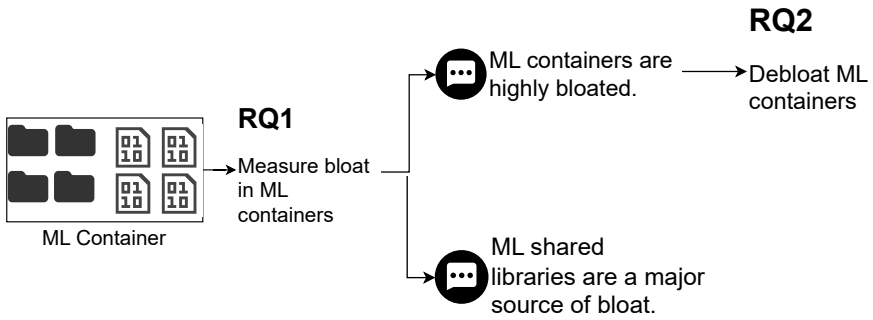
RQ1

Measure bloat in ML containers


ML containers are highly bloated.
RQ1
Measure bloat in ML containers
ML Container


RQ1
Measure bloat in ML containers
ML Container
ML containers are highly bloated.
ML shared libraries are a major source of bloat.

RQ2: Debloatin (Machine Learning) Containers



- Zhang, H., Alhanahnah, M., Ahmed, Leitner, P., & Ali-Eldin, A. (2025). BLAFS: A Bloat Aware Container File System. Proceedings of the 16th ACM Symposium on Cloud Computing (SoCC).

Rethinking Cause of Container Bloat

Rethinking Cause of Container Bloat

- A container has a layered filesystem.

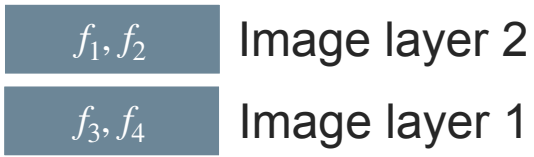
Rethinking Cause of Container Bloat

- A container has a layered filesystem.



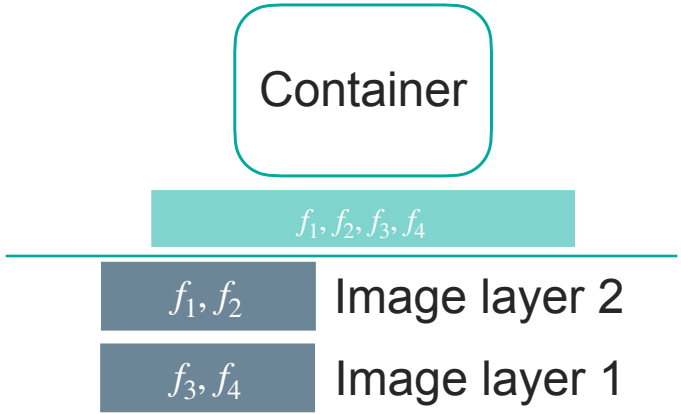
Rethinking Cause of Container Bloat

- A container has a layered filesystem.



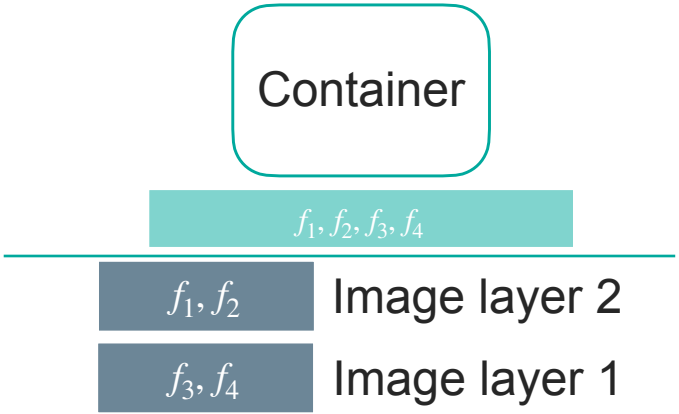
Rethinking Cause of Container Bloat

- A container has a layered filesystem.



Rethinking Cause of Container Bloat

- A container has a layered filesystem.
- This layered filesystem architecture is also a major cause of bloat.



How Existing Container Debloaters Work?

How Existing Container Debloaters Work?

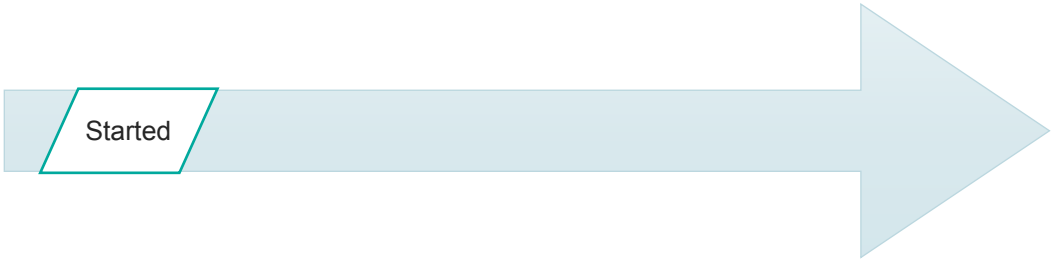
SlimToolkit
Cimiplifier

How Existing Container Debloaters Work?

SlimToolkit
Cimiplifier

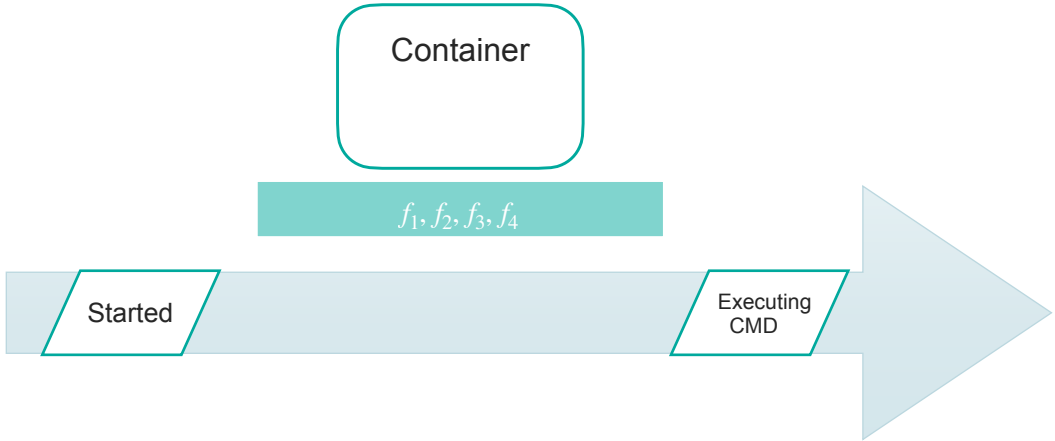
How Existing Container Debloaters Work?

SlimToolkit
Cimiplifier

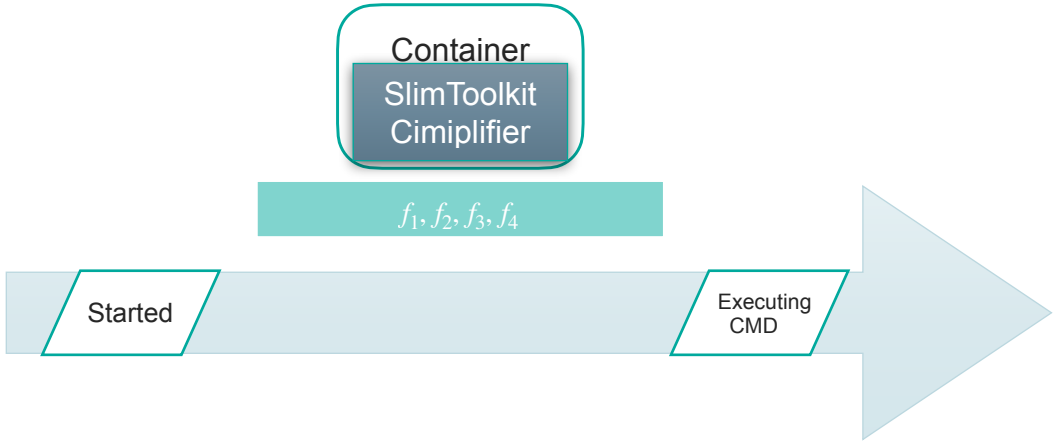


How Existing Container Debloaters Work?

SlimToolkit
Cimiplifier



How Existing Container Debloaters Work?



Limitations of Existing Container Debloaters

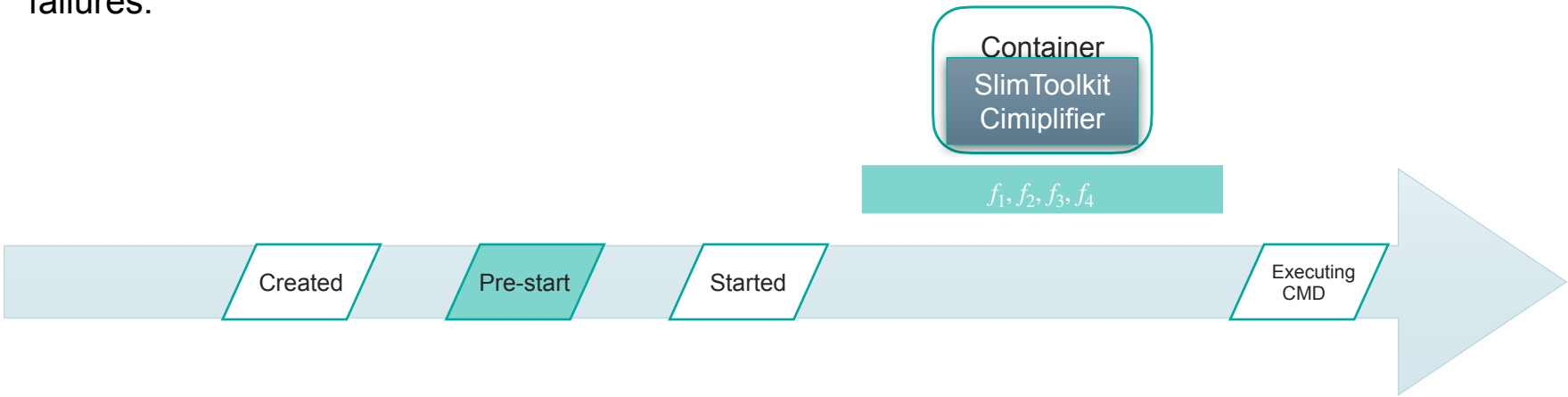
Limitations of Existing Container Debloaters

- **Layer structure broken**: Both tools break container layers, losing layer-sharing benefits — sometimes making debloated containers larger than the originals.



Limitations of Existing Container Debloaters

- **Layer structure broken**: Both tools break container layers, losing layer-sharing benefits — sometimes making debloated containers larger than the originals.
- **Startup phase issue**: Debloaters start tracing only after container starts, missing files accessed during startup (e.g., CUDA libraries in ML containers), which can cause startup failures.



The Cure is in the Cause

The Cure is in the Cause

- BLAFS: A Bloat-Aware Container File System

The Cure is in the Cause

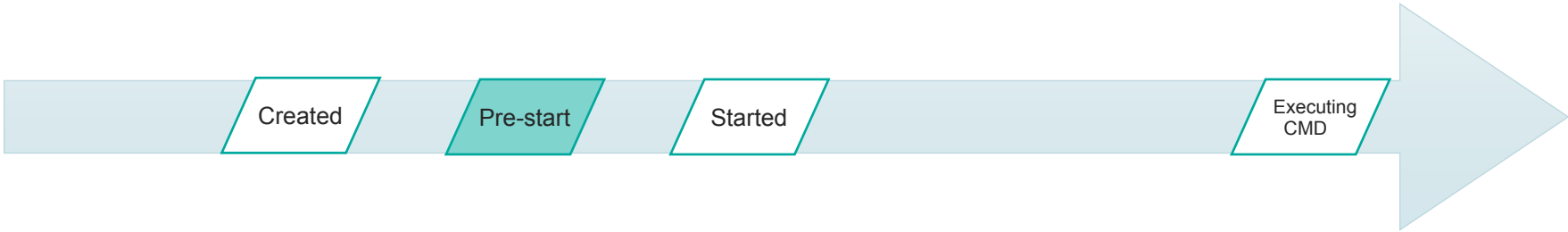
- BLAFS: A Bloat-Aware Container File System
- The filesystem causes the bloat; BLAFS debloats at the filesystem level.

The Cure is in the Cause

- BLAFS: A Bloat-Aware Container File System
- The filesystem causes the bloat; BLAFS debloats at the filesystem level.
- The layered structure causes the bloat; BLAFS reuses the layered structure to debloat.

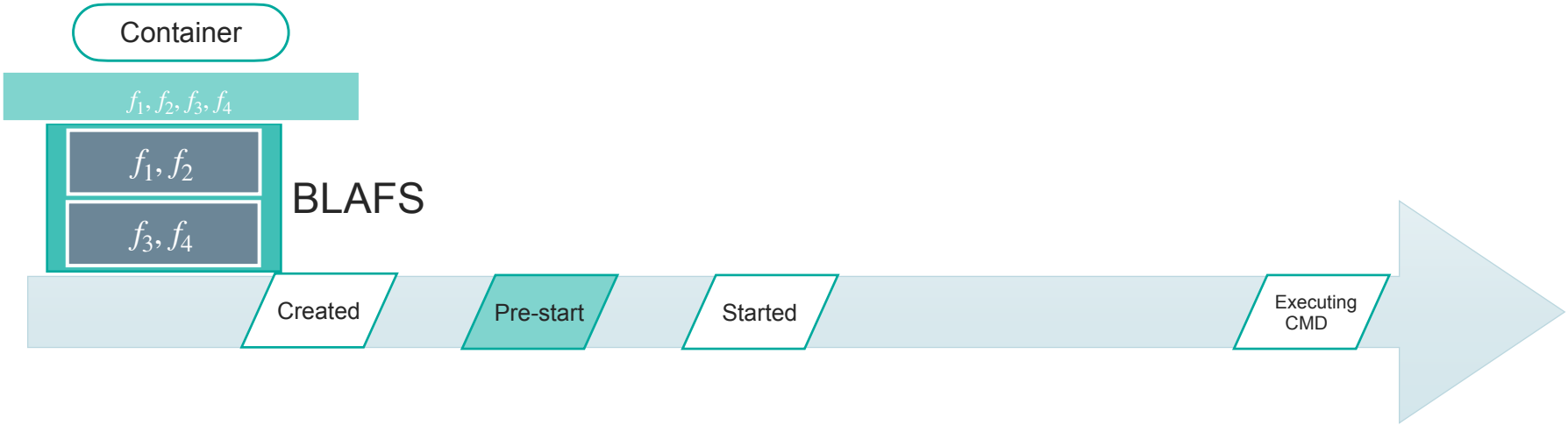
The Cure is in the Cause

- BLAFS: A Bloat-Aware Container File System
- The filesystem causes the bloat; BLAFS debloats at the filesystem level.
- The layered structure causes the bloat; BLAFS reuses the layered structure to debloat.



The Cure is in the Cause

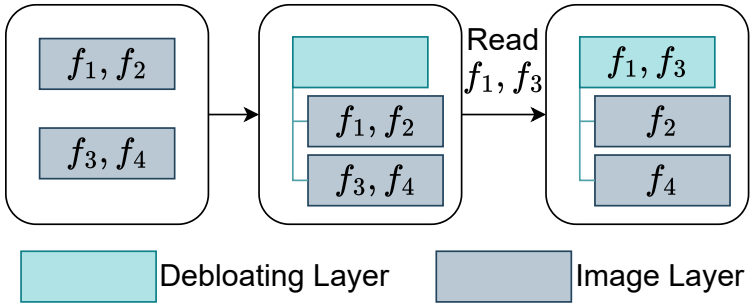
- BLAFS: A Bloat-Aware Container File System
- The filesystem causes the bloat; BLAFS debloats at the filesystem level.
- The layered structure causes the bloat; BLAFS reuses the layered structure to debloat.



BLAFS: Debloating Layer

BLAFS: Debloating Layer

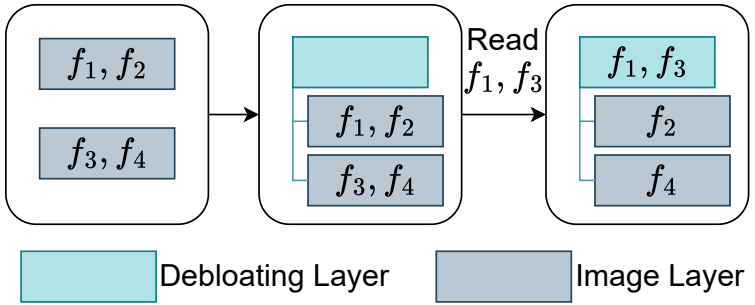
- BLAFS introduces a new layer, the debloating layer, to the original container filesystem.



An example of the debloating layer.

BLAFS: Debloating Layer

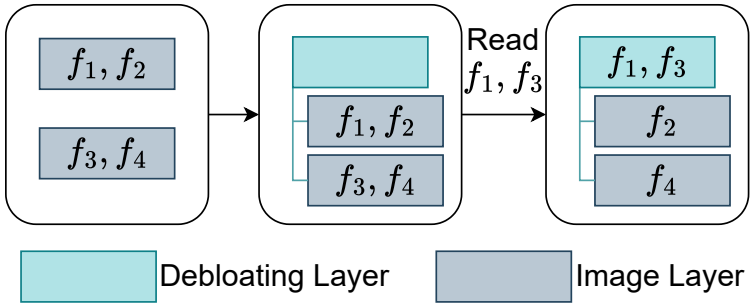
- BLAFS introduces a new layer, the debloating layer, to the original container filesystem.
- A debloating layer D has a list of image layers as its child layers (D.children).



An example of the debloating layer.

BLAFS: Debloating Layer

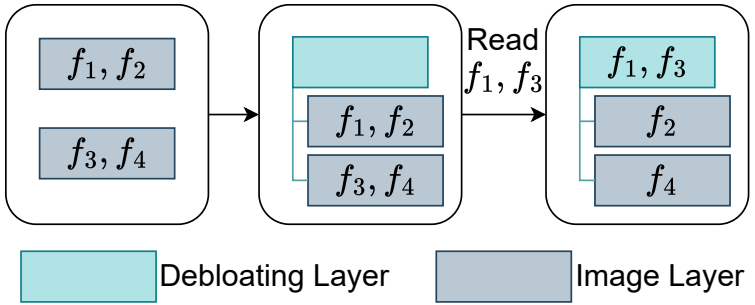
- BLAFS introduces a new layer, the debloating layer, to the original container filesystem.
- A debloating layer D has a list of image layers as its child layers (D.children).
- When reading a file from a debloating layer, the file is moved from the child layers to the debloating layer.



An example of the debloating layer.

BLAFS: Debloating Layer

- BLAFS introduces a new layer, the debloating layer, to the original container filesystem.
- A debloating layer D has a list of image layers as its child layers (D.children).
- When reading a file from a debloating layer, the file is moved from the child layers to the debloating layer.
- In doing so, the debloating layer only includes used files.

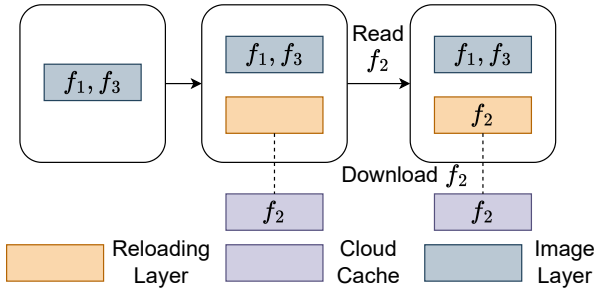


An example of the debloating layer.

BLAFS: Reloading Layer

BLAFS: Reloading Layer

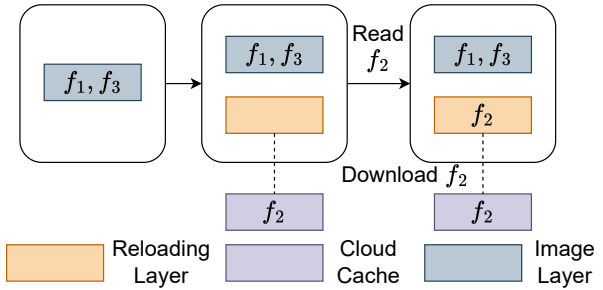
- BLAFS also has an **optional** reloading layer, fetching files accessed but missing from a debloated container.



An example of the reloading layer.

BLAFS: Reloading Layer

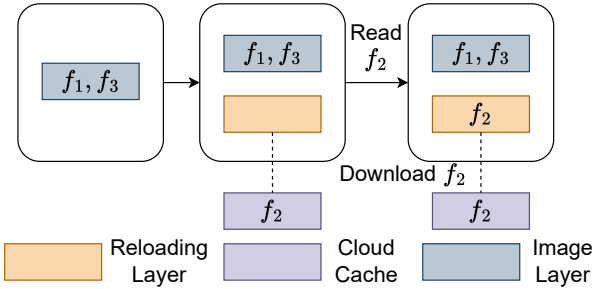
- BLAFS also has an **optional** reloading layer, fetching files accessed but missing from a debloated container.
- A reloading layer is appended to the **bottom** of the image layers.



An example of the reloading layer.

BLAFS: Reloading Layer

- BLAFS also has an **optional** reloading layer, fetching files accessed but missing from a debloated container.
- A reloading layer is appended to the **bottom** of the image layers.
- If a file request reaches the reloading layer, it indicates that the file was not found in any of the preceding layers and can be reloaded on demand.



An example of the reloading layer.

Flexibility: No-sharing mode and Fully-sharing mode

Flexibility: No-sharing mode and Fully-sharing mode

- BLAFS provides flexible design: debloating layer and reloading layer can be organized to suit different use cases.

Flexibility: No-sharing mode and Fully-sharing mode

- BLAFS provides flexible design: debloating layer and reloading layer can be organized to suit different use cases.
- We provide four modes based on how layers are organized:

Flexibility: No-sharing mode and Fully-sharing mode

- BLAFS provides flexible design: debloating layer and reloading layer can be organized to suit different use cases.
- We provide four modes based on how layers are organized:
  - No-sharing mode: minimize per-container size.

Flexibility: No-sharing mode and Fully-sharing mode

- BLAFS provides flexible design: debloating layer and reloading layer can be organized to suit different use cases.
- We provide four modes based on how layers are organized:
  - No-sharing mode: minimize per-container size.
  - Fully-sharing mode: minimize aggregate size across multiple containers.

Flexibility: No-sharing mode and Fully-sharing mode

- BLAFS provides flexible design: debloating layer and reloading layer can be organized to suit different use cases.
- We provide four modes based on how layers are organized:
  - No-sharing mode: minimize per-container size.
  - Fully-sharing mode: minimize aggregate size across multiple containers.
  - Security-hardened mode: restrict file access to debloated container files only.

Flexibility: No-sharing mode and Fully-sharing mode

- BLAFS provides flexible design: debloating layer and reloading layer can be organized to suit different use cases.
- We provide four modes based on how layers are organized:
  - No-sharing mode: minimize per-container size.
  - Fully-sharing mode: minimize aggregate size across multiple containers.
  - Security-hardened mode: restrict file access to debloated container files only.
  - Dynamic deployment mode: reload files on demand during runtime.

Evaluation on Effectiveness

Evaluation on Effectiveness

- BLAFS has been evaluated on more than 30 containers, including generic containers, serverless function containers and machine learning containers.

Evaluation on Effectiveness

- BLAFS has been evaluated on more than 30 containers, including generic containers, serverless function containers and machine learning containers.
- BLAFS successfully debloated all containers, while the two state-of-the-art debloaters failed on more than 50% of them.

Evaluation for Size Reduction

Debloating results for the 22 containers sorted by the reduction percentage.

Container	Original (MB)	Debloated (MB)	Reduction
httpd:2.4	141	7	95%
nginx:1.27.2	183	12	93%
memcached:1.6.32	81	9	89%
pytorch-train	16,822	2,902	83%
tensorflow-train	10,952	1,695	85%
mysql:9.1	574	99	83%
postgres:17	415	85	79%
ghost:5.101.3	547	121	78%
redis:7.4.1	112	27	75%
haproxy:3.0.6	98	27	72%
mongo:8.0	815	233	71%
solr:9.7.0	561	195	65%
rabbitmq:4.0	209	73	65%
maven:3.9.9	505	195	61%
elasticsearch:8.16.0	1,241	479	61%
eclipse-mosquitto:2.0.20	14	7	51%
telegraf:1.30	435	223	49%
nextcloud:28.0.12	1,200	761	37%
sonarqube:9.9.7	576	428	26%
registry:2.8.3	24	18	25%
consul:1.15.4	148	137	7%
traefik:v3.2.0	176	169	4%

Evaluation for Size Reduction

- Over half of the containers achieved a size reduction greater than 60%.

Debloating results for the 22 containers sorted by the reduction percentage.

Container	Original (MB)	Debloated (MB)	Reduction
httpd:2.4	141	7	95%
nginx:1.27.2	183	12	93%
memcached:1.6.32	81	9	89%
pytorch-train	16,822	2,902	83%
tensorflow-train	10,952	1,695	85%
mysql:9.1	574	99	83%
postgres:17	415	85	79%
ghost:5.101.3	547	121	78%
redis:7.4.1	112	27	75%
haproxy:3.0.6	98	27	72%
mongo:8.0	815	233	71%
solr:9.7.0	561	195	65%
rabbitmq:4.0	209	73	65%
maven:3.9.9	505	195	61%
elasticsearch:8.16.0	1,241	479	61%
eclipse-mosquitto:2.0.20	14	7	51%
telegraf:1.30	435	223	49%
nextcloud:28.0.12	1,200	761	37%
sonarqube:9.9.7	576	428	26%
registry:2.8.3	24	18	25%
consul:1.15.4	148	137	7%
traefik:v3.2.0	176	169	4%

Evaluation with Lazy-loading Snapshotters

Conversion time reduction after debloating for eStargz and Starlight.

Function	eStargz	Starlight
dynamic-html	73s (79%)	114s (93%)
uploader	67s (77%)	113s (93%)
thumbnailer	68s (77%)	114s (93%)
video-proc.	70s (70%)	115s (92%)
compression	67s (77%)	114s (93%)
image-recog.	111s (28%)	117s (45%)
graph-page.	68s (77%)	113s (93%)
graph-mst	63s (76%)	112s (93%)
graph-bfs	65s (76%)	114s (93%)

Evaluation with Lazy-loading Snapshotters

- Conversion time reduction: 28%~79% for eStargz, 45%~93% for Starlight.

Conversion time reduction after debloating for eStargz and Starlight.

Function	eStargz	Starlight
dynamic-html	73s (79%)	114s (93%)
uploader	67s (77%)	113s (93%)
thumbnailer	68s (77%)	114s (93%)
video-proc.	70s (70%)	115s (92%)
compression	67s (77%)	114s (93%)
image-recog.	111s (28%)	117s (45%)
graph-page.	68s (77%)	113s (93%)
graph-mst	63s (76%)	112s (93%)
graph-bfs	65s (76%)	114s (93%)

Evaluation for Serverless Computing

Sizes and latencies of serverless functions on OpenWhisk.

Function	Size/MB		Latency/ms	
	Ori.	Deb.	Ori.	Deb.
dynamic-html	1,085 (96%)	49	18,192 (68%)	5,839
uploader	1,084 (95%)	49	18,682 (68%)	6,029
thumbnailer	1,099 (95%)	55	18,854 (68%)	6,017
video-proc.	1,339 (91%)	125	21,737 (67%)	7,244
compression	1,084 (96%)	49	19,206 (67%)	6,335
image-recog.	1,802 (66%)	619	28,283 (46%)	15,392
graph-page.	1,093 (95%)	60	18,619 (62%)	7,158
graph-mst	1,093 (95%)	60	18,869 (67%)	6,162
graph-bfs	1,093 (95%)	60	18,682 (68%)	5,975

Evaluation for Serverless Computing

- Size reduction: 66%~96%

Sizes and latencies of serverless functions on OpenWhisk.

Function	Size/MB		Latency/ms	
	Ori.	Deb.	Ori.	Deb.
dynamic-html	1,085 (96%)	49	18,192 (68%)	5,839
uploader	1,084 (95%)	49	18,682 (68%)	6,029
thumbnailer	1,099 (95%)	55	18,854 (68%)	6,017
video-proc.	1,339 (91%)	125	21,737 (67%)	7,244
compression	1,084 (96%)	49	19,206 (67%)	6,335
image-recog.	1,802 (66%)	619	28,283 (46%)	15,392
graph-page.	1,093 (95%)	60	18,619 (62%)	7,158
graph-mst	1,093 (95%)	60	18,869 (67%)	6,162
graph-bfs	1,093 (95%)	60	18,682 (68%)	5,975

Evaluation for Serverless Computing

- Size reduction: 66%~96%
- Latency reduction: 46%~68%

Sizes and latencies of serverless functions on OpenWhisk.

Function	Size/MB		Latency/ms	
	Ori.	Deb.	Ori.	Deb.
dynamic-html	1,085 (96%)	49	18,192 (68%)	5,839
uploader	1,084 (95%)	49	18,682 (68%)	6,029
thumbnailer	1,099 (95%)	55	18,854 (68%)	6,017
video-proc.	1,339 (91%)	125	21,737 (67%)	7,244
compression	1,084 (96%)	49	19,206 (67%)	6,335
image-recog.	1,802 (66%)	619	28,283 (46%)	15,392
graph-page.	1,093 (95%)	60	18,619 (62%)	7,158
graph-mst	1,093 (95%)	60	18,869 (67%)	6,162
graph-bfs	1,093 (95%)	60	18,682 (68%)	5,975

Evaluation on CVE Reduction

CVE Reduction.

Container	Critical	High	Medium	Low	Negligible	Total	Reduction
mysql:8.0.23	26 (3)	71 (7)	44 (5)	34 (3)	90 (11)	265(29)	89%
redis:6.2.1	20 (2)	67 (12)	32 (6)	25 (3)	60 (3)	204 (26)	87%
ghost:3.42.5-alpine	14 (12)	129 (108)	67 (46)	7 (7)	63 (2)	217 (173)	20%
registry:2.7.0	4 (3)	46 (37)	9 (7)	0 (0)	0 (0)	59(43)	27%
golang:1.16.2	63 (2)	376 (19)	314 (5)	80 (3)	499 (5)	1332(34)	97%
python:3.9.3	145 (15)	863 (47)	1048 (34)	418 (3)	898 (10)	3372 (109)	20%
bert_tf2:latest*	38 (26)	358 (256)	1426 (452)	549 (210)	57 (3)	2428 (947)	61%
nvidia_mrcnn_tf2:latest*	38 (26)	358 (225)	1445 (451)	558 (210)	57 (3)	2456 (945)	62%
merlin-pytorch-training:22.04*	47(46)	166(127)	902(134)	317(15)	70(3)	1502(325)	78%
merlin-tensorflow-training:22.04*	15(14)	109(76)	939(222)	304(34)	56(3)	1423(349)	75%

* These machine learning containers are collected from NVIDIA NGC services [37].

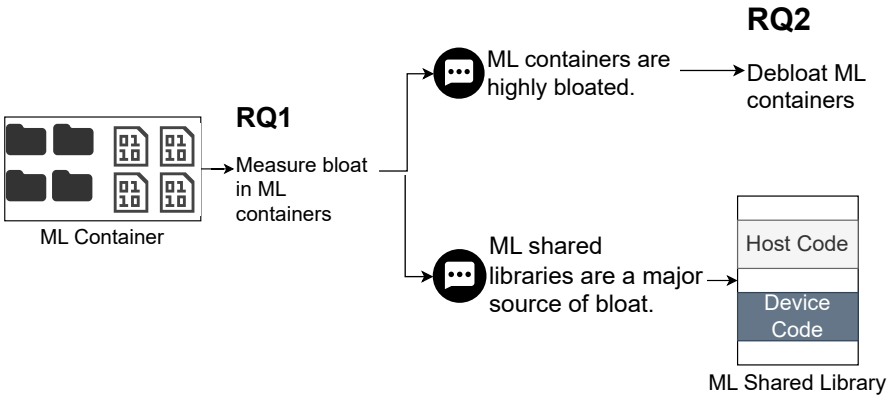
Evaluation on CVE Reduction

- BLAFS reduces the number of CVEs significantly, with reductions ranging from 20% to 97%.

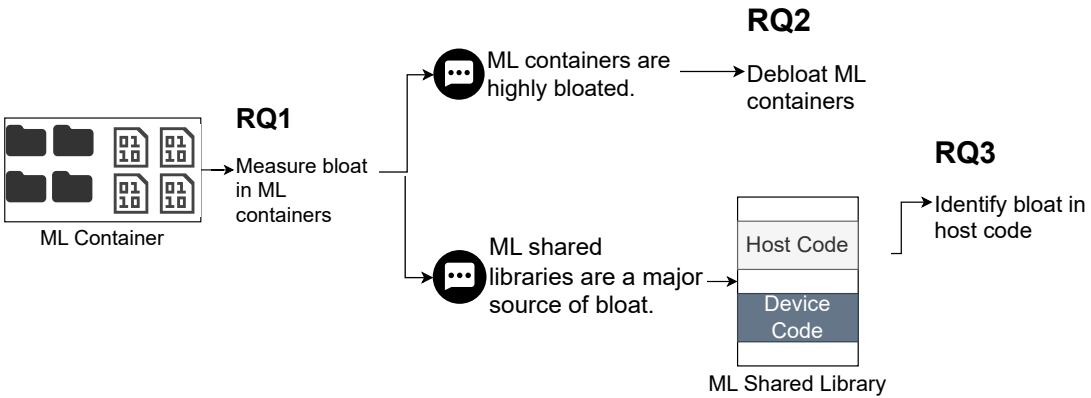
CVE Reduction.

Container	Critical	High	Medium	Low	Negligible	Total	Reduction
mysql:8.0.23	26 (3)	71 (7)	44 (5)	34 (3)	90 (11)	265(29)	89%
redis:6.2.1	20 (2)	67 (12)	32 (6)	25 (3)	60 (3)	204 (26)	87%
ghost:3.42.5-alpine	14 (12)	129 (108)	67 (46)	7 (7)	63 (2)	217 (173)	20%
registry:2.7.0	4 (3)	46 (37)	9 (7)	0 (0)	0 (0)	59(43)	27%
golang:1.16.2	63 (2)	376 (19)	314 (5)	80 (3)	499 (5)	1332(34)	97%
python:3.9.3	145 (15)	863 (47)	1048 (34)	418 (3)	898 (10)	3372 (109)	20%
bert_tf2:latest*	38 (26)	358 (256)	1426 (452)	549 (210)	57 (3)	2428 (947)	61%
nvidia_mrcnn_tf2:latest*	38 (26)	358 (225)	1445 (451)	558 (210)	57 (3)	2456 (945)	62%
merlin-pytorch-training:22.04*	47(46)	166(127)	902(134)	317(15)	70(3)	1502(325)	78%
merlin-tensorflow-training:22.04*	15(14)	109(76)	939(222)	304(34)	56(3)	1423(349)	75%

* These machine learning containers are collected from NVIDIA NGC services [37].


RQ2
ML containers are highly bloated.
Debloat ML containers
RQ1
Measure bloat in ML containers
ML Container
ML shared libraries are a major source of bloat.
Host Code
Device Code
ML Shared Library

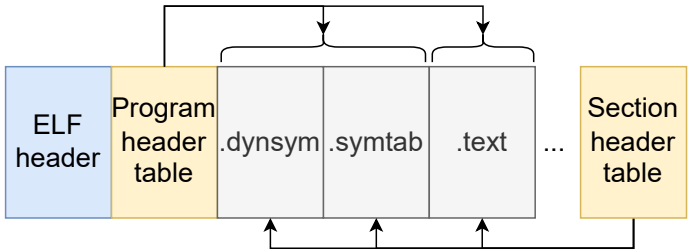
RQ3: Identify Bloat in Host Code



- Zhang, H. & Ali-Eldin, A. (2026). RTrace: Towards Better Visibility of Shared Library Execution. Proceedings of the 33rd Network and Distributed System Security (NDSS) Symposium.

Shared Library

- A shared library is a file that contains executable code designed to be used by multiple computer programs or other libraries at runtime.
- The file structure is organised according to the Executable and Linkable Format (ELF).



The ELF format.

Library Call Tracer and Debloating

Library Call Tracer and Debloating

- Identifying bloat in shared library host code involves determining which functions are actually executed by the workloads.

Library Call Tracer and Debloating

- Identifying bloat in shared library host code involves determining which functions are actually executed by the workloads.
- This can be achieved using a library call tracer, such as ltrace, which records executed function calls during runtime.

Library Call Tracer and Debloating

- Identifying bloat in shared library host code involves determining which functions are actually executed by the workloads.
- This can be achieved using a library call tracer, such as ltrace, which records executed function calls during runtime.
- During our evaluation, we found that relying on the tracing results from existing tracers often misses many executed functions.

Library Call Tracer and Debloating

- Identifying bloat in shared library host code involves determining which functions are actually executed by the workloads.
- This can be achieved using a library call tracer, such as ltrace, which records executed function calls during runtime.
- During our evaluation, we found that relying on the tracing results from existing tracers often misses many executed functions.
- As a result, debloating based on these incomplete traces can lead to broken shared libraries, since some actually executed code is mistakenly removed.

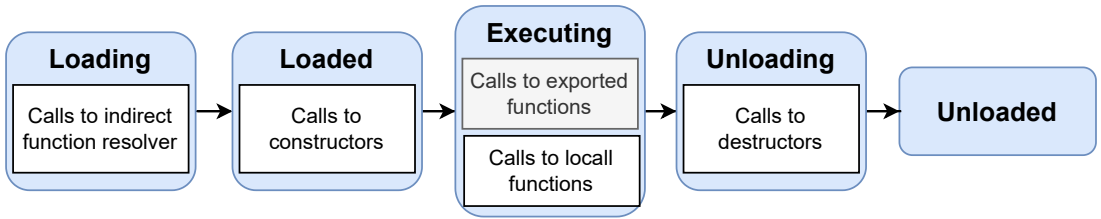
Limitations of Existing Library Call Tracers

Limitations of Existing Library Call Tracers

- Missing symbol tables: stripped shared libraries.

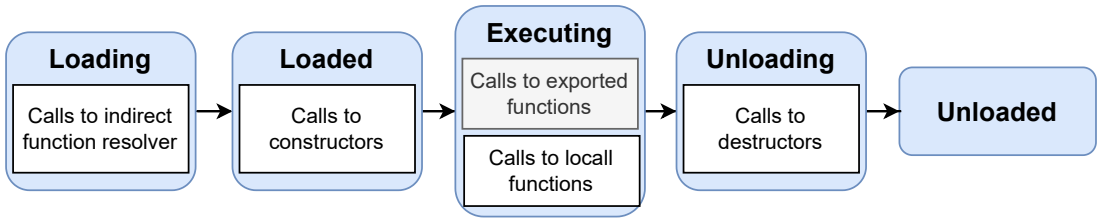
Limitations of Existing Library Call Tracers

- Missing symbol tables: stripped shared libraries.
- Early / implicit calls: indirect resolution, library constructors.



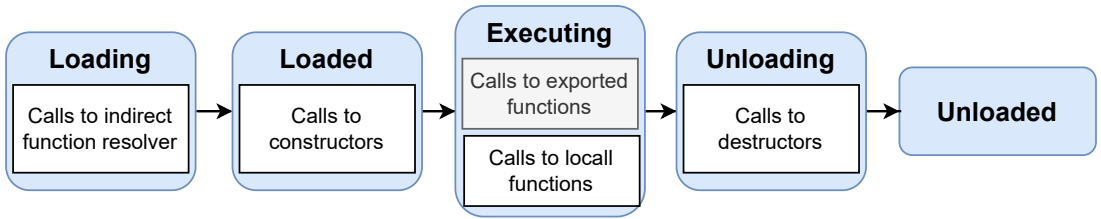
Limitations of Existing Library Call Tracers

- Missing symbol tables: stripped shared libraries.
- Early / implicit calls: indirect resolution, library constructors.
- Unconventional function calls.



Limitations of Existing Library Call Tracers

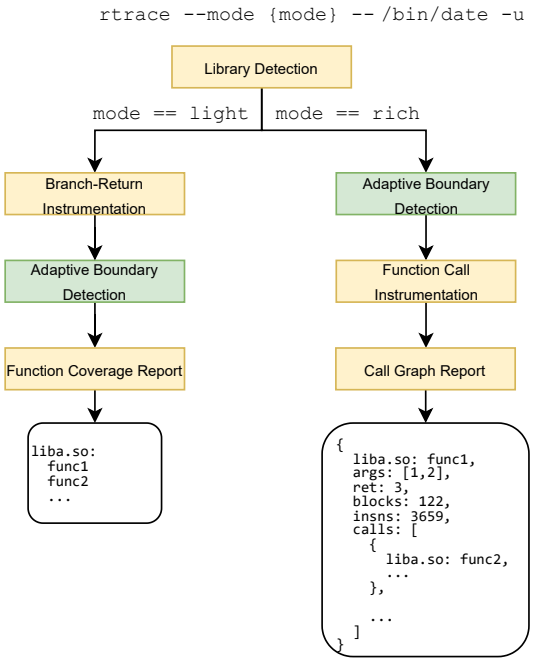
- Missing symbol tables: stripped shared libraries.
- Early / implicit calls: indirect resolution, library constructors.
- Unconventional function calls.
- High runtime overhead: drltrace slows a 50s task to over 1 hour.



RTrace Overview

RTrace Overview

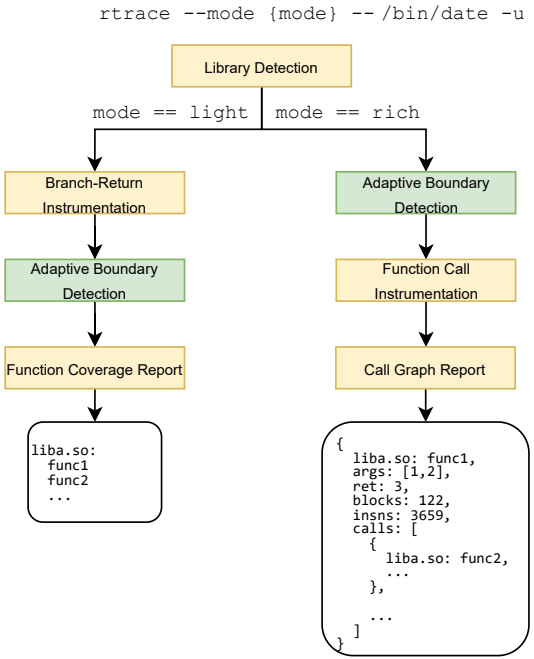
- RTrace: More complete & accurate shared library function tracing.



Workflow of RTrace.

RTrace Overview

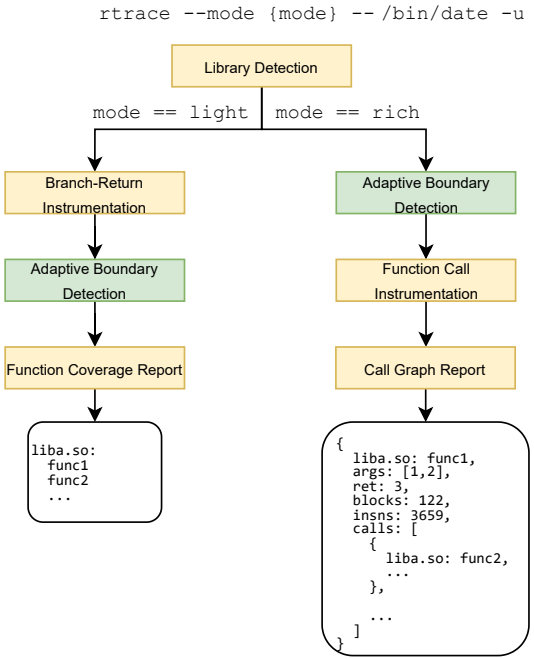
- RTrace: More complete & accurate shared library function tracing.
- RTrace has two modes: light mode and rich mode.



Workflow of RTrace.

RTrace Overview

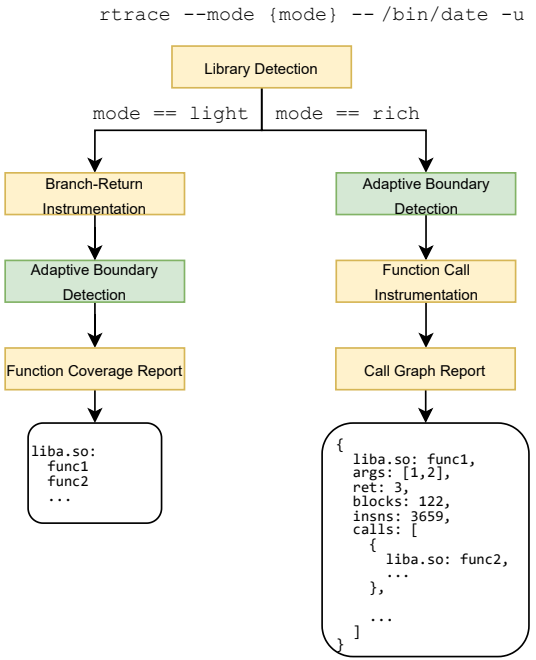
- RTrace: More complete & accurate shared library function tracing.
- RTrace has two modes: light mode and rich mode.
- Light Mode:



Workflow of RTrace.

RTrace Overview

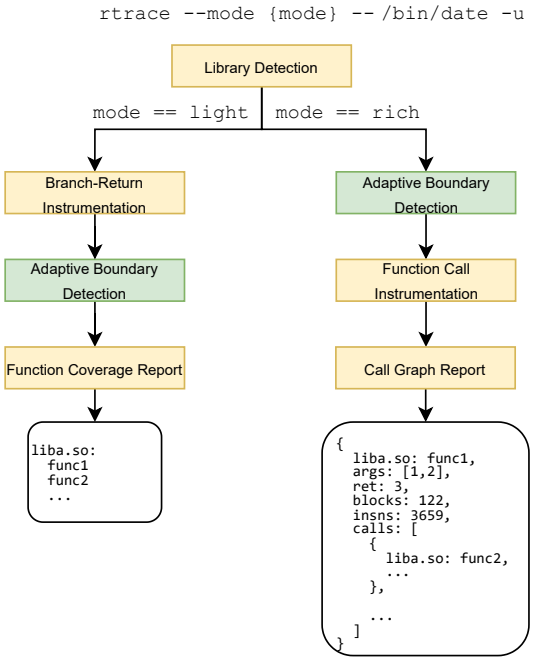
- RTrace: More complete & accurate shared library function tracing.
- RTrace has two modes: light mode and rich mode.
- Light Mode:
  - Precisely detects whether a function is executed



Workflow of RTrace.

RTrace Overview

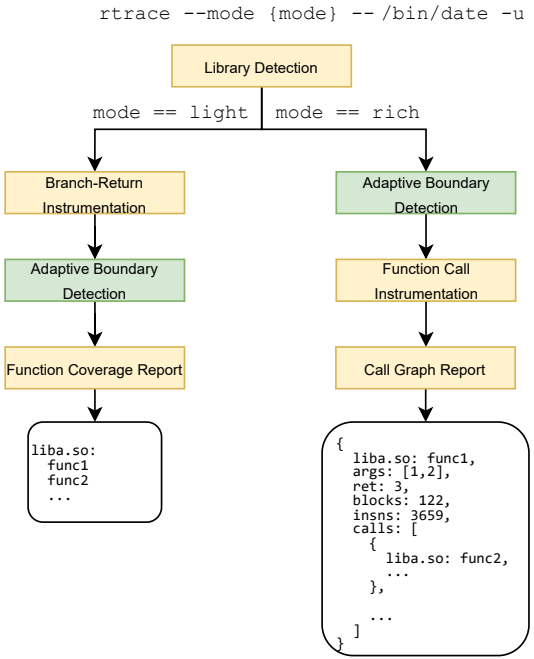
- RTrace: More complete & accurate shared library function tracing.
- RTrace has two modes: light mode and rich mode.
- Light Mode:
  - Precisely detects whether a function is executed
  - Minimal runtime overhead



Workflow of RTrace.

RTrace Overview

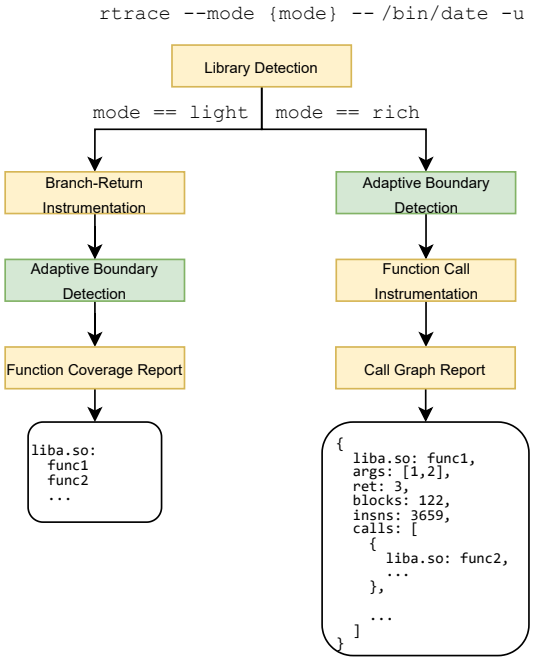
- RTrace: More complete & accurate shared library function tracing.
- RTrace has two modes: light mode and rich mode.
- Light Mode:
  - Precisely detects whether a function is executed
  - Minimal runtime overhead
- Rich Mode:



Workflow of RTrace.

RTrace Overview

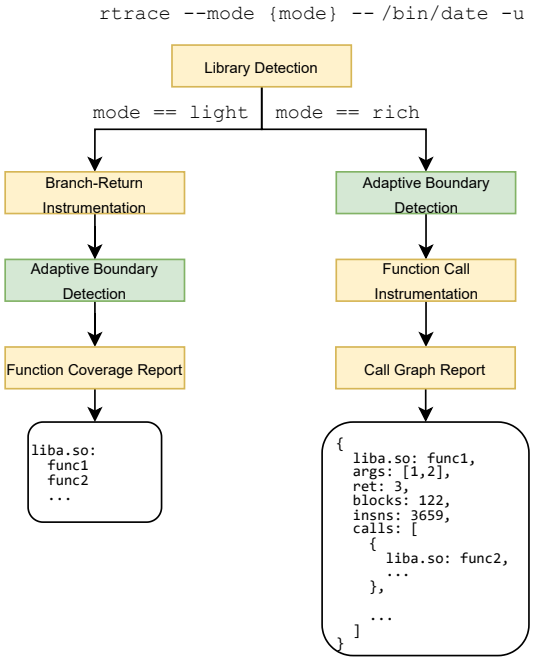
- RTrace: More complete & accurate shared library function tracing.
- RTrace has two modes: light mode and rich mode.
- Light Mode:
  - Precisely detects whether a function is executed
  - Minimal runtime overhead
- Rich Mode:
  - Generates function call graphs



Workflow of RTrace.

RTrace Overview

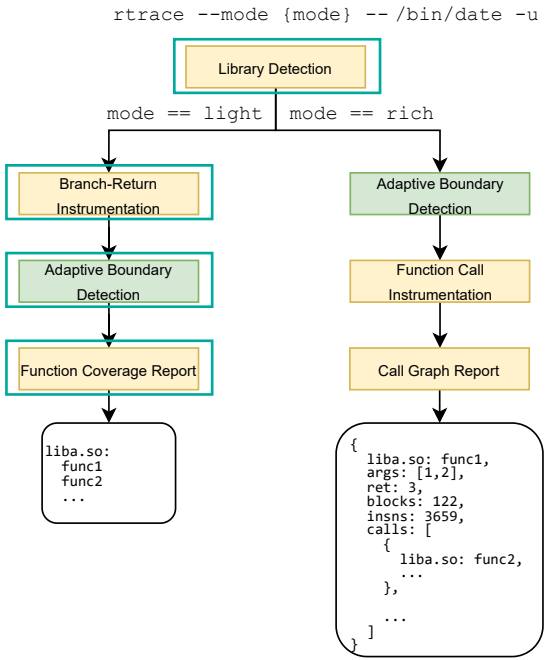
- RTrace: More complete & accurate shared library function tracing.
- RTrace has two modes: light mode and rich mode.
- Light Mode:
  - Precisely detects whether a function is executed
  - Minimal runtime overhead
- Rich Mode:
  - Generates function call graphs
  - Captures detailed runtime info (parameters, return values, etc.)



Workflow of RTrace.

RTrace Overview

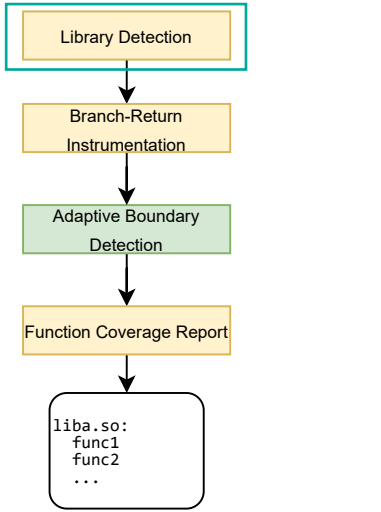
- RTrace: More complete & accurate shared library function tracing.
- RTrace has two modes: light mode and rich mode.
- Light Mode:
  - Precisely detects whether a function is executed
  - Minimal runtime overhead
- Rich Mode:
  - Generates function call graphs
  - Captures detailed runtime info (parameters, return values, etc.)



Workflow of RTrace.

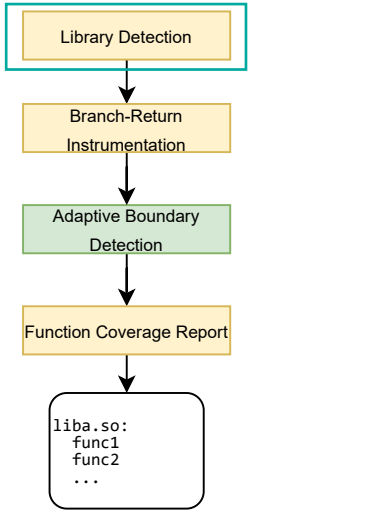
Light Mode: Library Detection

```
rtrace --mode light -- /bin/date -u
```



Light Mode: Library Detection

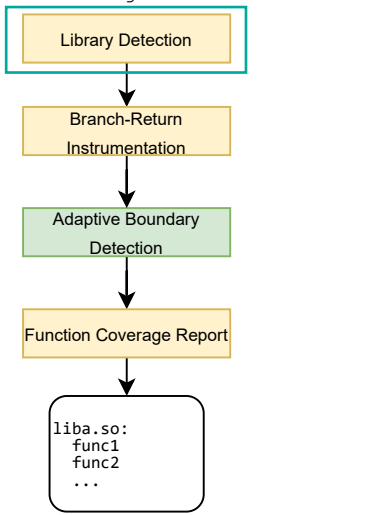
```
rtrace --mode light -- /bin/date -u
```



- This step aims to accurately identify all shared libraries used during program execution.

Light Mode: Library Detection

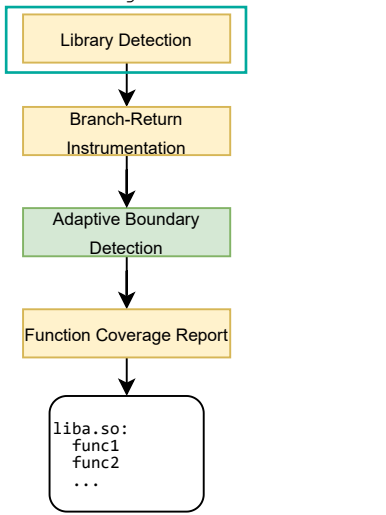
```
rtrace --mode light -- /bin/date -u
```



- This step aims to accurately identify all shared libraries used during program execution.
- Built on top of the binary instrumentation framework DynamoRIO, RTrace monitors shared library loading events.

Light Mode: Library Detection

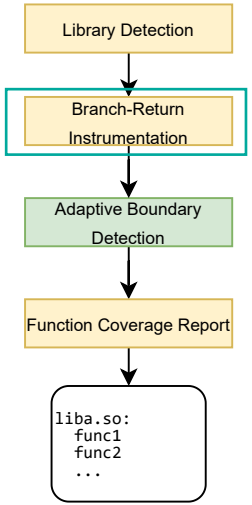
```
rtrace --mode light -- /bin/date -u
```



- This step aims to accurately identify all shared libraries used during program execution.
- Built on top of the binary instrumentation framework DynamoRIO, RTrace monitors shared library loading events.
  - Records both library file path and their memory segments.

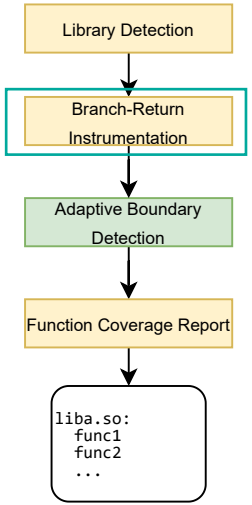
Light Mode: Branch-Return Instrumentation

```
rtrace --mode light -- /bin/date -u
```



Light Mode: Branch-Return Instrumentation

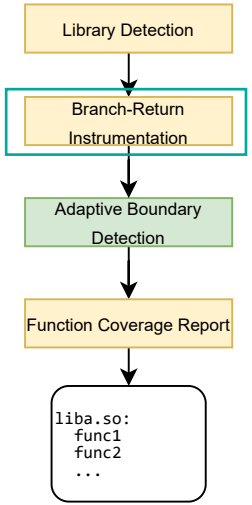
```
rtrace --mode light -- /bin/date -u
```



- This step aims to accurately identify executed functions.

Light Mode: Branch-Return Instrumentation

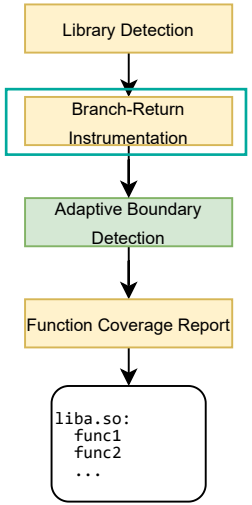
```
rtrace --mode light -- /bin/date -u
```



- This step aims to accurately identify executed functions.
- RTrace only instruments branch-related instructions (e.g., RET, JMP, CALL).

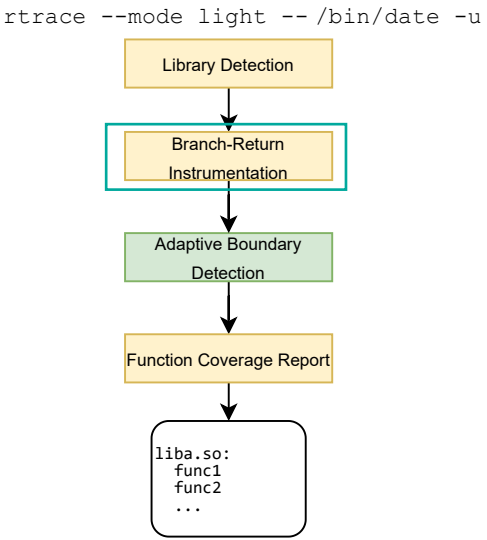
Light Mode: Branch-Return Instrumentation

```
rtrace --mode light -- /bin/date -u
```



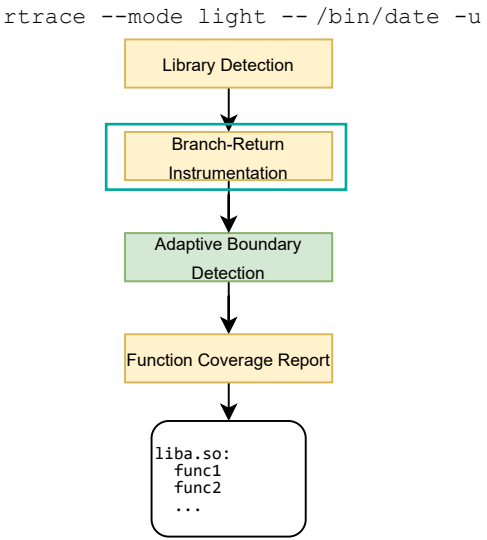
- This step aims to accurately identify executed functions.
- RTrace only instruments branch-related instructions (e.g., RET, JMP, CALL).
  - No need to instrument function entry points, since symbol tables may be incomplete.

Light Mode: Branch-Return Instrumentation



- This step aims to accurately identify executed functions.
- RTrace only instruments branch-related instructions (e.g., RET, JMP, CALL).
  - No need to instrument function entry points, since symbol tables may be incomplete.
  - Every function eventually exits through a branch instruction, ensuring full coverage, even for unconventional calls.

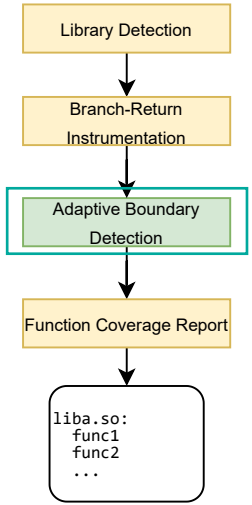
Light Mode: Branch-Return Instrumentation



- This step aims to accurately identify executed functions.
- RTrace only instruments branch-related instructions (e.g., RET, JMP, CALL).
  - No need to instrument function entry points, since symbol tables may be incomplete.
  - Every function eventually exits through a branch instruction, ensuring full coverage, even for unconventional calls.
  - Lower runtime overhead — no need to detect function boundaries during instrumentation.

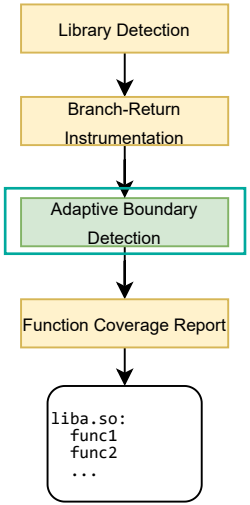
Light Mode: Adaptive Boundary Detection

```
rtrace --mode light -- /bin/date -u
```



Light Mode: Adaptive Boundary Detection

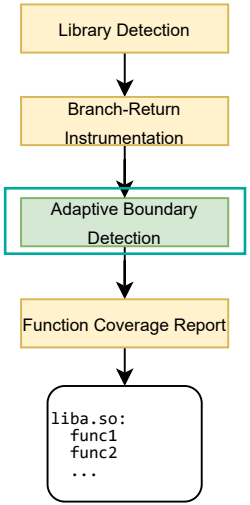
```
rtrace --mode light -- /bin/date -u
```



- This step aims to detect function boundaries without relying on on symbol table.

Light Mode: Adaptive Boundary Detection

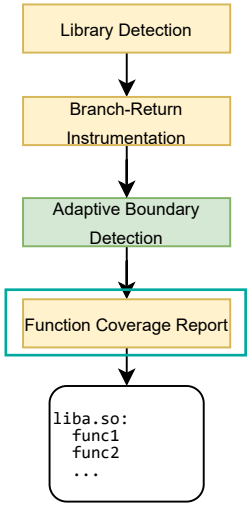
```
rtrace --mode light -- /bin/date -u
```



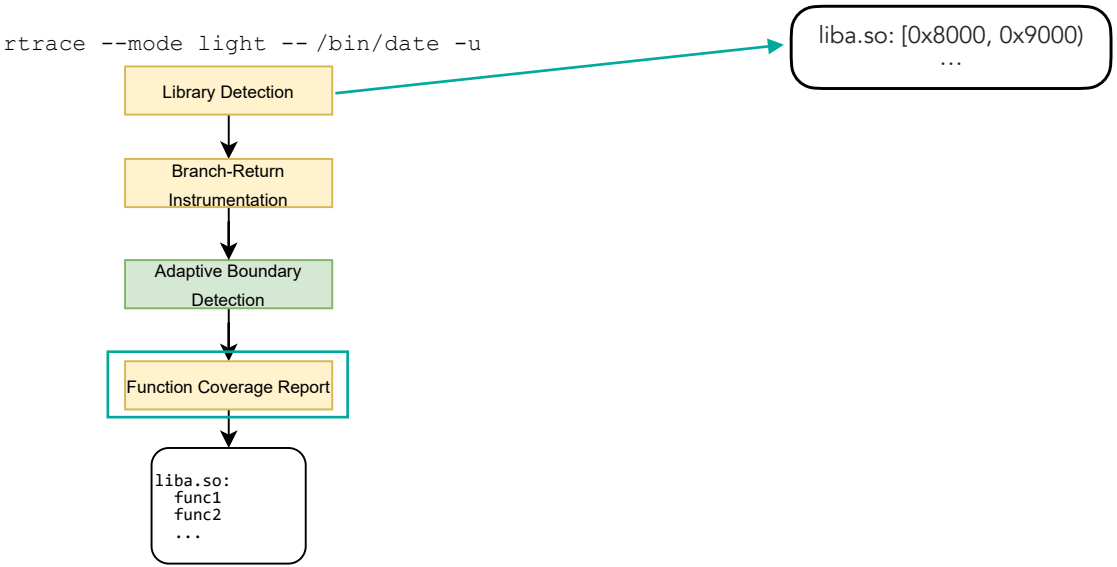
- This step aims to detect function boundaries without relying on on symbol table.
- RTrace leverages library metadata to select suitable boundary detection algorithms for different types of shared libraries.

Light Mode: Function Coverage Report

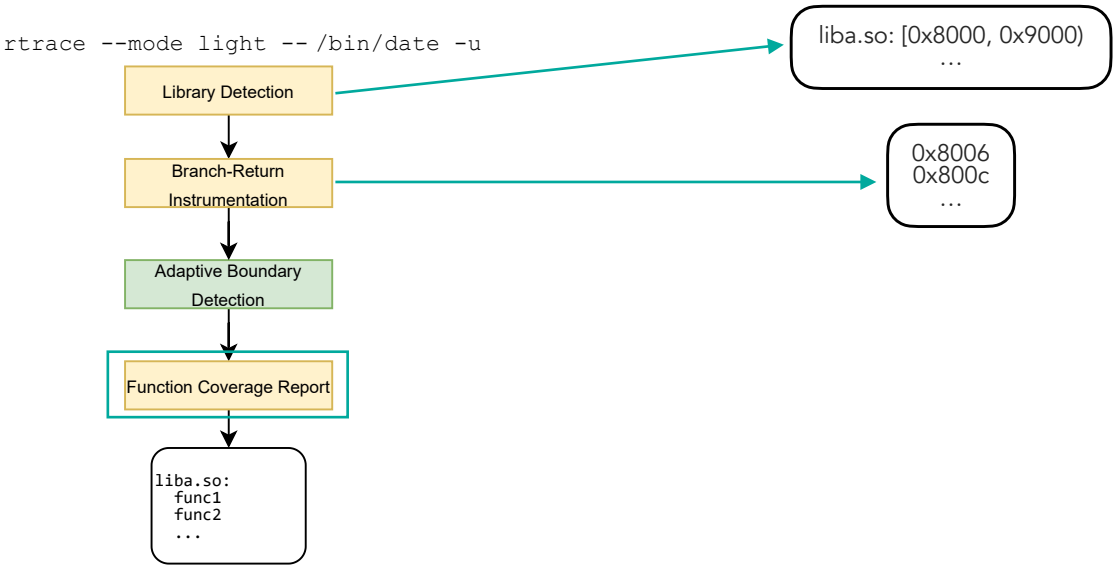
```
rtrace --mode light -- /bin/date -u
```



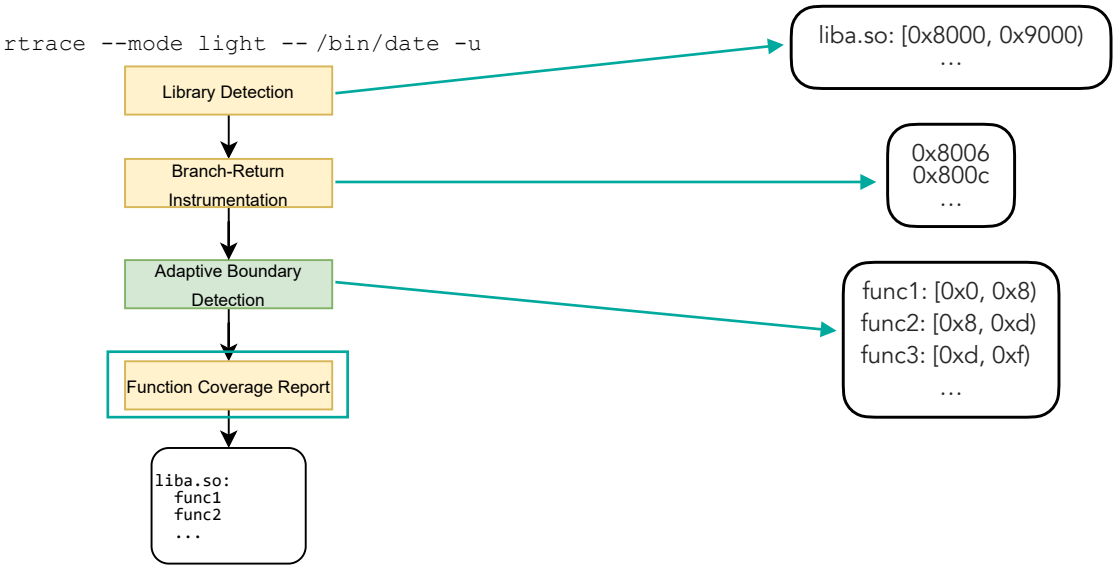
Light Mode: Function Coverage Report



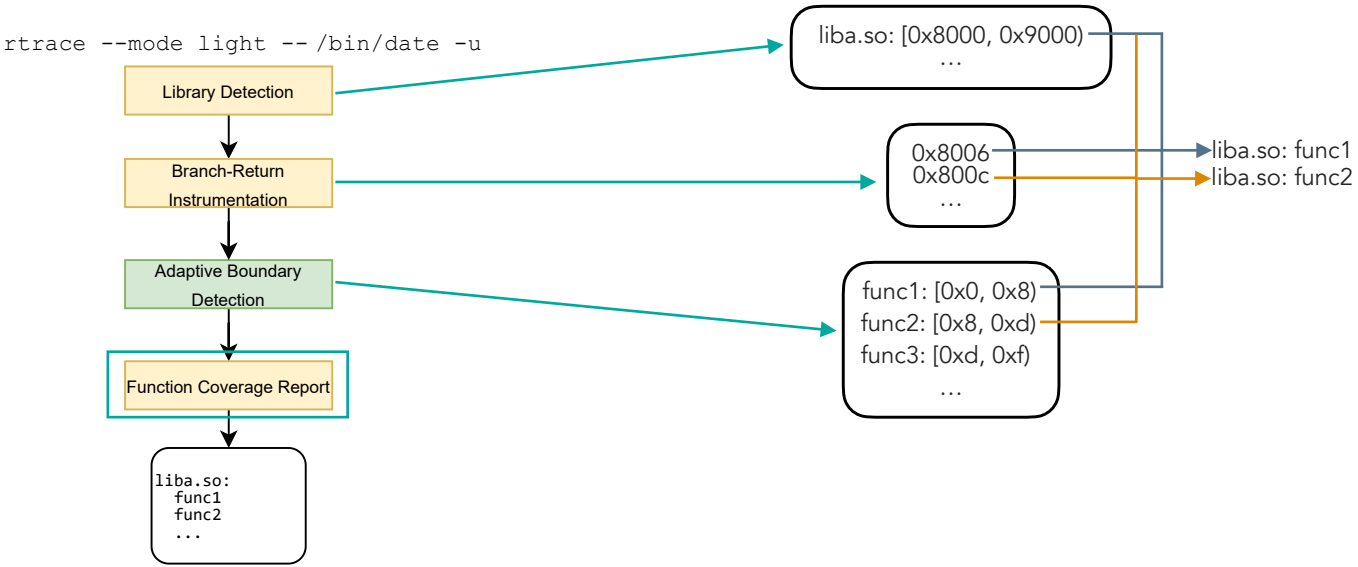
Light Mode: Function Coverage Report



Light Mode: Function Coverage Report



Light Mode: Function Coverage Report



Evaluation on Accuracy

Precision, recall and F1-score of each tracer on the benchmarks.

Benchmark	ltrace			drltrace			ldaudit			IntelPT			RTrace (light)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
bftpd	0.96	0.38	0.54	0.97	0.35	0.52	0.43	0.21	0.28	0.99	0.50	0.67	0.96	0.98	0.97
bzip2	0.96	0.28	0.43	0.97	0.32	0.48	0.71	0.11	0.20	0.99	0.54	0.70	0.97	1.00	0.98
chown	0.97	0.31	0.47	0.97	0.35	0.52	0.71	0.08	0.15	0.99	0.52	0.68	0.97	0.99	0.98
date	0.94	0.27	0.41	0.95	0.32	0.48	0.69	0.09	0.15	0.99	0.53	0.69	0.99	0.99	0.99
grep	0.98	0.27	0.42	0.98	0.31	0.47	0.80	0.13	0.22	0.99	0.55	0.71	0.99	0.99	0.99
gzip	1.00	0.15	0.26	0.97	0.19	0.31	0.61	0.11	0.18	0.98	0.50	0.66	0.99	1.00	1.00
imagemagick	0.99	0.25	0.40	0.99	0.56	0.71	0.92	0.49	0.64	1.00	0.56	0.72	0.98	0.99	0.98
lighttpd	0.97	0.29	0.45	0.97	0.31	0.47	0.81	0.14	0.24	0.99	0.54	0.70	0.99	0.99	0.99
make	0.95	0.36	0.52	0.95	0.39	0.56	0.61	0.17	0.26	0.98	0.58	0.73	0.96	0.99	0.97
memcached	0.96	0.34	0.51	0.94	0.35	0.51	0.62	0.22	0.32	0.99	0.56	0.71	0.94	0.98	0.96
mkdir	0.96	0.17	0.28	0.97	0.24	0.39	0.56	0.06	0.11	0.99	0.53	0.69	0.98	0.99	0.99
nginx	0.98	0.33	0.49	0.98	0.35	0.52	0.84	0.17	0.29	0.99	0.55	0.71	0.97	0.98	0.97
nmap	0.97	0.29	0.44	0.98	0.39	0.56	0.14	0.30	0.19	0.99	0.46	0.63	0.97	0.97	0.97
objdump	0.94	0.27	0.42	0.95	0.31	0.47	0.53	0.16	0.25	0.99	0.54	0.70	0.95	0.99	0.97
poppler	0.99	0.36	0.53	0.99	0.44	0.60	0.11	0.17	0.13	1.00	0.38	0.55	0.99	0.96	0.98
rm	0.97	0.22	0.36	0.97	0.28	0.43	0.68	0.09	0.16	0.88	0.52	0.65	0.99	0.99	0.99
sort	0.93	0.27	0.41	0.94	0.31	0.47	0.75	0.13	0.22	0.99	0.46	0.62	0.99	0.99	0.99
tar	0.96	0.35	0.51	0.98	0.31	0.48	0.37	0.18	0.24	0.99	0.49	0.66	0.98	0.99	0.98
uniq	0.95	0.26	0.41	0.96	0.32	0.48	0.72	0.09	0.17	0.99	0.53	0.69	0.99	0.99	0.99
wget	N/A[1]	N/A[1]	N/A[1]	1.00	0.41	0.58	0.54	0.15	0.24	1.00	0.26	0.41	0.99	0.98	0.98
ml_train	N/A[2]	N/A[2]	N/A[2]	0.99	0.03	0.06	0.88	0.03	0.05	1.00	0.24	0.38	0.93	0.92	0.92

[1] Due to the significant slowdown introduced by `ltrace`, the `wget` benchmark repeatedly timed out and failed run correctly.
[2] The `ml_train` benchmark failed to run with `ltrace`.

Evaluation on Accuracy

- Oracle: Establishing Ground Truth

Precision, recall and F1-score of each tracer on the benchmarks.

Benchmark	ltrace			drltrace			ldaudit			IntelPT			RTrace (light)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
bftpd	0.96	0.38	0.54	0.97	0.35	0.52	0.43	0.21	0.28	0.99	0.50	0.67	0.96	0.98	0.97
bzip2	0.96	0.28	0.43	0.97	0.32	0.48	0.71	0.11	0.20	0.99	0.54	0.70	0.97	1.00	0.98
chown	0.97	0.31	0.47	0.97	0.35	0.52	0.71	0.08	0.15	0.99	0.52	0.68	0.97	0.99	0.98
date	0.94	0.27	0.41	0.95	0.32	0.48	0.69	0.09	0.15	0.99	0.53	0.69	0.99	0.99	0.99
grep	0.98	0.27	0.42	0.98	0.31	0.47	0.80	0.13	0.22	0.99	0.55	0.71	0.99	0.99	0.99
gzip	1.00	0.15	0.26	0.97	0.19	0.31	0.61	0.11	0.18	0.98	0.50	0.66	0.99	1.00	1.00
imagemagick	0.99	0.25	0.40	0.99	0.56	0.71	0.92	0.49	0.64	1.00	0.56	0.72	0.98	0.99	0.98
lighttpd	0.97	0.29	0.45	0.97	0.31	0.47	0.81	0.14	0.24	0.99	0.54	0.70	0.99	0.99	0.99
make	0.95	0.36	0.52	0.95	0.39	0.56	0.61	0.17	0.26	0.98	0.58	0.73	0.96	0.99	0.97
memcached	0.96	0.34	0.51	0.94	0.35	0.51	0.62	0.22	0.32	0.99	0.56	0.71	0.94	0.98	0.96
mkdir	0.96	0.17	0.28	0.97	0.24	0.39	0.56	0.06	0.11	0.99	0.53	0.69	0.98	0.99	0.99
nginx	0.98	0.33	0.49	0.98	0.35	0.52	0.84	0.17	0.29	0.99	0.55	0.71	0.97	0.98	0.97
nmap	0.97	0.29	0.44	0.98	0.39	0.56	0.14	0.30	0.19	0.99	0.46	0.63	0.97	0.97	0.97
objdump	0.94	0.27	0.42	0.95	0.31	0.47	0.53	0.16	0.25	0.99	0.54	0.70	0.95	0.99	0.97
poppler	0.99	0.36	0.53	0.99	0.44	0.60	0.11	0.17	0.13	1.00	0.38	0.55	0.99	0.96	0.98
rm	0.97	0.22	0.36	0.97	0.28	0.43	0.68	0.09	0.16	0.88	0.52	0.65	0.99	0.99	0.99
sort	0.93	0.27	0.41	0.94	0.31	0.47	0.75	0.13	0.22	0.99	0.46	0.62	0.99	0.99	0.99
tar	0.96	0.35	0.51	0.98	0.31	0.48	0.37	0.18	0.24	0.99	0.49	0.66	0.98	0.99	0.98
uniq	0.95	0.26	0.41	0.96	0.32	0.48	0.72	0.09	0.17	0.99	0.53	0.69	0.99	0.99	0.99
wget	N/A[1]	N/A[1]	N/A[1]	1.00	0.41	0.58	0.54	0.15	0.24	1.00	0.26	0.41	0.99	0.98	0.98
ml_train	N/A[2]	N/A[2]	N/A[2]	0.99	0.03	0.06	0.88	0.03	0.05	1.00	0.24	0.38	0.93	0.92	0.92

[1] Due to the significant slowdown introduced by `ltrace`, the `wget` benchmark repeatedly timed out and failed run correctly.
[2] The `ml_train` benchmark failed to run with `ltrace`.

Evaluation on Accuracy

- Oracle: Establishing Ground Truth
- Obtain the debug symbol table for each shared library using corresponding package managers.

Precision, recall and F1-score of each tracer on the benchmarks.

Benchmark	ltrace			drltrace			ldaudit			IntelPT			RTrace (light)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
bftpd	0.96	0.38	0.54	0.97	0.35	0.52	0.43	0.21	0.28	0.99	0.50	0.67	0.96	0.98	0.97
bzip2	0.96	0.28	0.43	0.97	0.32	0.48	0.71	0.11	0.20	0.99	0.54	0.70	0.97	1.00	0.98
chown	0.97	0.31	0.47	0.97	0.35	0.52	0.71	0.08	0.15	0.99	0.52	0.68	0.97	0.99	0.98
date	0.94	0.27	0.41	0.95	0.32	0.48	0.69	0.09	0.15	0.99	0.53	0.69	0.99	0.99	0.99
grep	0.98	0.27	0.42	0.98	0.31	0.47	0.80	0.13	0.22	0.99	0.55	0.71	0.99	0.99	0.99
gzip	1.00	0.15	0.26	0.97	0.19	0.31	0.61	0.11	0.18	0.98	0.50	0.66	0.99	1.00	1.00
imagemagick	0.99	0.25	0.40	0.99	0.56	0.71	0.92	0.49	0.64	1.00	0.56	0.72	0.98	0.99	0.98
lighttpd	0.97	0.29	0.45	0.97	0.31	0.47	0.81	0.14	0.24	0.99	0.54	0.70	0.99	0.99	0.99
make	0.95	0.36	0.52	0.95	0.39	0.56	0.61	0.17	0.26	0.98	0.58	0.73	0.96	0.99	0.97
memcached	0.96	0.34	0.51	0.94	0.35	0.51	0.62	0.22	0.32	0.99	0.56	0.71	0.94	0.98	0.96
mkdir	0.96	0.17	0.28	0.97	0.24	0.39	0.56	0.06	0.11	0.99	0.53	0.69	0.98	0.99	0.99
nginx	0.98	0.33	0.49	0.98	0.35	0.52	0.84	0.17	0.29	0.99	0.55	0.71	0.97	0.98	0.97
nmap	0.97	0.29	0.44	0.98	0.39	0.56	0.14	0.30	0.19	0.99	0.46	0.63	0.97	0.97	0.97
objdump	0.94	0.27	0.42	0.95	0.31	0.47	0.53	0.16	0.25	0.99	0.54	0.70	0.95	0.99	0.97
poppler	0.99	0.36	0.53	0.99	0.44	0.60	0.11	0.17	0.13	1.00	0.38	0.55	0.99	0.96	0.98
rm	0.97	0.22	0.36	0.97	0.28	0.43	0.68	0.09	0.16	0.88	0.52	0.65	0.99	0.99	0.99
sort	0.93	0.27	0.41	0.94	0.31	0.47	0.75	0.13	0.22	0.99	0.46	0.62	0.99	0.99	0.99
tar	0.96	0.35	0.51	0.98	0.31	0.48	0.37	0.18	0.24	0.99	0.49	0.66	0.98	0.99	0.98
uniq	0.95	0.26	0.41	0.96	0.32	0.48	0.72	0.09	0.17	0.99	0.53	0.69	0.99	0.99	0.99
wget	N/A[1]	N/A[1]	N/A[1]	1.00	0.41	0.58	0.54	0.15	0.24	1.00	0.26	0.41	0.99	0.98	0.98
ml_train	N/A[2]	N/A[2]	N/A[2]	0.99	0.03	0.06	0.88	0.03	0.05	1.00	0.24	0.38	0.93	0.92	0.92

[1] Due to the significant slowdown introduced by `ltrace`, the `wget` benchmark repeatedly timed out and failed run correctly.
[2] The `ml_train` benchmark failed to run with `ltrace`.

Evaluation on Accuracy

- Oracle: Establishing Ground Truth
- Obtain the debug symbol table for each shared library using corresponding package managers.
- Perform instruction-level tracing to record **every** executed instruction address.

Precision, recall and F1-score of each tracer on the benchmarks.

Benchmark	ltrace			drltrace			ldaudit			IntelPT			RTrace (light)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
bftpd	0.96	0.38	0.54	0.97	0.35	0.52	0.43	0.21	0.28	0.99	0.50	0.67	0.96	0.98	0.97
bzip2	0.96	0.28	0.43	0.97	0.32	0.48	0.71	0.11	0.20	0.99	0.54	0.70	0.97	1.00	0.98
chown	0.97	0.31	0.47	0.97	0.35	0.52	0.71	0.08	0.15	0.99	0.52	0.68	0.97	0.99	0.98
date	0.94	0.27	0.41	0.95	0.32	0.48	0.69	0.09	0.15	0.99	0.53	0.69	0.99	0.99	0.99
grep	0.98	0.27	0.42	0.98	0.31	0.47	0.80	0.13	0.22	0.99	0.55	0.71	0.99	0.99	0.99
gzip	1.00	0.15	0.26	0.97	0.19	0.31	0.61	0.11	0.18	0.98	0.50	0.66	0.99	1.00	1.00
imagemagick	0.99	0.25	0.40	0.99	0.56	0.71	0.92	0.49	0.64	1.00	0.56	0.72	0.98	0.99	0.98
lighttpd	0.97	0.29	0.45	0.97	0.31	0.47	0.81	0.14	0.24	0.99	0.54	0.70	0.99	0.99	0.99
make	0.95	0.36	0.52	0.95	0.39	0.56	0.61	0.17	0.26	0.98	0.58	0.73	0.96	0.99	0.97
memcached	0.96	0.34	0.51	0.94	0.35	0.51	0.62	0.22	0.32	0.99	0.56	0.71	0.94	0.98	0.96
mkdir	0.96	0.17	0.28	0.97	0.24	0.39	0.56	0.06	0.11	0.99	0.53	0.69	0.98	0.99	0.99
nginx	0.98	0.33	0.49	0.98	0.35	0.52	0.84	0.17	0.29	0.99	0.55	0.71	0.97	0.98	0.97
nmap	0.97	0.29	0.44	0.98	0.39	0.56	0.14	0.30	0.19	0.99	0.46	0.63	0.97	0.97	0.97
objdump	0.94	0.27	0.42	0.95	0.31	0.47	0.53	0.16	0.25	0.99	0.54	0.70	0.95	0.99	0.97
poppler	0.99	0.36	0.53	0.99	0.44	0.60	0.11	0.17	0.13	1.00	0.38	0.55	0.99	0.96	0.98
rm	0.97	0.22	0.36	0.97	0.28	0.43	0.68	0.09	0.16	0.88	0.52	0.65	0.99	0.99	0.99
sort	0.93	0.27	0.41	0.94	0.31	0.47	0.75	0.13	0.22	0.99	0.46	0.62	0.99	0.99	0.99
tar	0.96	0.35	0.51	0.98	0.31	0.48	0.37	0.18	0.24	0.99	0.49	0.66	0.98	0.99	0.98
uniq	0.95	0.26	0.41	0.96	0.32	0.48	0.72	0.09	0.17	0.99	0.53	0.69	0.99	0.99	0.99
wget	N/A[1]	N/A[1]	N/A[1]	1.00	0.41	0.58	0.54	0.15	0.24	1.00	0.26	0.41	0.99	0.98	0.98
ml_train	N/A[2]	N/A[2]	N/A[2]	0.99	0.03	0.06	0.88	0.03	0.05	1.00	0.24	0.38	0.93	0.92	0.92

[1] Due to the significant slowdown introduced by `ltrace`, the `wget` benchmark repeatedly timed out and failed run correctly.
[2] The `ml_train` benchmark failed to run with `ltrace`.

Evaluation on Accuracy

- Oracle: Establishing Ground Truth
- Obtain the debug symbol table for each shared library using corresponding package managers.
- Perform instruction-level tracing to record **every** executed instruction address.
- A function is considered executed if any of its instructions run at runtime.

Precision, recall and F1-score of each tracer on the benchmarks.

Benchmark	ltrace			drltrace			ldaudit			IntelPT			RTrace (light)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
bftpd	0.96	0.38	0.54	0.97	0.35	0.52	0.43	0.21	0.28	0.99	0.50	0.67	0.96	0.98	0.97
bzip2	0.96	0.28	0.43	0.97	0.32	0.48	0.71	0.11	0.20	0.99	0.54	0.70	0.97	1.00	0.98
chown	0.97	0.31	0.47	0.97	0.35	0.52	0.71	0.08	0.15	0.99	0.52	0.68	0.97	0.99	0.98
date	0.94	0.27	0.41	0.95	0.32	0.48	0.69	0.09	0.15	0.99	0.53	0.69	0.99	0.99	0.99
grep	0.98	0.27	0.42	0.98	0.31	0.47	0.80	0.13	0.22	0.99	0.55	0.71	0.99	0.99	0.99
gzip	1.00	0.15	0.26	0.97	0.19	0.31	0.61	0.11	0.18	0.98	0.50	0.66	0.99	1.00	1.00
imagemagick	0.99	0.25	0.40	0.99	0.56	0.71	0.92	0.49	0.64	1.00	0.56	0.72	0.98	0.99	0.98
lighttpd	0.97	0.29	0.45	0.97	0.31	0.47	0.81	0.14	0.24	0.99	0.54	0.70	0.99	0.99	0.99
make	0.95	0.36	0.52	0.95	0.39	0.56	0.61	0.17	0.26	0.98	0.58	0.73	0.96	0.99	0.97
memcached	0.96	0.34	0.51	0.94	0.35	0.51	0.62	0.22	0.32	0.99	0.56	0.71	0.94	0.98	0.96
mkdir	0.96	0.17	0.28	0.97	0.24	0.39	0.56	0.06	0.11	0.99	0.53	0.69	0.98	0.99	0.99
nginx	0.98	0.33	0.49	0.98	0.35	0.52	0.84	0.17	0.29	0.99	0.55	0.71	0.97	0.98	0.97
nmap	0.97	0.29	0.44	0.98	0.39	0.56	0.14	0.30	0.19	0.99	0.46	0.63	0.97	0.97	0.97
objdump	0.94	0.27	0.42	0.95	0.31	0.47	0.53	0.16	0.25	0.99	0.54	0.70	0.95	0.99	0.97
poppler	0.99	0.36	0.53	0.99	0.44	0.60	0.11	0.17	0.13	1.00	0.38	0.55	0.99	0.96	0.98
rm	0.97	0.22	0.36	0.97	0.28	0.43	0.68	0.09	0.16	0.88	0.52	0.65	0.99	0.99	0.99
sort	0.93	0.27	0.41	0.94	0.31	0.47	0.75	0.13	0.22	0.99	0.46	0.62	0.99	0.99	0.99
tar	0.96	0.35	0.51	0.98	0.31	0.48	0.37	0.18	0.24	0.99	0.49	0.66	0.98	0.99	0.98
uniq	0.95	0.26	0.41	0.96	0.32	0.48	0.72	0.09	0.17	0.99	0.53	0.69	0.99	0.99	0.99
wget	N/A[1]	N/A[1]	N/A[1]	1.00	0.41	0.58	0.54	0.15	0.24	1.00	0.26	0.41	0.99	0.98	0.98
ml_train	N/A[2]	N/A[2]	N/A[2]	0.99	0.03	0.06	0.88	0.03	0.05	1.00	0.24	0.38	0.93	0.92	0.92

[1] Due to the significant slowdown introduced by `ltrace`, the `wget` benchmark repeatedly timed out and failed run correctly.
[2] The `ml_train` benchmark failed to run with `ltrace`.

Evaluation on Accuracy

- Oracle: Establishing Ground Truth
- Obtain the debug symbol table for each shared library using corresponding package managers.
- Perform instruction-level tracing to record **every** executed instruction address.
- A function is considered executed if any of its instructions run at runtime.
- Benchmarks include 103 shared libraries; 92 with available debug symbols serve as ground truth.

Precision, recall and F1-score of each tracer on the benchmarks.

Benchmark	ltrace			drltrace			ldaudit			IntelPT			RTrace (light)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
bftpd	0.96	0.38	0.54	0.97	0.35	0.52	0.43	0.21	0.28	0.99	0.50	0.67	0.96	0.98	0.97
bzip2	0.96	0.28	0.43	0.97	0.32	0.48	0.71	0.11	0.20	0.99	0.54	0.70	0.97	1.00	0.98
chown	0.97	0.31	0.47	0.97	0.35	0.52	0.71	0.08	0.15	0.99	0.52	0.68	0.97	0.99	0.98
date	0.94	0.27	0.41	0.95	0.32	0.48	0.69	0.09	0.15	0.99	0.53	0.69	0.99	0.99	0.99
grep	0.98	0.27	0.42	0.98	0.31	0.47	0.80	0.13	0.22	0.99	0.55	0.71	0.99	0.99	0.99
gzip	1.00	0.15	0.26	0.97	0.19	0.31	0.61	0.11	0.18	0.98	0.50	0.66	0.99	1.00	1.00
imagemagick	0.99	0.25	0.40	0.99	0.56	0.71	0.92	0.49	0.64	1.00	0.56	0.72	0.98	0.99	0.98
lighttpd	0.97	0.29	0.45	0.97	0.31	0.47	0.81	0.14	0.24	0.99	0.54	0.70	0.99	0.99	0.99
make	0.95	0.36	0.52	0.95	0.39	0.56	0.61	0.17	0.26	0.98	0.58	0.73	0.96	0.99	0.97
memcached	0.96	0.34	0.51	0.94	0.35	0.51	0.62	0.22	0.32	0.99	0.56	0.71	0.94	0.98	0.96
mkdir	0.96	0.17	0.28	0.97	0.24	0.39	0.56	0.06	0.11	0.99	0.53	0.69	0.98	0.99	0.99
nginx	0.98	0.33	0.49	0.98	0.35	0.52	0.84	0.17	0.29	0.99	0.55	0.71	0.97	0.98	0.97
nmap	0.97	0.29	0.44	0.98	0.39	0.56	0.14	0.30	0.19	0.99	0.46	0.63	0.97	0.97	0.97
objdump	0.94	0.27	0.42	0.95	0.31	0.47	0.53	0.16	0.25	0.99	0.54	0.70	0.95	0.99	0.97
poppler	0.99	0.36	0.53	0.99	0.44	0.60	0.11	0.17	0.13	1.00	0.38	0.55	0.99	0.96	0.98
rm	0.97	0.22	0.36	0.97	0.28	0.43	0.68	0.09	0.16	0.88	0.52	0.65	0.99	0.99	0.99
sort	0.93	0.27	0.41	0.94	0.31	0.47	0.75	0.13	0.22	0.99	0.46	0.62	0.99	0.99	0.99
tar	0.96	0.35	0.51	0.98	0.31	0.48	0.37	0.18	0.24	0.99	0.49	0.66	0.98	0.99	0.98
uniq	0.95	0.26	0.41	0.96	0.32	0.48	0.72	0.09	0.17	0.99	0.53	0.69	0.99	0.99	0.99
wget	N/A[1]	N/A[1]	N/A[1]	1.00	0.41	0.58	0.54	0.15	0.24	1.00	0.26	0.41	0.99	0.98	0.98
ml_train	N/A[2]	N/A[2]	N/A[2]	0.99	0.03	0.06	0.88	0.03	0.05	1.00	0.24	0.38	0.93	0.92	0.92

[1] Due to the significant slowdown introduced by `ltrace`, the `wget` benchmark repeatedly timed out and failed run correctly.
[2] The `ml_train` benchmark failed to run with `ltrace`.

Evaluation on Accuracy

Precision, recall and F1-score of each tracer on the benchmarks.

Benchmark	ltrace			drltrace			ldaudit			IntelPT			RTrace (light)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
bftpd	0.96	0.38	0.54	0.97	0.35	0.52	0.43	0.21	0.28	0.99	0.50	0.67	0.96	0.98	0.97
bzip2	0.96	0.28	0.43	0.97	0.32	0.48	0.71	0.11	0.20	0.99	0.54	0.70	0.97	1.00	0.98
chown	0.97	0.31	0.47	0.97	0.35	0.52	0.71	0.08	0.15	0.99	0.52	0.68	0.97	0.99	0.98
date	0.94	0.27	0.41	0.95	0.32	0.48	0.69	0.09	0.15	0.99	0.53	0.69	0.99	0.99	0.99
grep	0.98	0.27	0.42	0.98	0.31	0.47	0.80	0.13	0.22	0.99	0.55	0.71	0.99	0.99	0.99
gzip	1.00	0.15	0.26	0.97	0.19	0.31	0.61	0.11	0.18	0.98	0.50	0.66	0.99	1.00	1.00
imagemagick	0.99	0.25	0.40	0.99	0.56	0.71	0.92	0.49	0.64	1.00	0.56	0.72	0.98	0.99	0.98
lighttpd	0.97	0.29	0.45	0.97	0.31	0.47	0.81	0.14	0.24	0.99	0.54	0.70	0.99	0.99	0.99
make	0.95	0.36	0.52	0.95	0.39	0.56	0.61	0.17	0.26	0.98	0.58	0.73	0.96	0.99	0.97
memcached	0.96	0.34	0.51	0.94	0.35	0.51	0.62	0.22	0.32	0.99	0.56	0.71	0.94	0.98	0.96
mkdir	0.96	0.17	0.28	0.97	0.24	0.39	0.56	0.06	0.11	0.99	0.53	0.69	0.98	0.99	0.99
nginx	0.98	0.33	0.49	0.98	0.35	0.52	0.84	0.17	0.29	0.99	0.55	0.71	0.97	0.98	0.97
nmap	0.97	0.29	0.44	0.98	0.39	0.56	0.14	0.30	0.19	0.99	0.46	0.63	0.97	0.97	0.97
objdump	0.94	0.27	0.42	0.95	0.31	0.47	0.53	0.16	0.25	0.99	0.54	0.70	0.95	0.99	0.97
poppler	0.99	0.36	0.53	0.99	0.44	0.60	0.11	0.17	0.13	1.00	0.38	0.55	0.99	0.96	0.98
rm	0.97	0.22	0.36	0.97	0.28	0.43	0.68	0.09	0.16	0.88	0.52	0.65	0.99	0.99	0.99
sort	0.93	0.27	0.41	0.94	0.31	0.47	0.75	0.13	0.22	0.99	0.46	0.62	0.99	0.99	0.99
tar	0.96	0.35	0.51	0.98	0.31	0.48	0.37	0.18	0.24	0.99	0.49	0.66	0.98	0.99	0.98
uniq	0.95	0.26	0.41	0.96	0.32	0.48	0.72	0.09	0.17	0.99	0.53	0.69	0.99	0.99	0.99
wget	N/A[1]	N/A[1]	N/A[1]	1.00	0.41	0.58	0.54	0.15	0.24	1.00	0.26	0.41	0.99	0.98	0.98
ml_train	N/A[2]	N/A[2]	N/A[2]	0.99	0.03	0.06	0.88	0.03	0.05	1.00	0.24	0.38	0.93	0.92	0.92

[1] Due to the significant slowdown introduced by `ltrace`, the `wget` benchmark repeatedly timed out and failed run correctly.
[2] The `ml_train` benchmark failed to run with `ltrace`.

Evaluation on Accuracy

- Precision: the ratio of detected functions that are actually executed.

Precision, recall and F1-score of each tracer on the benchmarks.

Benchmark	ltrace			drltrace			ldaudit			IntelPT			RTrace (light)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
bftpd	0.96	0.38	0.54	0.97	0.35	0.52	0.43	0.21	0.28	0.99	0.50	0.67	0.96	0.98	0.97
bzip2	0.96	0.28	0.43	0.97	0.32	0.48	0.71	0.11	0.20	0.99	0.54	0.70	0.97	1.00	0.98
chown	0.97	0.31	0.47	0.97	0.35	0.52	0.71	0.08	0.15	0.99	0.52	0.68	0.97	0.99	0.98
date	0.94	0.27	0.41	0.95	0.32	0.48	0.69	0.09	0.15	0.99	0.53	0.69	0.99	0.99	0.99
grep	0.98	0.27	0.42	0.98	0.31	0.47	0.80	0.13	0.22	0.99	0.55	0.71	0.99	0.99	0.99
gzip	1.00	0.15	0.26	0.97	0.19	0.31	0.61	0.11	0.18	0.98	0.50	0.66	0.99	1.00	1.00
imagemagick	0.99	0.25	0.40	0.99	0.56	0.71	0.92	0.49	0.64	1.00	0.56	0.72	0.98	0.99	0.98
lighttpd	0.97	0.29	0.45	0.97	0.31	0.47	0.81	0.14	0.24	0.99	0.54	0.70	0.99	0.99	0.99
make	0.95	0.36	0.52	0.95	0.39	0.56	0.61	0.17	0.26	0.98	0.58	0.73	0.96	0.99	0.97
memcached	0.96	0.34	0.51	0.94	0.35	0.51	0.62	0.22	0.32	0.99	0.56	0.71	0.94	0.98	0.96
mkdir	0.96	0.17	0.28	0.97	0.24	0.39	0.56	0.06	0.11	0.99	0.53	0.69	0.98	0.99	0.99
nginx	0.98	0.33	0.49	0.98	0.35	0.52	0.84	0.17	0.29	0.99	0.55	0.71	0.97	0.98	0.97
nmap	0.97	0.29	0.44	0.98	0.39	0.56	0.14	0.30	0.19	0.99	0.46	0.63	0.97	0.97	0.97
objdump	0.94	0.27	0.42	0.95	0.31	0.47	0.53	0.16	0.25	0.99	0.54	0.70	0.95	0.99	0.97
poppler	0.99	0.36	0.53	0.99	0.44	0.60	0.11	0.17	0.13	1.00	0.38	0.55	0.99	0.96	0.98
rm	0.97	0.22	0.36	0.97	0.28	0.43	0.68	0.09	0.16	0.88	0.52	0.65	0.99	0.99	0.99
sort	0.93	0.27	0.41	0.94	0.31	0.47	0.75	0.13	0.22	0.99	0.46	0.62	0.99	0.99	0.99
tar	0.96	0.35	0.51	0.98	0.31	0.48	0.37	0.18	0.24	0.99	0.49	0.66	0.98	0.99	0.98
uniq	0.95	0.26	0.41	0.96	0.32	0.48	0.72	0.09	0.17	0.99	0.53	0.69	0.99	0.99	0.99
wget	N/A[1]	N/A[1]	N/A[1]	1.00	0.41	0.58	0.54	0.15	0.24	1.00	0.26	0.41	0.99	0.98	0.98
ml_train	N/A[2]	N/A[2]	N/A[2]	0.99	0.03	0.06	0.88	0.03	0.05	1.00	0.24	0.38	0.93	0.92	0.92

[1] Due to the significant slowdown introduced by `ltrace`, the `wget` benchmark repeatedly timed out and failed run correctly.
[2] The `ml_train` benchmark failed to run with `ltrace`.

Evaluation on Accuracy

- Precision: the ratio of detected functions that are actually executed.
- All tracers — except ldaudit — achieve very high precision, indicating few false positives.

Precision, recall and F1-score of each tracer on the benchmarks.

Benchmark	ltrace			drltrace			ldaudit			IntelPT			RTrace (light)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
bftpd	0.96	0.38	0.54	0.97	0.35	0.52	0.43	0.21	0.28	0.99	0.50	0.67	0.96	0.98	0.97
bzip2	0.96	0.28	0.43	0.97	0.32	0.48	0.71	0.11	0.20	0.99	0.54	0.70	0.97	1.00	0.98
chown	0.97	0.31	0.47	0.97	0.35	0.52	0.71	0.08	0.15	0.99	0.52	0.68	0.97	0.99	0.98
date	0.94	0.27	0.41	0.95	0.32	0.48	0.69	0.09	0.15	0.99	0.53	0.69	0.99	0.99	0.99
grep	0.98	0.27	0.42	0.98	0.31	0.47	0.80	0.13	0.22	0.99	0.55	0.71	0.99	0.99	0.99
gzip	1.00	0.15	0.26	0.97	0.19	0.31	0.61	0.11	0.18	0.98	0.50	0.66	0.99	1.00	1.00
imagemagick	0.99	0.25	0.40	0.99	0.56	0.71	0.92	0.49	0.64	1.00	0.56	0.72	0.98	0.99	0.98
lighttpd	0.97	0.29	0.45	0.97	0.31	0.47	0.81	0.14	0.24	0.99	0.54	0.70	0.99	0.99	0.99
make	0.95	0.36	0.52	0.95	0.39	0.56	0.61	0.17	0.26	0.98	0.58	0.73	0.96	0.99	0.97
memcached	0.96	0.34	0.51	0.94	0.35	0.51	0.62	0.22	0.32	0.99	0.56	0.71	0.94	0.98	0.96
mkdir	0.96	0.17	0.28	0.97	0.24	0.39	0.56	0.06	0.11	0.99	0.53	0.69	0.98	0.99	0.99
nginx	0.98	0.33	0.49	0.98	0.35	0.52	0.84	0.17	0.29	0.99	0.55	0.71	0.97	0.98	0.97
nmap	0.97	0.29	0.44	0.98	0.39	0.56	0.14	0.30	0.19	0.99	0.46	0.63	0.97	0.97	0.97
objdump	0.94	0.27	0.42	0.95	0.31	0.47	0.53	0.16	0.25	0.99	0.54	0.70	0.95	0.99	0.97
poppler	0.99	0.36	0.53	0.99	0.44	0.60	0.11	0.17	0.13	1.00	0.38	0.55	0.99	0.96	0.98
rm	0.97	0.22	0.36	0.97	0.28	0.43	0.68	0.09	0.16	0.88	0.52	0.65	0.99	0.99	0.99
sort	0.93	0.27	0.41	0.94	0.31	0.47	0.75	0.13	0.22	0.99	0.46	0.62	0.99	0.99	0.99
tar	0.96	0.35	0.51	0.98	0.31	0.48	0.37	0.18	0.24	0.99	0.49	0.66	0.98	0.99	0.98
uniq	0.95	0.26	0.41	0.96	0.32	0.48	0.72	0.09	0.17	0.99	0.53	0.69	0.99	0.99	0.99
wget	N/A[1]	N/A[1]	N/A[1]	1.00	0.41	0.58	0.54	0.15	0.24	1.00	0.26	0.41	0.99	0.98	0.98
ml_train	N/A[2]	N/A[2]	N/A[2]	0.99	0.03	0.06	0.88	0.03	0.05	1.00	0.24	0.38	0.93	0.92	0.92

[1] Due to the significant slowdown introduced by `ltrace`, the `wget` benchmark repeatedly timed out and failed run correctly.
[2] The `ml_train` benchmark failed to run with `ltrace`.

Evaluation on Accuracy

Precision, recall and F1-score of each tracer on the benchmarks.

Benchmark	ltrace			drltrace			ldaudit			IntelPT			RTrace (light)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
bftpd	0.96	0.38	0.54	0.97	0.35	0.52	0.43	0.21	0.28	0.99	0.50	0.67	0.96	0.98	0.97
bzip2	0.96	0.28	0.43	0.97	0.32	0.48	0.71	0.11	0.20	0.99	0.54	0.70	0.97	1.00	0.98
chown	0.97	0.31	0.47	0.97	0.35	0.52	0.71	0.08	0.15	0.99	0.52	0.68	0.97	0.99	0.98
date	0.94	0.27	0.41	0.95	0.32	0.48	0.69	0.09	0.15	0.99	0.53	0.69	0.99	0.99	0.99
grep	0.98	0.27	0.42	0.98	0.31	0.47	0.80	0.13	0.22	0.99	0.55	0.71	0.99	0.99	0.99
gzip	1.00	0.15	0.26	0.97	0.19	0.31	0.61	0.11	0.18	0.98	0.50	0.66	0.99	1.00	1.00
imagemagick	0.99	0.25	0.40	0.99	0.56	0.71	0.92	0.49	0.64	1.00	0.56	0.72	0.98	0.99	0.98
lighttpd	0.97	0.29	0.45	0.97	0.31	0.47	0.81	0.14	0.24	0.99	0.54	0.70	0.99	0.99	0.99
make	0.95	0.36	0.52	0.95	0.39	0.56	0.61	0.17	0.26	0.98	0.58	0.73	0.96	0.99	0.97
memcached	0.96	0.34	0.51	0.94	0.35	0.51	0.62	0.22	0.32	0.99	0.56	0.71	0.94	0.98	0.96
mkdir	0.96	0.17	0.28	0.97	0.24	0.39	0.56	0.06	0.11	0.99	0.53	0.69	0.98	0.99	0.99
nginx	0.98	0.33	0.49	0.98	0.35	0.52	0.84	0.17	0.29	0.99	0.55	0.71	0.97	0.98	0.97
nmap	0.97	0.29	0.44	0.98	0.39	0.56	0.14	0.30	0.19	0.99	0.46	0.63	0.97	0.97	0.97
objdump	0.94	0.27	0.42	0.95	0.31	0.47	0.53	0.16	0.25	0.99	0.54	0.70	0.95	0.99	0.97
poppler	0.99	0.36	0.53	0.99	0.44	0.60	0.11	0.17	0.13	1.00	0.38	0.55	0.99	0.96	0.98
rm	0.97	0.22	0.36	0.97	0.28	0.43	0.68	0.09	0.16	0.88	0.52	0.65	0.99	0.99	0.99
sort	0.93	0.27	0.41	0.94	0.31	0.47	0.75	0.13	0.22	0.99	0.46	0.62	0.99	0.99	0.99
tar	0.96	0.35	0.51	0.98	0.31	0.48	0.37	0.18	0.24	0.99	0.49	0.66	0.98	0.99	0.98
uniq	0.95	0.26	0.41	0.96	0.32	0.48	0.72	0.09	0.17	0.99	0.53	0.69	0.99	0.99	0.99
wget	N/A[1]	N/A[1]	N/A[1]	1.00	0.41	0.58	0.54	0.15	0.24	1.00	0.26	0.41	0.99	0.98	0.98
ml_train	N/A[2]	N/A[2]	N/A[2]	0.99	0.03	0.06	0.88	0.03	0.05	1.00	0.24	0.38	0.93	0.92	0.92

[1] Due to the significant slowdown introduced by `ltrace`, the `wget` benchmark repeatedly timed out and failed run correctly.
[2] The `ml_train` benchmark failed to run with `ltrace`.

Evaluation on Accuracy

- Recall: the ratio of executed functions that are detected.

Precision, recall and F1-score of each tracer on the benchmarks.

Benchmark	ltrace			drltrace			ldaudit			IntelPT			RTrace (light)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
bftpd	0.96	0.38	0.54	0.97	0.35	0.52	0.43	0.21	0.28	0.99	0.50	0.67	0.96	0.98	0.97
bzip2	0.96	0.28	0.43	0.97	0.32	0.48	0.71	0.11	0.20	0.99	0.54	0.70	0.97	1.00	0.98
chown	0.97	0.31	0.47	0.97	0.35	0.52	0.71	0.08	0.15	0.99	0.52	0.68	0.97	0.99	0.98
date	0.94	0.27	0.41	0.95	0.32	0.48	0.69	0.09	0.15	0.99	0.53	0.69	0.99	0.99	0.99
grep	0.98	0.27	0.42	0.98	0.31	0.47	0.80	0.13	0.22	0.99	0.55	0.71	0.99	0.99	0.99
gzip	1.00	0.15	0.26	0.97	0.19	0.31	0.61	0.11	0.18	0.98	0.50	0.66	0.99	1.00	1.00
imagemagick	0.99	0.25	0.40	0.99	0.56	0.71	0.92	0.49	0.64	1.00	0.56	0.72	0.98	0.99	0.98
lighttpd	0.97	0.29	0.45	0.97	0.31	0.47	0.81	0.14	0.24	0.99	0.54	0.70	0.99	0.99	0.99
make	0.95	0.36	0.52	0.95	0.39	0.56	0.61	0.17	0.26	0.98	0.58	0.73	0.96	0.99	0.97
memcached	0.96	0.34	0.51	0.94	0.35	0.51	0.62	0.22	0.32	0.99	0.56	0.71	0.94	0.98	0.96
mkdir	0.96	0.17	0.28	0.97	0.24	0.39	0.56	0.06	0.11	0.99	0.53	0.69	0.98	0.99	0.99
nginx	0.98	0.33	0.49	0.98	0.35	0.52	0.84	0.17	0.29	0.99	0.55	0.71	0.97	0.98	0.97
nmap	0.97	0.29	0.44	0.98	0.39	0.56	0.14	0.30	0.19	0.99	0.46	0.63	0.97	0.97	0.97
objdump	0.94	0.27	0.42	0.95	0.31	0.47	0.53	0.16	0.25	0.99	0.54	0.70	0.95	0.99	0.97
poppler	0.99	0.36	0.53	0.99	0.44	0.60	0.11	0.17	0.13	1.00	0.38	0.55	0.99	0.96	0.98
rm	0.97	0.22	0.36	0.97	0.28	0.43	0.68	0.09	0.16	0.88	0.52	0.65	0.99	0.99	0.99
sort	0.93	0.27	0.41	0.94	0.31	0.47	0.75	0.13	0.22	0.99	0.46	0.62	0.99	0.99	0.99
tar	0.96	0.35	0.51	0.98	0.31	0.48	0.37	0.18	0.24	0.99	0.49	0.66	0.98	0.99	0.98
uniq	0.95	0.26	0.41	0.96	0.32	0.48	0.72	0.09	0.17	0.99	0.53	0.69	0.99	0.99	0.99
wget	N/A[1]	N/A[1]	N/A[1]	1.00	0.41	0.58	0.54	0.15	0.24	1.00	0.26	0.41	0.99	0.98	0.98
ml_train	N/A[2]	N/A[2]	N/A[2]	0.99	0.03	0.06	0.88	0.03	0.05	1.00	0.24	0.38	0.93	0.92	0.92

[1] Due to the significant slowdown introduced by `ltrace`, the `wget` benchmark repeatedly timed out and failed run correctly.
[2] The `ml_train` benchmark failed to run with `ltrace`.

Evaluation on Accuracy

- Recall: the ratio of executed functions that are detected.
- All baseline tracers show very low recall, indicating that false negatives are high, i.e., many executed but undetected functions.

Precision, recall and F1-score of each tracer on the benchmarks.

Benchmark	ltrace			drltrace			ldaudit			IntelPT			RTrace (light)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
bftpd	0.96	0.38	0.54	0.97	0.35	0.52	0.43	0.21	0.28	0.99	0.50	0.67	0.96	0.98	0.97
bzip2	0.96	0.28	0.43	0.97	0.32	0.48	0.71	0.11	0.20	0.99	0.54	0.70	0.97	1.00	0.98
chown	0.97	0.31	0.47	0.97	0.35	0.52	0.71	0.08	0.15	0.99	0.52	0.68	0.97	0.99	0.98
date	0.94	0.27	0.41	0.95	0.32	0.48	0.69	0.09	0.15	0.99	0.53	0.69	0.99	0.99	0.99
grep	0.98	0.27	0.42	0.98	0.31	0.47	0.80	0.13	0.22	0.99	0.55	0.71	0.99	0.99	0.99
gzip	1.00	0.15	0.26	0.97	0.19	0.31	0.61	0.11	0.18	0.98	0.50	0.66	0.99	1.00	1.00
imagemagick	0.99	0.25	0.40	0.99	0.56	0.71	0.92	0.49	0.64	1.00	0.56	0.72	0.98	0.99	0.98
lighttpd	0.97	0.29	0.45	0.97	0.31	0.47	0.81	0.14	0.24	0.99	0.54	0.70	0.99	0.99	0.99
make	0.95	0.36	0.52	0.95	0.39	0.56	0.61	0.17	0.26	0.98	0.58	0.73	0.96	0.99	0.97
memcached	0.96	0.34	0.51	0.94	0.35	0.51	0.62	0.22	0.32	0.99	0.56	0.71	0.94	0.98	0.96
mkdir	0.96	0.17	0.28	0.97	0.24	0.39	0.56	0.06	0.11	0.99	0.53	0.69	0.98	0.99	0.99
nginx	0.98	0.33	0.49	0.98	0.35	0.52	0.84	0.17	0.29	0.99	0.55	0.71	0.97	0.98	0.97
nmap	0.97	0.29	0.44	0.98	0.39	0.56	0.14	0.30	0.19	0.99	0.46	0.63	0.97	0.97	0.97
objdump	0.94	0.27	0.42	0.95	0.31	0.47	0.53	0.16	0.25	0.99	0.54	0.70	0.95	0.99	0.97
poppler	0.99	0.36	0.53	0.99	0.44	0.60	0.11	0.17	0.13	1.00	0.38	0.55	0.99	0.96	0.98
rm	0.97	0.22	0.36	0.97	0.28	0.43	0.68	0.09	0.16	0.88	0.52	0.65	0.99	0.99	0.99
sort	0.93	0.27	0.41	0.94	0.31	0.47	0.75	0.13	0.22	0.99	0.46	0.62	0.99	0.99	0.99
tar	0.96	0.35	0.51	0.98	0.31	0.48	0.37	0.18	0.24	0.99	0.49	0.66	0.98	0.99	0.98
uniq	0.95	0.26	0.41	0.96	0.32	0.48	0.72	0.09	0.17	0.99	0.53	0.69	0.99	0.99	0.99
wget	N/A[1]	N/A[1]	N/A[1]	1.00	0.41	0.58	0.54	0.15	0.24	1.00	0.26	0.41	0.99	0.98	0.98
ml_train	N/A[2]	N/A[2]	N/A[2]	0.99	0.03	0.06	0.88	0.03	0.05	1.00	0.24	0.38	0.93	0.92	0.92

[1] Due to the significant slowdown introduced by `ltrace`, the `wget` benchmark repeatedly timed out and failed run correctly.
[2] The `ml_train` benchmark failed to run with `ltrace`.

Evaluation on Accuracy

- Recall: the ratio of executed functions that are detected.
- All baseline tracers show very low recall, indicating that false negatives are high, i.e., many executed but undetected functions.
- RTrace still shows very high recall, capturing nearly all executed function.

Precision, recall and F1-score of each tracer on the benchmarks.

Benchmark	ltrace			drltrace			ldaudit			IntelPT			RTrace (light)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
bftpd	0.96	0.38	0.54	0.97	0.35	0.52	0.43	0.21	0.28	0.99	0.50	0.67	0.96	0.98	0.97
bzip2	0.96	0.28	0.43	0.97	0.32	0.48	0.71	0.11	0.20	0.99	0.54	0.70	0.97	1.00	0.98
chown	0.97	0.31	0.47	0.97	0.35	0.52	0.71	0.08	0.15	0.99	0.52	0.68	0.97	0.99	0.98
date	0.94	0.27	0.41	0.95	0.32	0.48	0.69	0.09	0.15	0.99	0.53	0.69	0.99	0.99	0.99
grep	0.98	0.27	0.42	0.98	0.31	0.47	0.80	0.13	0.22	0.99	0.55	0.71	0.99	0.99	0.99
gzip	1.00	0.15	0.26	0.97	0.19	0.31	0.61	0.11	0.18	0.98	0.50	0.66	0.99	1.00	1.00
imagemagick	0.99	0.25	0.40	0.99	0.56	0.71	0.92	0.49	0.64	1.00	0.56	0.72	0.98	0.99	0.98
lighttpd	0.97	0.29	0.45	0.97	0.31	0.47	0.81	0.14	0.24	0.99	0.54	0.70	0.99	0.99	0.99
make	0.95	0.36	0.52	0.95	0.39	0.56	0.61	0.17	0.26	0.98	0.58	0.73	0.96	0.99	0.97
memcached	0.96	0.34	0.51	0.94	0.35	0.51	0.62	0.22	0.32	0.99	0.56	0.71	0.94	0.98	0.96
mkdir	0.96	0.17	0.28	0.97	0.24	0.39	0.56	0.06	0.11	0.99	0.53	0.69	0.98	0.99	0.99
nginx	0.98	0.33	0.49	0.98	0.35	0.52	0.84	0.17	0.29	0.99	0.55	0.71	0.97	0.98	0.97
nmap	0.97	0.29	0.44	0.98	0.39	0.56	0.14	0.30	0.19	0.99	0.46	0.63	0.97	0.97	0.97
objdump	0.94	0.27	0.42	0.95	0.31	0.47	0.53	0.16	0.25	0.99	0.54	0.70	0.95	0.99	0.97
poppler	0.99	0.36	0.53	0.99	0.44	0.60	0.11	0.17	0.13	1.00	0.38	0.55	0.99	0.96	0.98
rm	0.97	0.22	0.36	0.97	0.28	0.43	0.68	0.09	0.16	0.88	0.52	0.65	0.99	0.99	0.99
sort	0.93	0.27	0.41	0.94	0.31	0.47	0.75	0.13	0.22	0.99	0.46	0.62	0.99	0.99	0.99
tar	0.96	0.35	0.51	0.98	0.31	0.48	0.37	0.18	0.24	0.99	0.49	0.66	0.98	0.99	0.98
uniq	0.95	0.26	0.41	0.96	0.32	0.48	0.72	0.09	0.17	0.99	0.53	0.69	0.99	0.99	0.99
wget	N/A[1]	N/A[1]	N/A[1]	1.00	0.41	0.58	0.54	0.15	0.24	1.00	0.26	0.41	0.99	0.98	0.98
ml_train	N/A[2]	N/A[2]	N/A[2]	0.99	0.03	0.06	0.88	0.03	0.05	1.00	0.24	0.38	0.93	0.92	0.92

[1] Due to the significant slowdown introduced by `ltrace`, the `wget` benchmark repeatedly timed out and failed run correctly.
[2] The `ml_train` benchmark failed to run with `ltrace`.

Evaluation on Accuracy

Precision, recall and F1-score of each tracer on the benchmarks.

Benchmark	ltrace			drltrace			ldaudit			IntelPT			RTrace (light)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
bftpd	0.96	0.38	0.54	0.97	0.35	0.52	0.43	0.21	0.28	0.99	0.50	0.67	0.96	0.98	0.97
bzip2	0.96	0.28	0.43	0.97	0.32	0.48	0.71	0.11	0.20	0.99	0.54	0.70	0.97	1.00	0.98
chown	0.97	0.31	0.47	0.97	0.35	0.52	0.71	0.08	0.15	0.99	0.52	0.68	0.97	0.99	0.98
date	0.94	0.27	0.41	0.95	0.32	0.48	0.69	0.09	0.15	0.99	0.53	0.69	0.99	0.99	0.99
grep	0.98	0.27	0.42	0.98	0.31	0.47	0.80	0.13	0.22	0.99	0.55	0.71	0.99	0.99	0.99
gzip	1.00	0.15	0.26	0.97	0.19	0.31	0.61	0.11	0.18	0.98	0.50	0.66	0.99	1.00	1.00
imagemagick	0.99	0.25	0.40	0.99	0.56	0.71	0.92	0.49	0.64	1.00	0.56	0.72	0.98	0.99	0.98
lighttpd	0.97	0.29	0.45	0.97	0.31	0.47	0.81	0.14	0.24	0.99	0.54	0.70	0.99	0.99	0.99
make	0.95	0.36	0.52	0.95	0.39	0.56	0.61	0.17	0.26	0.98	0.58	0.73	0.96	0.99	0.97
memcached	0.96	0.34	0.51	0.94	0.35	0.51	0.62	0.22	0.32	0.99	0.56	0.71	0.94	0.98	0.96
mkdir	0.96	0.17	0.28	0.97	0.24	0.39	0.56	0.06	0.11	0.99	0.53	0.69	0.98	0.99	0.99
nginx	0.98	0.33	0.49	0.98	0.35	0.52	0.84	0.17	0.29	0.99	0.55	0.71	0.97	0.98	0.97
nmap	0.97	0.29	0.44	0.98	0.39	0.56	0.14	0.30	0.19	0.99	0.46	0.63	0.97	0.97	0.97
objdump	0.94	0.27	0.42	0.95	0.31	0.47	0.53	0.16	0.25	0.99	0.54	0.70	0.95	0.99	0.97
poppler	0.99	0.36	0.53	0.99	0.44	0.60	0.11	0.17	0.13	1.00	0.38	0.55	0.99	0.96	0.98
rm	0.97	0.22	0.36	0.97	0.28	0.43	0.68	0.09	0.16	0.88	0.52	0.65	0.99	0.99	0.99
sort	0.93	0.27	0.41	0.94	0.31	0.47	0.75	0.13	0.22	0.99	0.46	0.62	0.99	0.99	0.99
tar	0.96	0.35	0.51	0.98	0.31	0.48	0.37	0.18	0.24	0.99	0.49	0.66	0.98	0.99	0.98
uniq	0.95	0.26	0.41	0.96	0.32	0.48	0.72	0.09	0.17	0.99	0.53	0.69	0.99	0.99	0.99
wget	N/A[1]	N/A[1]	N/A[1]	1.00	0.41	0.58	0.54	0.15	0.24	1.00	0.26	0.41	0.99	0.98	0.98
ml_train	N/A[2]	N/A[2]	N/A[2]	0.99	0.03	0.06	0.88	0.03	0.05	1.00	0.24	0.38	0.93	0.92	0.92

[1] Due to the significant slowdown introduced by `ltrace`, the `wget` benchmark repeatedly timed out and failed run correctly.
[2] The `ml_train` benchmark failed to run with `ltrace`.

Evaluation on Accuracy

- F1-Score: overall accuracy by combining precision and recall.

Precision, recall and F1-score of each tracer on the benchmarks.

Benchmark	ltrace			drltrace			ldaudit			IntelPT			RTrace (light)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
bftpd	0.96	0.38	0.54	0.97	0.35	0.52	0.43	0.21	0.28	0.99	0.50	0.67	0.96	0.98	0.97
bzip2	0.96	0.28	0.43	0.97	0.32	0.48	0.71	0.11	0.20	0.99	0.54	0.70	0.97	1.00	0.98
chown	0.97	0.31	0.47	0.97	0.35	0.52	0.71	0.08	0.15	0.99	0.52	0.68	0.97	0.99	0.98
date	0.94	0.27	0.41	0.95	0.32	0.48	0.69	0.09	0.15	0.99	0.53	0.69	0.99	0.99	0.99
grep	0.98	0.27	0.42	0.98	0.31	0.47	0.80	0.13	0.22	0.99	0.55	0.71	0.99	0.99	0.99
gzip	1.00	0.15	0.26	0.97	0.19	0.31	0.61	0.11	0.18	0.98	0.50	0.66	0.99	1.00	1.00
imagemagick	0.99	0.25	0.40	0.99	0.56	0.71	0.92	0.49	0.64	1.00	0.56	0.72	0.98	0.99	0.98
lighttpd	0.97	0.29	0.45	0.97	0.31	0.47	0.81	0.14	0.24	0.99	0.54	0.70	0.99	0.99	0.99
make	0.95	0.36	0.52	0.95	0.39	0.56	0.61	0.17	0.26	0.98	0.58	0.73	0.96	0.99	0.97
memcached	0.96	0.34	0.51	0.94	0.35	0.51	0.62	0.22	0.32	0.99	0.56	0.71	0.94	0.98	0.96
mkdir	0.96	0.17	0.28	0.97	0.24	0.39	0.56	0.06	0.11	0.99	0.53	0.69	0.98	0.99	0.99
nginx	0.98	0.33	0.49	0.98	0.35	0.52	0.84	0.17	0.29	0.99	0.55	0.71	0.97	0.98	0.97
nmap	0.97	0.29	0.44	0.98	0.39	0.56	0.14	0.30	0.19	0.99	0.46	0.63	0.97	0.97	0.97
objdump	0.94	0.27	0.42	0.95	0.31	0.47	0.53	0.16	0.25	0.99	0.54	0.70	0.95	0.99	0.97
poppler	0.99	0.36	0.53	0.99	0.44	0.60	0.11	0.17	0.13	1.00	0.38	0.55	0.99	0.96	0.98
rm	0.97	0.22	0.36	0.97	0.28	0.43	0.68	0.09	0.16	0.88	0.52	0.65	0.99	0.99	0.99
sort	0.93	0.27	0.41	0.94	0.31	0.47	0.75	0.13	0.22	0.99	0.46	0.62	0.99	0.99	0.99
tar	0.96	0.35	0.51	0.98	0.31	0.48	0.37	0.18	0.24	0.99	0.49	0.66	0.98	0.99	0.98
uniq	0.95	0.26	0.41	0.96	0.32	0.48	0.72	0.09	0.17	0.99	0.53	0.69	0.99	0.99	0.99
wget	N/A[1]	N/A[1]	N/A[1]	1.00	0.41	0.58	0.54	0.15	0.24	1.00	0.26	0.41	0.99	0.98	0.98
ml_train	N/A[2]	N/A[2]	N/A[2]	0.99	0.03	0.06	0.88	0.03	0.05	1.00	0.24	0.38	0.93	0.92	0.92

[1] Due to the significant slowdown introduced by `ltrace`, the `wget` benchmark repeatedly timed out and failed run correctly.
[2] The `ml_train` benchmark failed to run with `ltrace`.

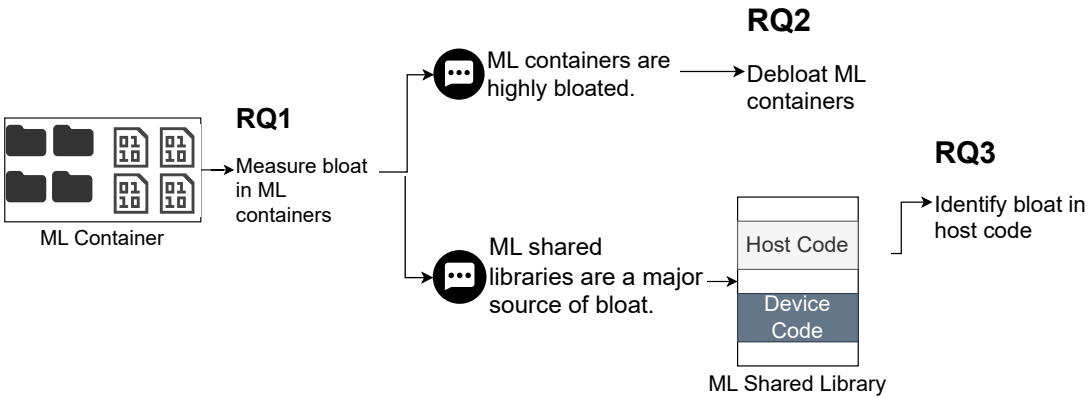
Evaluation on Accuracy

- F1-Score: overall accuracy by combining precision and recall.
- RTrace achieves the highest F1-Score across all benchmarks.

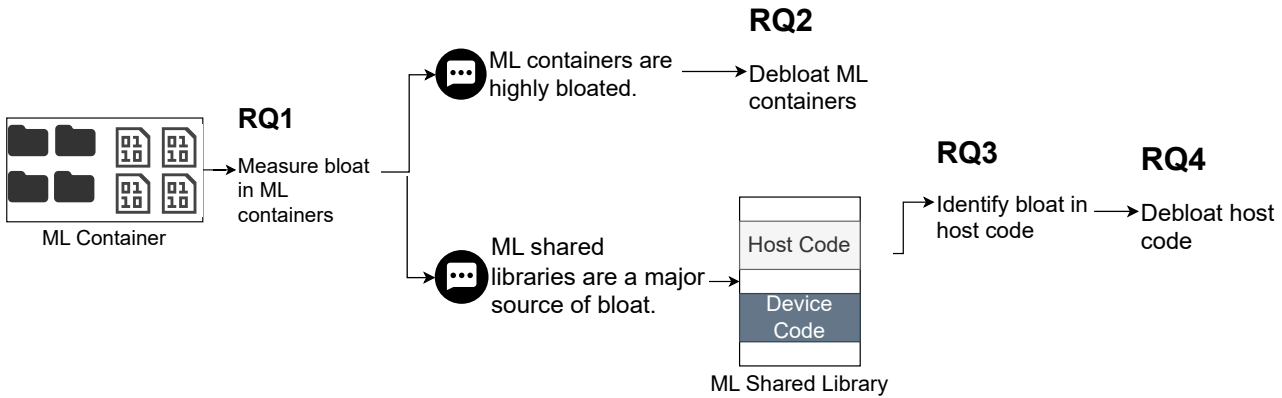
Precision, recall and F1-score of each tracer on the benchmarks.

Benchmark	ltrace			drltrace			ldaudit			IntelPT			RTrace (light)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
bftpd	0.96	0.38	0.54	0.97	0.35	0.52	0.43	0.21	0.28	0.99	0.50	0.67	0.96	0.98	0.97
bzip2	0.96	0.28	0.43	0.97	0.32	0.48	0.71	0.11	0.20	0.99	0.54	0.70	0.97	1.00	0.98
chown	0.97	0.31	0.47	0.97	0.35	0.52	0.71	0.08	0.15	0.99	0.52	0.68	0.97	0.99	0.98
date	0.94	0.27	0.41	0.95	0.32	0.48	0.69	0.09	0.15	0.99	0.53	0.69	0.99	0.99	0.99
grep	0.98	0.27	0.42	0.98	0.31	0.47	0.80	0.13	0.22	0.99	0.55	0.71	0.99	0.99	0.99
gzip	1.00	0.15	0.26	0.97	0.19	0.31	0.61	0.11	0.18	0.98	0.50	0.66	0.99	1.00	1.00
imagemagick	0.99	0.25	0.40	0.99	0.56	0.71	0.92	0.49	0.64	1.00	0.56	0.72	0.98	0.99	0.98
lighttpd	0.97	0.29	0.45	0.97	0.31	0.47	0.81	0.14	0.24	0.99	0.54	0.70	0.99	0.99	0.99
make	0.95	0.36	0.52	0.95	0.39	0.56	0.61	0.17	0.26	0.98	0.58	0.73	0.96	0.99	0.97
memcached	0.96	0.34	0.51	0.94	0.35	0.51	0.62	0.22	0.32	0.99	0.56	0.71	0.94	0.98	0.96
mkdir	0.96	0.17	0.28	0.97	0.24	0.39	0.56	0.06	0.11	0.99	0.53	0.69	0.98	0.99	0.99
nginx	0.98	0.33	0.49	0.98	0.35	0.52	0.84	0.17	0.29	0.99	0.55	0.71	0.97	0.98	0.97
nmap	0.97	0.29	0.44	0.98	0.39	0.56	0.14	0.30	0.19	0.99	0.46	0.63	0.97	0.97	0.97
objdump	0.94	0.27	0.42	0.95	0.31	0.47	0.53	0.16	0.25	0.99	0.54	0.70	0.95	0.99	0.97
poppler	0.99	0.36	0.53	0.99	0.44	0.60	0.11	0.17	0.13	1.00	0.38	0.55	0.99	0.96	0.98
rm	0.97	0.22	0.36	0.97	0.28	0.43	0.68	0.09	0.16	0.88	0.52	0.65	0.99	0.99	0.99
sort	0.93	0.27	0.41	0.94	0.31	0.47	0.75	0.13	0.22	0.99	0.46	0.62	0.99	0.99	0.99
tar	0.96	0.35	0.51	0.98	0.31	0.48	0.37	0.18	0.24	0.99	0.49	0.66	0.98	0.99	0.98
uniq	0.95	0.26	0.41	0.96	0.32	0.48	0.72	0.09	0.17	0.99	0.53	0.69	0.99	0.99	0.99
wget	N/A[1]	N/A[1]	N/A[1]	1.00	0.41	0.58	0.54	0.15	0.24	1.00	0.26	0.41	0.99	0.98	0.98
ml_train	N/A[2]	N/A[2]	N/A[2]	0.99	0.03	0.06	0.88	0.03	0.05	1.00	0.24	0.38	0.93	0.92	0.92

[1] Due to the significant slowdown introduced by `ltrace`, the `wget` benchmark repeatedly timed out and failed run correctly.
[2] The `ml_train` benchmark failed to run with `ltrace`.


RQ1
Measure bloat in ML containers
ML Container
ML containers are highly bloated.
RQ2
Debloat ML containers
ML shared libraries are a major source of bloat.
Host Code
Device Code
ML Shared Library
RQ3
Identify bloat in host code

RQ4: Debloat Host Code



- Zhang, H. & Ali-Eldin, A. MERGESHUFFLE: Reconstruct Shared Library for Improved Performance and Security. Under submission.

Problem Definition

Problem Definition

- Scenario

Problem Definition

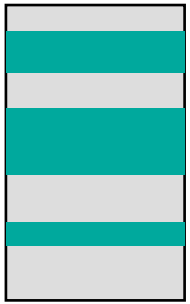
- Scenario
  - A shared library L_{in} provides many features. A(n) service(application) only uses a small subset.



L_{in}

Problem Definition

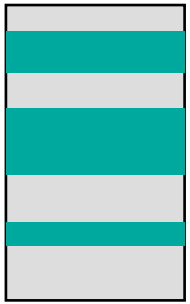
- Scenario
  - A shared library L_{in} provides many features. A(n) service(application) only uses a small subset.
  - The code ranges needed by the service(application) has been identified (e.g. using RTrace).



L_{in}

Problem Definition

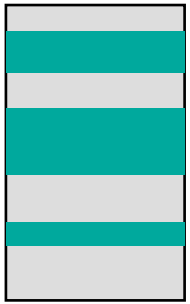
- Scenario
  - A shared library L_{in} provides many features. A(n) service(application) only uses a small subset.
  - The code ranges needed by the service(application) has been identified (e.g. using RTrace).
- Question:



L_{in}

Problem Definition

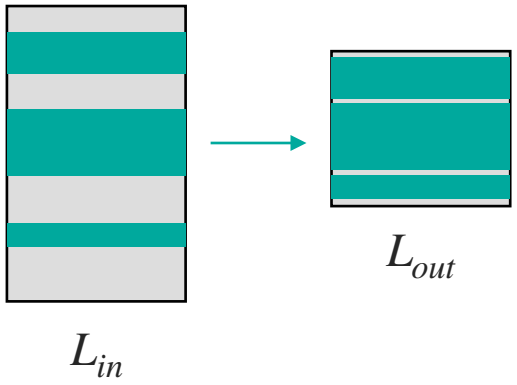
- Scenario
  - A shared library L_{in} provides many features. A(n) service(application) only uses a small subset.
  - The code ranges needed by the service(application) has been identified (e.g. using RTrace).
- Question:
  - How can we build a new library L_{out}, without access to the source code, containing only those required code ranges?



L_{in}

Problem Definition

- Scenario
  - A shared library L_{in} provides many features. A(n) service(application) only uses a small subset.
  - The code ranges needed by the service(application) has been identified (e.g. using RTrace).
- Question:
  - How can we build a new library L_{out}, without access to the source code, containing only those required code ranges?
  - Reduce file size, lower memory consumption, improve runtime performance, decrease attack surface?



Analysis of Library Loading Process

Analysis of Library Loading Process

- ELF Format
  - Library files follow the ELF format.

Analysis of Library Loading Process

- ELF Format
  - Library files follow the ELF format.
  - ELF has program headers, which define how file offsets map to memory.

```
Offset  VirtAddr FileSiz Align  ...
0x00000 0x00000  0x039e0 0x1000 ...
0x04000 0x04000  0x43519 0x1000 ...
0x48000 0x48000  0x398f0 0x1000 ...
...
```

An example of program headers.

Analysis of Library Loading Process

- ELF Format
  - Library files follow the ELF format.
  - ELF has program headers, which define how file offsets map to memory.
  - Library contents can be rearranged as long as program headers are updated accordingly.

```
Offset  VirtAddr FileSiz Align  ...
0x00000 0x00000  0x039e0 0x1000 ...
0x04000 0x04000  0x43519 0x1000 ...
0x48000 0x48000  0x398f0 0x1000 ...
...
```

An example of program headers.

MERGESHUFFLE Overview

Algorithm 1: MergeShuffle

Input : L_{in}: original shared library
R: $\{r_1, \ldots, r_n\}$, a list of file ranges
g: a threshold parameter

Output: L_{out}: reconstructed shared library

Replace all bytes in L_{in} outside R with `0xcc`;
`Merge(`R, g`)` ; `// (§3.3)`
$R' \leftarrow$ `Shuffle(`R, g`)` ; `// (§3.4)`
`// Create a new shared library`
Create L_{out} of size span(R'), filled with `0xcc`;
foreach $r_i \in R$ **do**
    Copy bytes from L_{in} at r_i to L_{out} at r'_i;
    Create a program header mapping: $r'_i \rightarrow r_i$;
end
return L_{out};

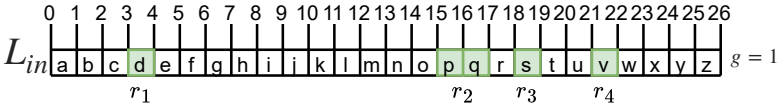
MERGESHUFFLE Overview

Algorithm 1: MergeShuffle

Input : L_{in}: original shared library
R: $\{r_1, \ldots, r_n\}$, a list of file ranges
g: a threshold parameter

Output: L_{out}: reconstructed shared library

1 Replace all bytes in L_{in} outside R with `0xcc`;
2 `Merge(`R, g`)` ; `// (§3.3)`
3 $R' \leftarrow$ `Shuffle(`R, g`)` ; `// (§3.4)`
`// Create a new shared library`
4 Create L_{out} of size span(R'), filled with `0xcc`;
5 **foreach** $r_i \in R$ **do**
6 Copy bytes from L_{in} at r_i to L_{out} at r'_i;
7 Create a program header mapping: $r'_i \rightarrow r_i$;
8 **end**
9 **return** L_{out};



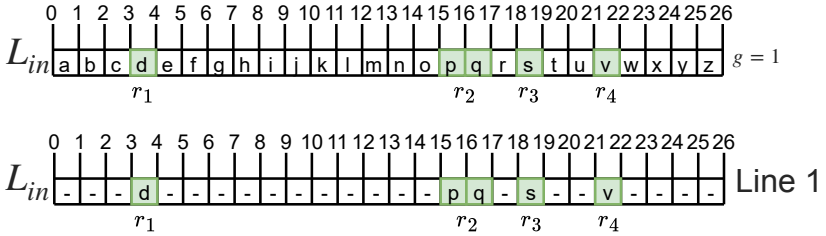
MERGESHUFFLE Overview

Algorithm 1: MergeShuffle

Input : L_{in}: original shared library
R: $\{r_1, \ldots, r_n\}$, a list of file ranges
g: a threshold parameter

Output: L_{out}: reconstructed shared library

1 Replace all bytes in L_{in} outside R with `0xcc`;
2 `Merge(`R, g`)` ; `// (§3.3)`
3 $R' \leftarrow$ `Shuffle(`R, g`)` ; `// (§3.4)`
`// Create a new shared library`
4 Create L_{out} of size span(R'), filled with `0xcc`;
5 **foreach** $r_i \in R$ **do**
6 Copy bytes from L_{in} at r_i to L_{out} at r'_i;
7 Create a program header mapping: $r'_i \rightarrow r_i$;
8 **end**
9 **return** L_{out};



MERGESHUFFLE Overview

Algorithm 1: MergeShuffle

Input : L_{in}: original shared library
R: $\{r_1, \ldots, r_n\}$, a list of file ranges
g: a threshold parameter

Output: L_{out}: reconstructed shared library

1 Replace all bytes in L_{in} outside R with `0xcc`;
2 `Merge(`R, g`)` ; `// (§3.3)`
3 $R' \leftarrow$ `Shuffle(`R, g`)` ; `// (§3.4)`
`// Create a new shared library`
4 Create L_{out} of size span(R'), filled with `0xcc`;
5 **foreach** $r_i \in R$ **do**
6 Copy bytes from L_{in} at r_i to L_{out} at r'_i;
7 Create a program header mapping: $r'_i \rightarrow r_i$;
8 **end**
9 **return** L_{out};



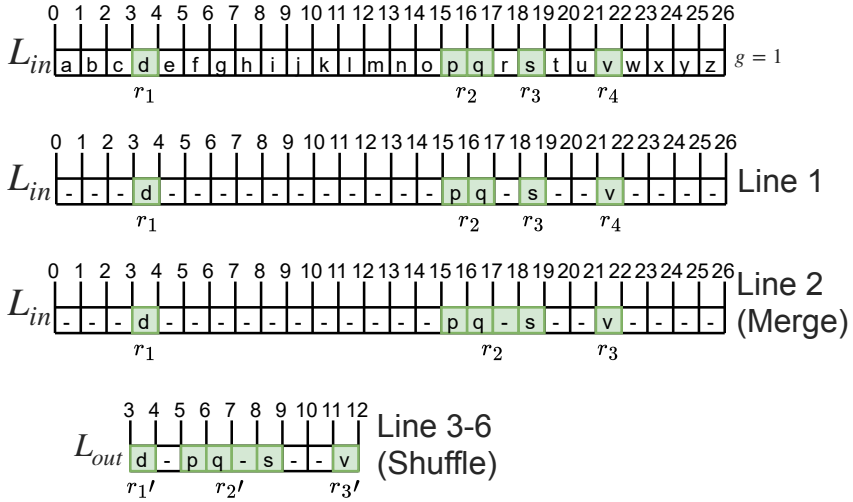
MERGESHUFFLE Overview

Algorithm 1: MergeShuffle

Input : L_{in}: original shared library
R: $\{r_1, \ldots, r_n\}$, a list of file ranges
g: a threshold parameter

Output: L_{out}: reconstructed shared library

1 Replace all bytes in L_{in} outside R with `0xcc`;
2 `Merge(`R, g`)` ; `// (§3.3)`
3 $R' \leftarrow$ `Shuffle(`R, g`)` ; `// (§3.4)`
`// Create a new shared library`
4 Create L_{out} of size span(R'), filled with `0xcc`;
5 **foreach** $r_i \in R$ **do**
6 Copy bytes from L_{in} at r_i to L_{out} at r'_i;
7 Create a program header mapping: $r'_i \rightarrow r_i$;
8 **end**
9 **return** L_{out};



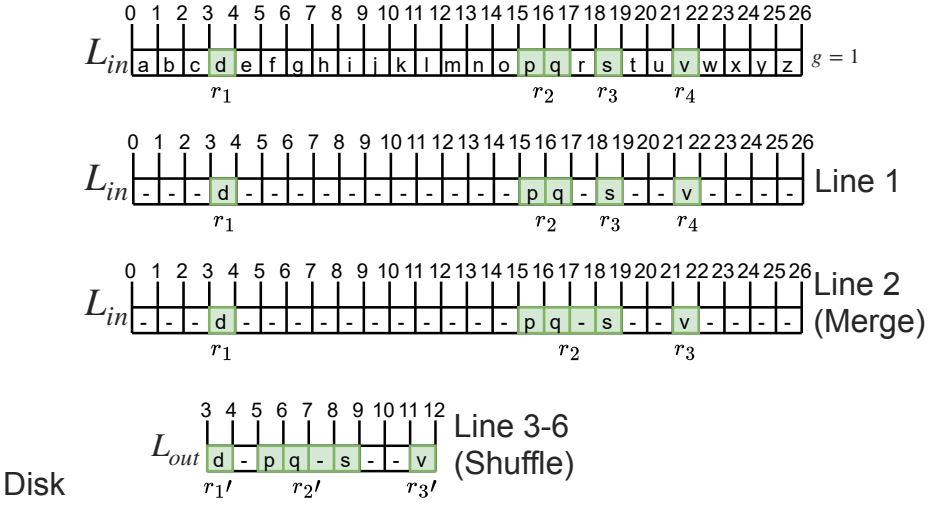
MERGESHUFFLE Overview

Algorithm 1: MergeShuffle

Input : L_{in}: original shared library
R: $\{r_1, \ldots, r_n\}$, a list of file ranges
g: a threshold parameter

Output: L_{out}: reconstructed shared library

1 Replace all bytes in L_{in} outside R with `0xcc`;
2 `Merge(`R, g`)` ; `// (§3.3)`
3 $R' \leftarrow$ `Shuffle(`R, g`)` ; `// (§3.4)`
`// Create a new shared library`
4 Create L_{out} of size span(R'), filled with `0xcc`;
5 **foreach** $r_i \in R$ **do**
6 Copy bytes from L_{in} at r_i to L_{out} at r'_i;
7 Create a program header mapping: $r'_i \rightarrow r_i$;
8 **end**
9 **return** L_{out};



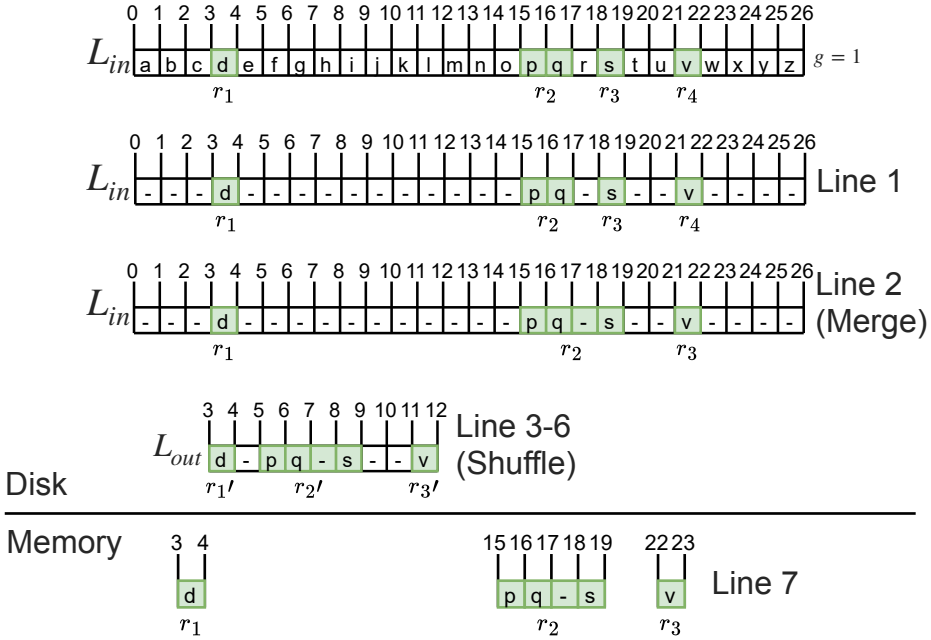
MERGESHUFFLE Overview

Algorithm 1: MergeShuffle

Input : L_{in}: original shared library
R: $\{r_1, \ldots, r_n\}$, a list of file ranges
g: a threshold parameter

Output: L_{out}: reconstructed shared library

1 Replace all bytes in L_{in} outside R with `0xcc`;
2 `Merge(`R, g`)` ; `// (§3.3)`
3 $R' \leftarrow$ `Shuffle(`R, g`)` ; `// (§3.4)`
`// Create a new shared library`
4 Create L_{out} of size span(R'), filled with `0xcc`;
5 **foreach** $r_i \in R$ **do**
6 Copy bytes from L_{in} at r_i to L_{out} at r_i';
7 Create a program header mapping: $r_i' \rightarrow r_i$;
8 **end**
9 **return** L_{out};



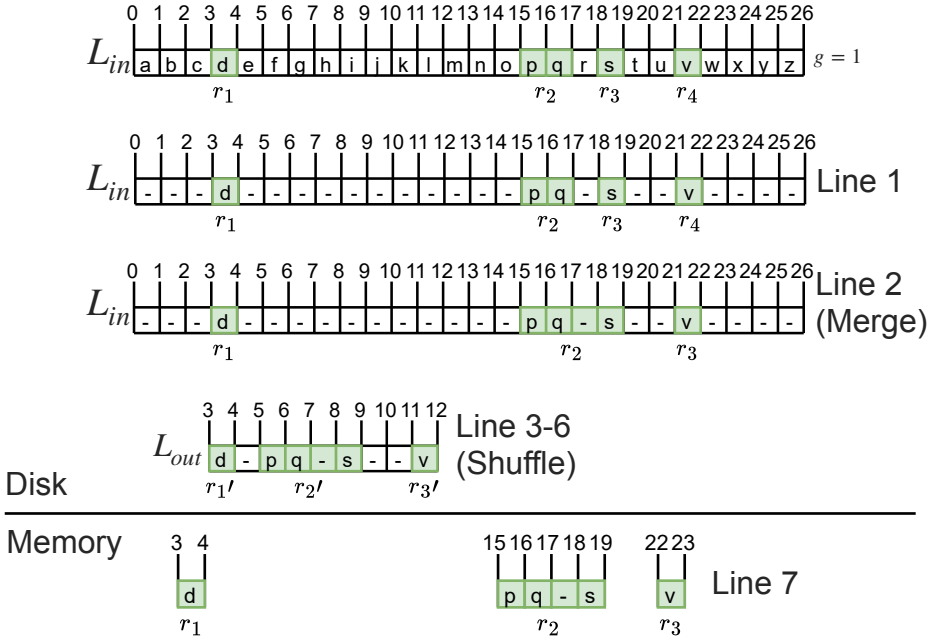
MERGESHUFFLE Overview

Algorithm 1: MergeShuffle

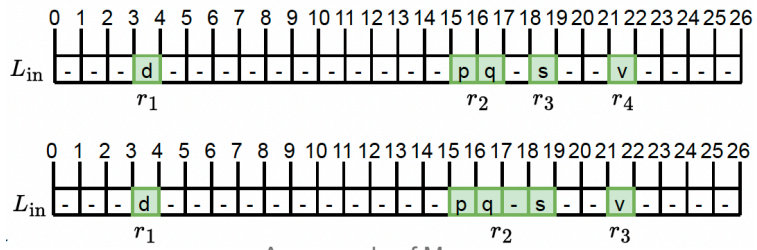
Input : L_{in}: original shared library
R: $\{r_1, \ldots, r_n\}$, a list of file ranges
g: a threshold parameter

Output: L_{out}: reconstructed shared library

1 Replace all bytes in L_{in} outside R with `0xcc`;
2 `Merge(`R, g`)` ; `// (§3.3)`
3 $R' \leftarrow$ `Shuffle(`R, g`)` ; `// (§3.4)`
`// Create a new shared library`
4 Create L_{out} of size span(R'), filled with `0xcc`;
5 **foreach** $r_i \in R$ **do**
6 Copy bytes from L_{in} at r_i to L_{out} at r_i';
7 Create a program header mapping: $r_i' \rightarrow r_i$;
8 **end**
9 **return** L_{out};



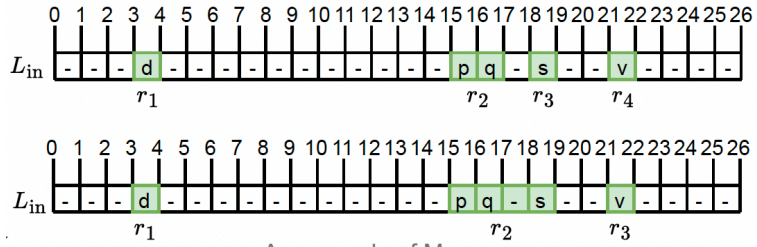
Merge: Balance Between Memory Usage and Run Time



An example of Merge.

Merge: Balance Between Memory Usage and Run Time

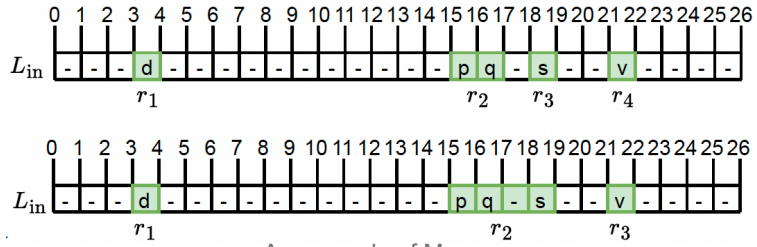
- **Strategy**: If two adjacent code ranges have a gap≤threshold $g \rightarrow$ merge into one program header.



An example of Merge.

Merge: Balance Between Memory Usage and Run Time

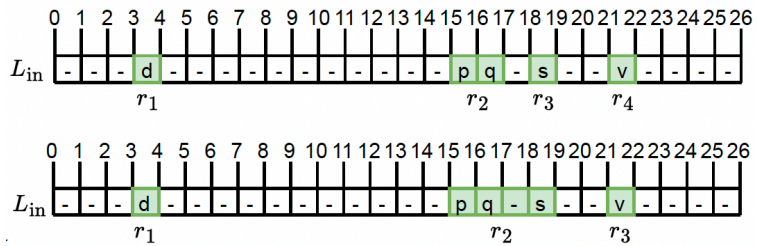
- **Strategy**: If two adjacent code ranges have a gap≤threshold $g \rightarrow$ merge into one program header.
- **Trade-off**: Mapped a few unused bytes.



An example of Merge.

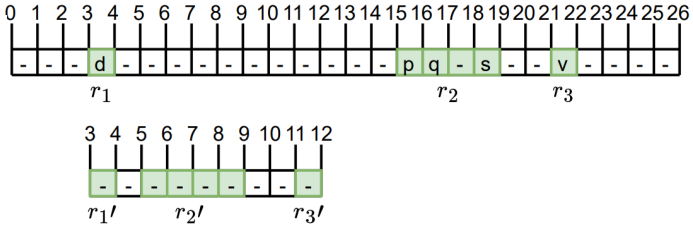
Merge: Balance Between Memory Usage and Run Time

- **Strategy**: If two adjacent code ranges have a gap≤threshold $g \rightarrow$ merge into one program header.
- **Trade-off**: Mapped a few unused bytes.
- **Benefit**: Reduced library loading time.



An example of Merge.

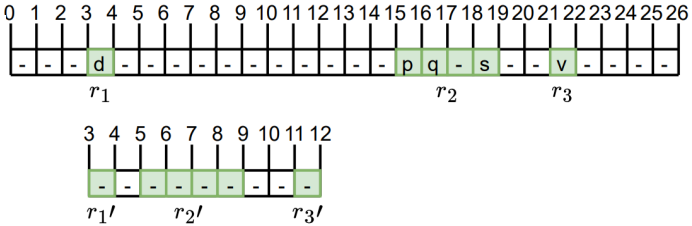
Shuffle: Reduce File Size and Increase Memory Locality under Congruence Constraint



An example of Shuffle.

Shuffle: Reduce File Size and Increase Memory Locality under Congruence Constraint

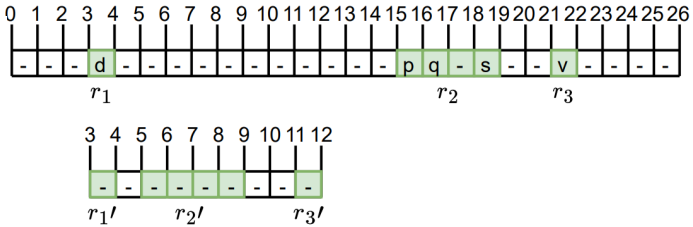
- **Strategy**: Move retained code ranges closer together.



An example of Shuffle.

Shuffle: Reduce File Size and Increase Memory Locality under Congruence Constraint

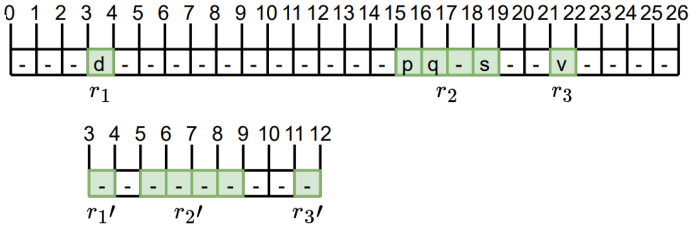
- **Strategy**: Move retained code ranges closer together.
- **Constraint**: The new position must have the same modulo (page size P) as the original location.



An example of Shuffle.

Shuffle: Reduce File Size and Increase Memory Locality under Congruence Constraint

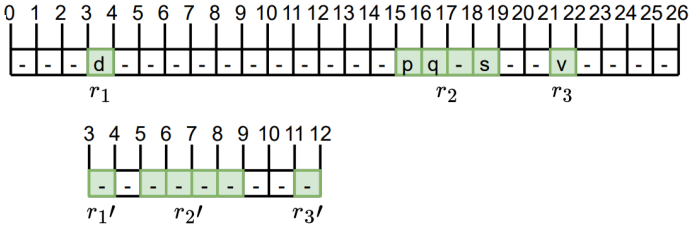
- **Strategy**: Move retained code ranges closer together.
- **Constraint**: The new position must have the same modulo (page size P) as the original location.
- **Approaches**:



An example of Shuffle.

Shuffle: Reduce File Size and Increase Memory Locality under Congruence Constraint

- **Strategy**: Move retained code ranges closer together.
- **Constraint**: The new position must have the same modulo (page size P) as the original location.
- **Approaches**:
  - LinearShuffle & TSPShuffle.



An example of Shuffle.

Results of Evaluation

Overview of number of libraries, reductions on size, memory and run time of applications running using original and reconstructed shared libraries. (g=0)

Application	#Lib.	Size(%)	Memory(%)	Run Time(%)
Torch-Inference	7	59.7	16.3	32.4
Torch-Train	7	59.4	7.7	1.4
Imagemagick	8	40.4	1.4	10.3
Nmap	5	38.7	3.5	-0.1
Poppler	12	34.2	12.0	2.8
ComposePostService	9	37.2	15.6	9.9
HomeTimelineService	9	37.2	15.2	7.8
TextService	9	37.2	15.7	11.3
UserService	24	35.9	18.4	6.3

Results of Evaluation

- Relative file size: 34%~59%.

Overview of number of libraries, reductions on size, memory and run time of applications running using original and reconstructed shared libraries. (g=0)

Application	#Lib.	Size(%)	Memory(%)	Run Time(%)
Torch-Inference	7	59.7	16.3	32.4
Torch-Train	7	59.4	7.7	1.4
Imagemagick	8	40.4	1.4	10.3
Nmap	5	38.7	3.5	-0.1
Poppler	12	34.2	12.0	2.8
ComposePostService	9	37.2	15.6	9.9
HomeTimelineService	9	37.2	15.2	7.8
TextService	9	37.2	15.7	11.3
UserService	24	35.9	18.4	6.3

Results of Evaluation

- Relative file size: 34%~59%.
- Relative memory usage: 1%~18%.

Overview of number of libraries, reductions on size, memory and run time of applications running using original and reconstructed shared libraries. (g=0)

Application	#Lib.	Size(%)	Memory(%)	Run Time(%)
Torch-Inference	7	59.7	16.3	32.4
Torch-Train	7	59.4	7.7	1.4
Imagemagick	8	40.4	1.4	10.3
Nmap	5	38.7	3.5	-0.1
Poppler	12	34.2	12.0	2.8
ComposePostService	9	37.2	15.6	9.9
HomeTimelineService	9	37.2	15.2	7.8
TextService	9	37.2	15.7	11.3
UserService	24	35.9	18.4	6.3

Results of Evaluation

- Relative file size: 34%~59%.
- Relative memory usage: 1%~18%.
- Relative run time(cold start): 0%~32%.

Overview of number of libraries, reductions on size, memory and run time of applications running using original and reconstructed shared libraries. (g=0)

Application	#Lib.	Size(%)	Memory(%)	Run Time(%)
Torch-Inference	7	59.7	16.3	32.4
Torch-Train	7	59.4	7.7	1.4
Imagemagick	8	40.4	1.4	10.3
Nmap	5	38.7	3.5	-0.1
Poppler	12	34.2	12.0	2.8
ComposePostService	9	37.2	15.6	9.9
HomeTimelineService	9	37.2	15.2	7.8
TextService	9	37.2	15.7	11.3
UserService	24	35.9	18.4	6.3

Impacts on Security

- Removing unused code can significantly reduce the number of gadgets in shared libraries, with a relative reduction ranging from 47% to 79%.

Gadget count (and reduction) after reconstructing. $g = 0$.

Application	ROP Gadgets	JOP Gadgets	COP Gadgets	S.P. Gadgets	Total
Torch-Inference*	8451(74.5%)	3358(63.6%)	6930(71.4%)	1317(53.2%)	20056(70.2%)
Torch-Train*	8451(75.4%)	3358(64.5%)	6930(72.0%)	1317(54.3%)	20056(71.0%)
Imagemagick	10839(49.7%)	4248(52.2%)	5030(37.2%)	2658(40.3%)	22775(46.3%)
Nmap	16048(77.4%)	2936(86.0%)	7110(79.0%)	2627(83.0%)	28721(79.2%)
Poppler	16097(74.1%)	4372(79.5%)	14760(69.5%)	1692(74.5%)	36921(72.9%)
ComposePostService	16610(62.9%)	8870(35.7%)	17310(55.9%)	5313(41.7%)	48103(53.0%)
HomeTimelineService	16610(62.9%)	8870(35.7%)	17310(56.0%)	5313(41.7%)	48103(53.1%)
TextService	16610(62.9%)	8870(35.7%)	17310(55.9%)	5313(41.6%)	48103(53.0%)
UserService	23062(55.7%)	12371(32.1%)	24935(48.9%)	6832(39.4%)	67200(47.2%)

* `libtorch_cpu` is excluded as the gadget analyzer fails to report results.

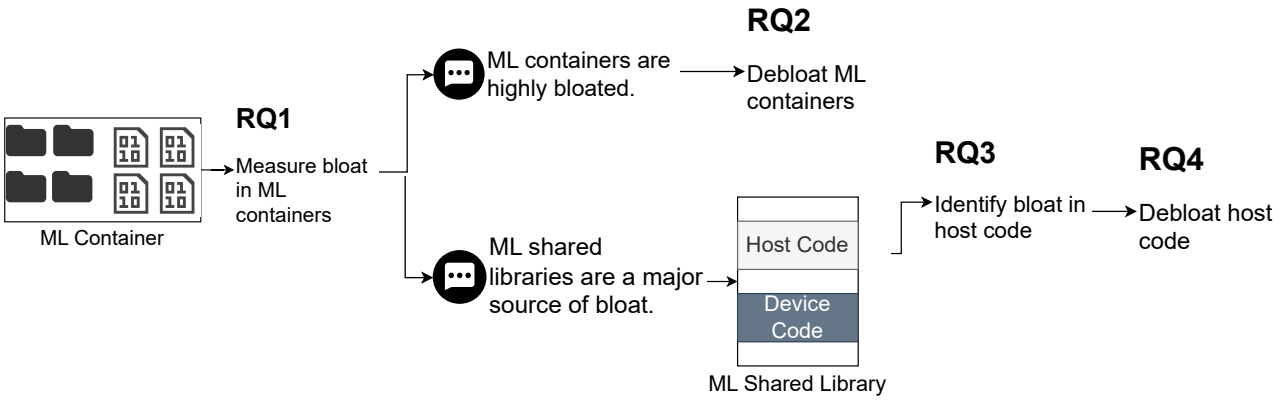
Impacts on Security

- Removing unused code can significantly reduce the number of gadgets in shared libraries, with a relative reduction ranging from 47% to 79%.

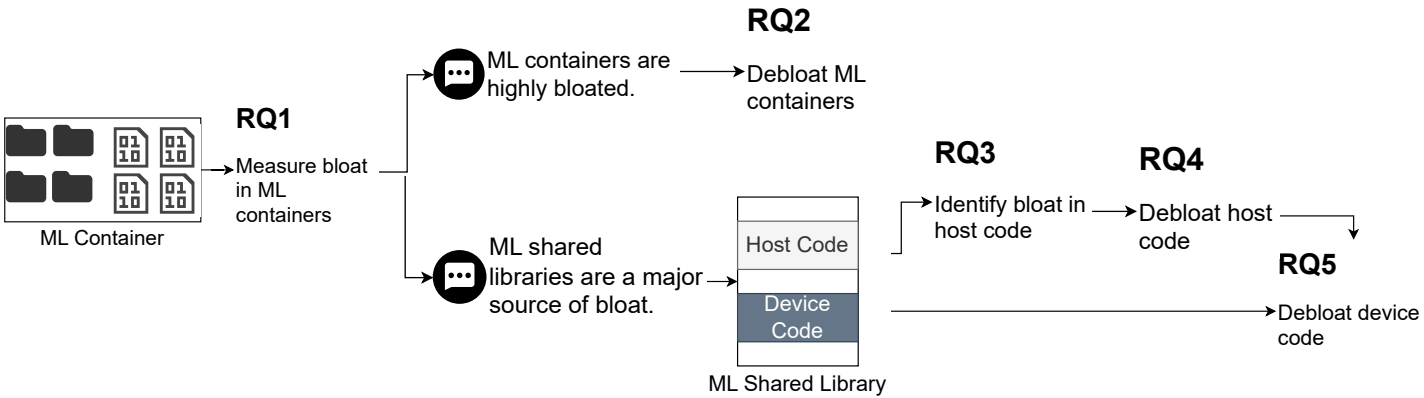
Gadget count (and reduction) after reconstructing. $g = 0$.

Application	ROP Gadgets	JOP Gadgets	COP Gadgets	S.P. Gadgets	Total
Torch-Inference*	8451(74.5%)	3358(63.6%)	6930(71.4%)	1317(53.2%)	20056(70.2%)
Torch-Train*	8451(75.4%)	3358(64.5%)	6930(72.0%)	1317(54.3%)	20056(71.0%)
Imagemagick	10839(49.7%)	4248(52.2%)	5030(37.2%)	2658(40.3%)	22775(46.3%)
Nmap	16048(77.4%)	2936(86.0%)	7110(79.0%)	2627(83.0%)	28721(79.2%)
Poppler	16097(74.1%)	4372(79.5%)	14760(69.5%)	1692(74.5%)	36921(72.9%)
ComposePostService	16610(62.9%)	8870(35.7%)	17310(55.9%)	5313(41.7%)	48103(53.0%)
HomeTimelineService	16610(62.9%)	8870(35.7%)	17310(56.0%)	5313(41.7%)	48103(53.1%)
TextService	16610(62.9%)	8870(35.7%)	17310(55.9%)	5313(41.6%)	48103(53.0%)
UserService	23062(55.7%)	12371(32.1%)	24935(48.9%)	6832(39.4%)	67200(47.2%)

* `libtorch_cpu` is excluded as the gadget analyzer fails to report results.


RQ1
Measure bloat in ML containers
ML Container
ML containers are highly bloated.
RQ2
Debloat ML containers
ML shared libraries are a major source of bloat.
Host Code
Device Code
ML Shared Library
RQ3
Identify bloat in host code
RQ4
Debloat host code

What about Device Code



- Zhang, H. & Ali-Eldin, A. (2025). The Hidden Bloat in Machine Learning Systems. Proceedings of the 8th Conference on Machine Learning and Systems (MLSys). (**Best Paper Award**)

Machine Learning Frameworks

- Machine learning frameworks like TensorFlow, PyTorch, vLLM, and SGLang are at the core of machine learning systems.
- While these frameworks offer convenience, they also introduce additional performance overhead—known as the *Framework Tax*.
- Example: These frameworks rely on C++, CUDA, and Python, resulting in large binary files that are not suitable for mobile and edge devices.

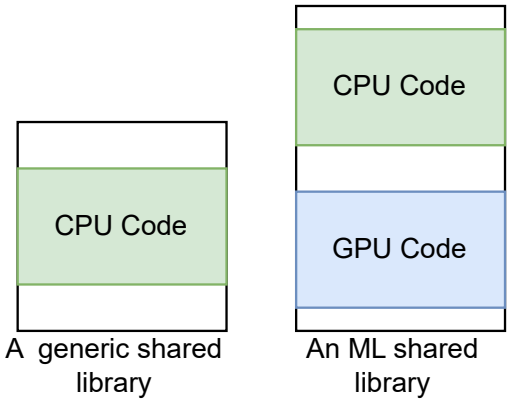
Ref: Fernandez, et. al. (2023, December). The Framework Tax: Disparities Between Inference Efficiency in NLP Research and Deployment. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (pp. 1588-1600).

Shared libraries in ML Frameworks

- The core (kernel) code of machine learning frameworks is mostly written in C++/CUDA and packaged as shared libraries.
- For example, in PyTorch and TensorFlow, shared libraries account for 93% and 75% of the total framework size, respectively.

ML Shared Libraries: CPU Code and GPU Code

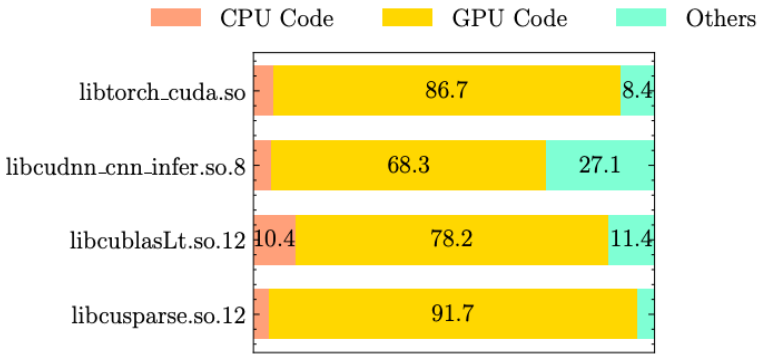
- The uniqueness of ML shared libraries is that they contain both CPU code and GPU code.
- CPU code runs on CPUs.
- GPU Code runs on GPUs.



ML Shared Libraries: CPU Code and GPU Code

ML Shared Libraries: CPU Code and GPU Code

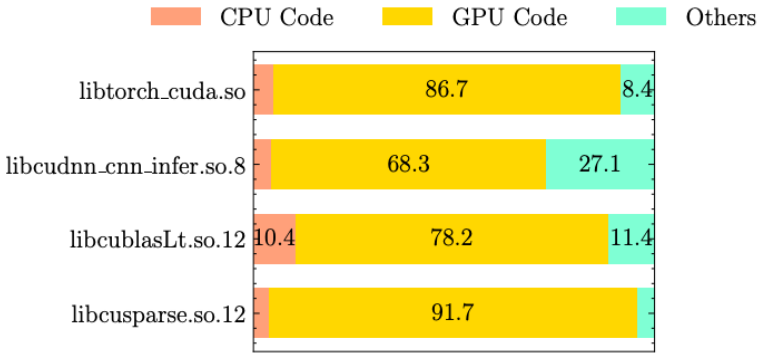
- In PyTorch, GPU code accounts for between 68% to more than 91% of the size of each shared library.



Size distribution of CPU code and GPU code in the top 4 largest shared libraries in PyTorch. 2.2

ML Shared Libraries: CPU Code and GPU Code

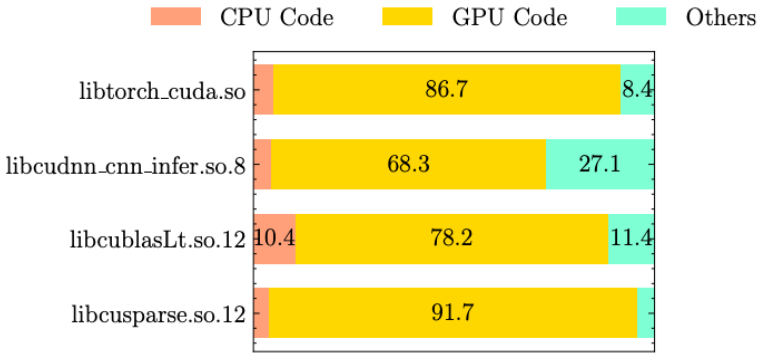
- In PyTorch, GPU code accounts for between 68% to more than 91% of the size of each shared library.
- In contrast, CPU code represents only a small fraction of these libraries.



Size distribution of CPU code and GPU code in the top 4 largest shared libraries in PyTorch. 2.2

ML Shared Libraries: CPU Code and GPU Code

- In PyTorch, GPU code accounts for between 68% to more than 91% of the size of each shared library.
- In contrast, CPU code represents only a small fraction of these libraries.
- Despite GPU code forming the majority of ML shared libraries, no existing research has investigated bloat within GPU code.



Size distribution of CPU code and GPU code in the top 4 largest shared libraries in PyTorch. 2.2

Debloat GPU Code in ML Shared Libraries

Debloat GPU Code in ML Shared Libraries

- Challenges:

Debloat GPU Code in ML Shared Libraries

- Challenges:
    - Some ML shared libraries are not open-sourced, making it impossible to perform source code analysis.

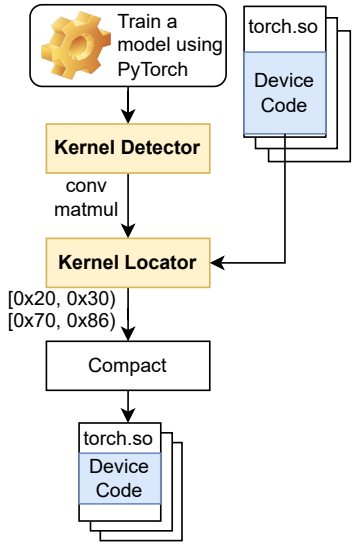
Debloat GPU Code in ML Shared Libraries

- Challenges:
    - Some ML shared libraries are not open-sourced, making it impossible to perform source code analysis.
    - ML shared libraries contain code that runs on both the CPU and GPU. GPU code is overlooked by existing research and no method measure and reduce bloat in it.

Debloat GPU Code in ML Shared Libraries

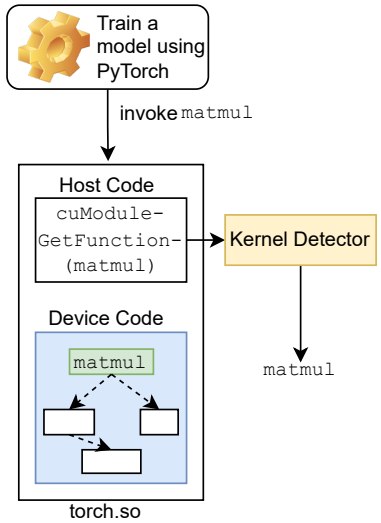
- Challenges:
    - Some ML shared libraries are not open-sourced, making it impossible to perform source code analysis.
    - ML shared libraries contain code that runs on both the CPU and GPU. GPU code is overlooked by existing research and no method measure and reduce bloat in it.
    - Although CPU code has a well-defined structure and the bloat in it has been previously studied, the structure of GPU code is under-explored.

Negativa-ML: Debloating Device Code



Overview of Negativa-ML

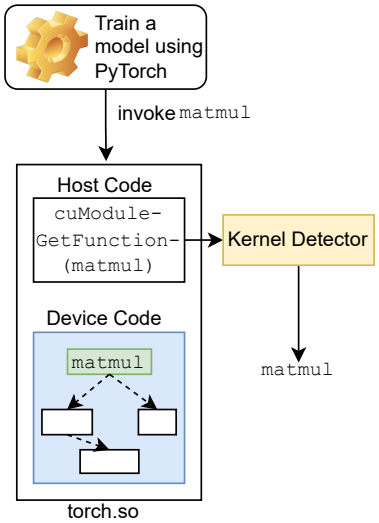
Kernel Detector: Detect Used Kernels



Kernel detector workflow.

Kernel Detector: Detect Used Kernels

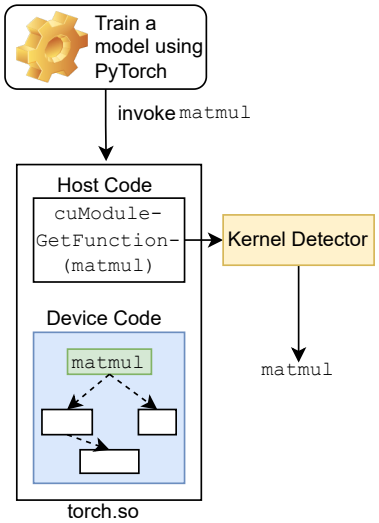
- To launch a kernel from the host, the CUDA driver must call the host function cuModuleGetFunction first.



Kernel detector workflow.

Kernel Detector: Detect Used Kernels

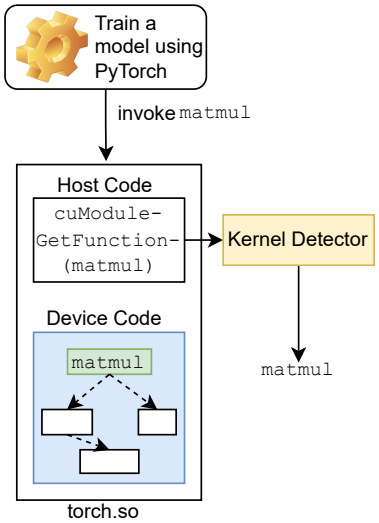
- To launch a kernel from the host, the CUDA driver must call the host function cuModuleGetFunction first.
- The function takes the name of the kernel to be launched as one of its inputs.



Kernel detector workflow.

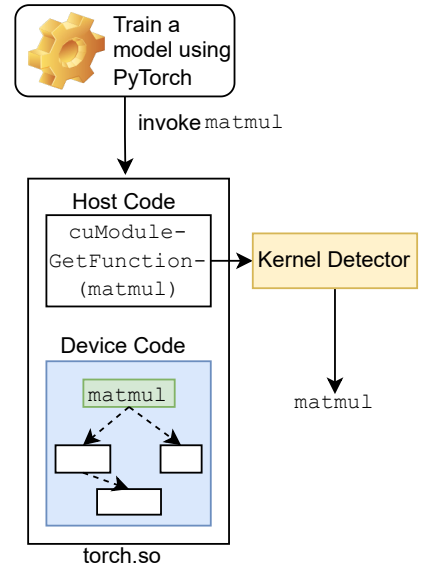
Kernel Detector: Detect Used Kernels

- To launch a kernel from the host, the CUDA driver must call the host function cuModuleGetFunction first.
- The function takes the name of the kernel to be launched as one of its inputs.
- It is only called once for each kernel.



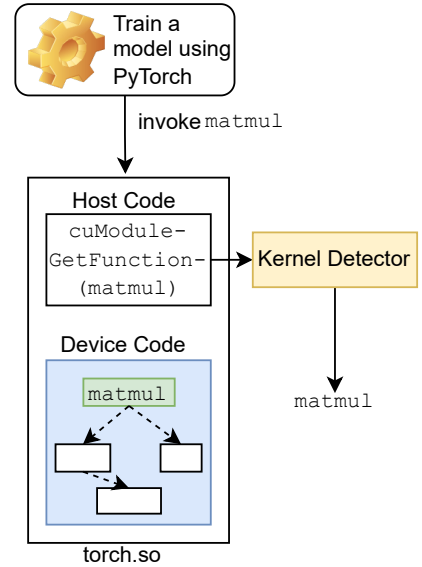
Kernel detector workflow.

Kernel Locator: Locate File Ranges of Used Kernels

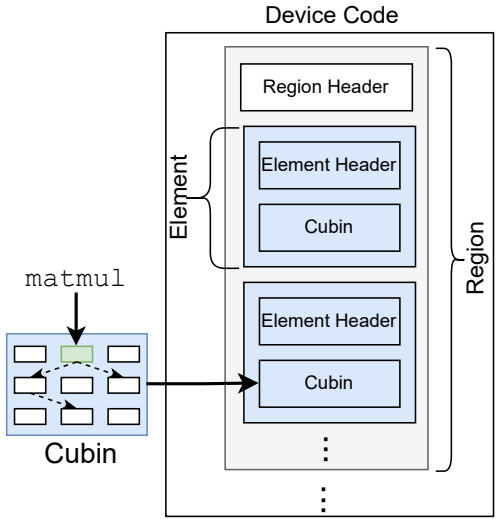


Kernel detector workflow.

Kernel Locator: Locate File Ranges of Used Kernels



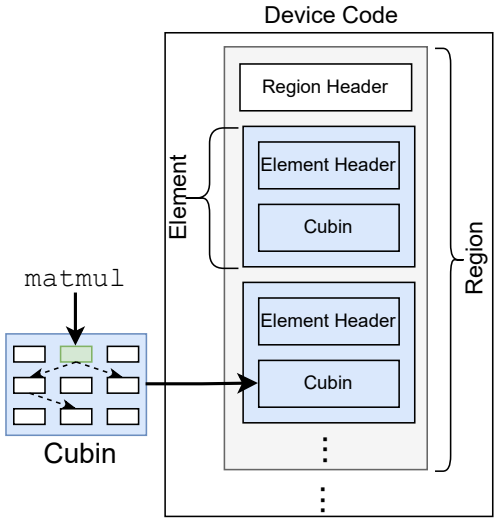
Kernel detector workflow.



Structure of device code.

Kernel Locator: Locate File Ranges of Used Kernels

- If a cubin matches the GPU architecture and contains used host-launching kernels, then we retain the cubin. The other cubins are removed.



Structure of device code.

Evaluation: Code Size Reduction

Total file size, host code, device code and their reductions of all shared libraries in each ML framework. The table shows the original value of a metric and the reduction in percentage of the metric in parentheses. K=1,000.

Model	Framework	Operation	#Lib.	Total File Size/MB	Host Code		Device Code	
					Size/MB	#Functions	Size/MB	#Elements
MobileNetV2	PyTorch	Train	113	3,762 (55)	557 (68)	616K (93)	2,279 (75)	14,062 (98)
		Inference	111	3,569 (55)	545 (70)	616K (93)	2,103 (75)	12,035 (98)
	TensorFlow	Train	253	3,274 (48)	598 (46)	984K (65)	1,774 (73)	15,081 (99)
		Inference	251	3,087 (47)	586 (48)	984K (65)	1,603 (72)	13,056 (99)
Transformer	PyTorch	Train	154	2,901 (53)	547 (71)	615K (93)	1,592 (72)	7,165 (97)
		Inference	154	2,901 (53)	547 (71)	615K (93)	1,592 (73)	7,165 (98)
	TensorFlow	Train	398	2,727 (42)	696 (46)	1,043K (66)	1,184 (70)	8,478 (98)
		Inference	396	2,640 (40)	692 (47)	1,042K (67)	1,103 (66)	8,325 (97)
LLama2	vLLM	Inference	170	3,884 (48)	724 (68)	873K (93)	1,901 (72)	7,690 (97)
	Transformers	Inference	98	2,860 (53)	511 (72)	582K (92)	1,592 (71)	7,165 (97)

Evaluation: Code Size Reduction

Total file size, host code, device code and their reductions of all shared libraries in each ML framework. The table shows the original value of a metric and the reduction in percentage of the metric in parentheses. K=1,000.

Model	Framework	Operation	#Lib.	Total File Size/MB	Host Code		Device Code	
					Size/MB	#Functions	Size/MB	#Elements
MobileNetV2	PyTorch	Train	113	3,762 (55)	557 (68)	616K (93)	2,279 (75)	14,062 (98)
		Inference	111	3,569 (55)	545 (70)	616K (93)	2,103 (75)	12,035 (98)
	TensorFlow	Train	253	3,274 (48)	598 (46)	984K (65)	1,774 (73)	15,081 (99)
		Inference	251	3,087 (47)	586 (48)	984K (65)	1,603 (72)	13,056 (99)
Transformer	PyTorch	Train	154	2,901 (53)	547 (71)	615K (93)	1,592 (72)	7,165 (97)
		Inference	154	2,901 (53)	547 (71)	615K (93)	1,592 (73)	7,165 (98)
	TensorFlow	Train	398	2,727 (42)	696 (46)	1,043K (66)	1,184 (70)	8,478 (98)
		Inference	396	2,640 (40)	692 (47)	1,042K (67)	1,103 (66)	8,325 (97)
LLama2	vLLM	Inference	170	3,884 (48)	724 (68)	873K (93)	1,901 (72)	7,690 (97)
	Transformers	Inference	98	2,860 (53)	511 (72)	582K (92)	1,592 (71)	7,165 (97)

- All ML frameworks show substantial reductions in both host code (≥ 46%) and device code (≥ 66%).

Evaluation: Code Size Reduction

Total file size, host code, device code and their reductions of all shared libraries in each ML framework. The table shows the original value of a metric and the reduction in percentage of the metric in parentheses. K=1,000.

Model	Framework	Operation	#Lib.	Total File Size/MB	Host Code		Device Code	
					Size/MB	#Functions	Size/MB	#Elements
MobileNetV2	PyTorch	Train	113	3,762 (55)	557 (68)	616K (93)	2,279 (75)	14,062 (98)
		Inference	111	3,569 (55)	545 (70)	616K (93)	2,103 (75)	12,035 (98)
	TensorFlow	Train	253	3,274 (48)	598 (46)	984K (65)	1,774 (73)	15,081 (99)
		Inference	251	3,087 (47)	586 (48)	984K (65)	1,603 (72)	13,056 (99)
Transformer	PyTorch	Train	154	2,901 (53)	547 (71)	615K (93)	1,592 (72)	7,165 (97)
		Inference	154	2,901 (53)	547 (71)	615K (93)	1,592 (73)	7,165 (98)
	TensorFlow	Train	398	2,727 (42)	696 (46)	1,043K (66)	1,184 (70)	8,478 (98)
		Inference	396	2,640 (40)	692 (47)	1,042K (67)	1,103 (66)	8,325 (97)
LLama2	vLLM	Inference	170	3,884 (48)	724 (68)	873K (93)	1,901 (72)	7,690 (97)
	Transformers	Inference	98	2,860 (53)	511 (72)	582K (92)	1,592 (71)	7,165 (97)

- All ML frameworks show substantial reductions in both host code (≥ 46%) and device code (≥ 66%).
- Device code is much larger and more bloated than host code, contributing to the majority of the bloat within the ML frameworks.

Evaluation: Run Time Improvements

Average runtime performance using original shared libraries and reductions using debloated shared libraries. The numbers in parentheses are the percentage of reduction. The standard deviation of all metrics for each workload is less than 2% and is not shown.

Model	Framework	Operation	Peak Host Memory/MB	Peak GPU Memory/MB	Execution Time/s
`MobileNetV2`	PyTorch	Train	5,487 (64.2)	1,539 (48.1)	179 (2.3)
		Inference	4,943 (74.6)	972 (69.6)	8 (44.6)
	TensorFlow	Train	6,009 (48.7)	14,395 (2.8)	53 (5.5)
		Inference	4,850 (60.0)	14,323 (2.8)	12 (24.2)
`Transformer`	PyTorch	Train	4,151 (56.0)	9,381 (5.0)	200 (1.1)
		Inference	4,054 (65.8)	1,349 (42.3)	13 (20.3)
	TensorFlow	Train	15,652 (13.6)	14,149 (1.6)	4,779 (0.0)
		Inference	4,217 (36.5)	14,069 (1.3)	69 (1.8)
`Llama2`	vLLM	Inference	12,527 (11.8)	14,679 (2.1)	43 (12.7)
	Transformers	Inference	12,065 (10.4)	13,793 (3.2)	21 (8.9)
Average Absolute Reduction$\pm\sigma$			2501$\pm$825	443 $\pm$171	2.6$\pm$1.6

Evaluation: Run Time Improvements

Average runtime performance using original shared libraries and reductions using debloated shared libraries. The numbers in parentheses are the percentage of reduction. The standard deviation of all metrics for each workload is less than 2% and is not shown.

Model	Framework	Operation	Peak Host Memory/MB	Peak GPU Memory/MB	Execution Time/s
MobileNetV2	PyTorch	Train	5,487 (64.2)	1,539 (48.1)	179 (2.3)
		Inference	4,943 (74.6)	972 (69.6)	8 (44.6)
	TensorFlow	Train	6,009 (48.7)	14,395 (2.8)	53 (5.5)
		Inference	4,850 (60.0)	14,323 (2.8)	12 (24.2)
Transformer	PyTorch	Train	4,151 (56.0)	9,381 (5.0)	200 (1.1)
		Inference	4,054 (65.8)	1,349 (42.3)	13 (20.3)
	TensorFlow	Train	15,652 (13.6)	14,149 (1.6)	4,779 (0.0)
		Inference	4,217 (36.5)	14,069 (1.3)	69 (1.8)
Llama2	vLLM	Inference	12,527 (11.8)	14,679 (2.1)	43 (12.7)
	Transformers	Inference	12,065 (10.4)	13,793 (3.2)	21 (8.9)
Average Absolute Reduction$\pm\sigma$			2501$\pm$825	443 $\pm$171	2.6$\pm$1.6

- Debloating significantly reduces CPU/GPU memory usage and execution time for small model inference.

Evaluation: Run Time Improvements

Average runtime performance using original shared libraries and reductions using debloated shared libraries. The numbers in parentheses are the percentage of reduction. The standard deviation of all metrics for each workload is less than 2% and is not shown.

Model	Framework	Operation	Peak Host Memory/MB	Peak GPU Memory/MB	Execution Time/s
MobileNetV2	PyTorch	Train	5,487 (64.2)	1,539 (48.1)	179 (2.3)
		Inference	4,943 (74.6)	972 (69.6)	8 (44.6)
	TensorFlow	Train	6,009 (48.7)	14,395 (2.8)	53 (5.5)
		Inference	4,850 (60.0)	14,323 (2.8)	12 (24.2)
Transformer	PyTorch	Train	4,151 (56.0)	9,381 (5.0)	200 (1.1)
		Inference	4,054 (65.8)	1,349 (42.3)	13 (20.3)
	TensorFlow	Train	15,652 (13.6)	14,149 (1.6)	4,779 (0.0)
		Inference	4,217 (36.5)	14,069 (1.3)	69 (1.8)
Llama2	vLLM	Inference	12,527 (11.8)	14,679 (2.1)	43 (12.7)
	Transformers	Inference	12,065 (10.4)	13,793 (3.2)	21 (8.9)
Average Absolute Reduction$\pm\sigma$			2501$\pm$825	443 $\pm$171	2.6$\pm$1.6

- Debloating significantly reduces CPU/GPU memory usage and execution time for small model inference.
- Debloating shows smaller gains for larger models and training workloads.

Evaluation: Run Time Improvements

Average runtime performance using original shared libraries and reductions using debloated shared libraries. The numbers in parentheses are the percentage of reduction. The standard deviation of all metrics for each workload is less than 2% and is not shown.

Model	Framework	Operation	Peak Host Memory/MB	Peak GPU Memory/MB	Execution Time/s
MobileNetV2	PyTorch	Train	5,487 (64.2)	1,539 (48.1)	179 (2.3)
		Inference	4,943 (74.6)	972 (69.6)	8 (44.6)
	TensorFlow	Train	6,009 (48.7)	14,395 (2.8)	53 (5.5)
		Inference	4,850 (60.0)	14,323 (2.8)	12 (24.2)
Transformer	PyTorch	Train	4,151 (56.0)	9,381 (5.0)	200 (1.1)
		Inference	4,054 (65.8)	1,349 (42.3)	13 (20.3)
	TensorFlow	Train	15,652 (13.6)	14,149 (1.6)	4,779 (0.0)
		Inference	4,217 (36.5)	14,069 (1.3)	69 (1.8)
Llama2	vLLM	Inference	12,527 (11.8)	14,679 (2.1)	43 (12.7)
	Transformers	Inference	12,065 (10.4)	13,793 (3.2)	21 (8.9)
Average Absolute Reduction$\pm\sigma$			2501$\pm$825	443 $\pm$171	2.6$\pm$1.6

- Debloating significantly reduces CPU/GPU memory usage and execution time for small model inference.
- Debloating shows smaller gains for larger models and training workloads.
- On average across all workloads, CPU memory, GPU memory and execution time are reduced by 2501 MB, 443 MB and 2.6s.

Outline

- Overview of Research Questions
- RQ1: Measure Bloat in Machine Learning Containers (Paper A, SIGMETRICS’24)
- RQ2: Debloat Machine Learning Containers (Paper B, SoCC'25)
- RQ3: Identify Bloat in Host Code (Paper C, NDSS’26)
- RQ4: Debloat Host Code (Paper D, Under Review)
- RQ5: Debloat Device Code (Paper E, MLSys’25, Best Paper Award)
- Conclusion

Contact Us

Huaifeng
huaifeng@resecureit.com

Ahmed
ahmed.hassan@chalmers.se

https://resecureit.com/

https://github.com/reSecureIt



CHALMERS

UNIVERSITY OF TECHNOLOGY

Cost of Hardened Images today!

Vendor	What it sells	Public pricing	Limit
Chainguard	Curated hardened base images from source	Catalog from $19K for a team of 10; per-image licensing on request	Base layer only; catalog lock-in
Docker Hardened Images	Curated hardened base images	Community tier free; Select from $5,000 per repo / year	Base layer only
VulnFree	Hardened / custom images	$800 per image per month (co-founder, public forum)	Base layer only
Bitnami Secure Images	Hardened application images	$62,000 flat annual enterprise licence	Catalog only
RapidFort	Runtime-profiled hardening of your images	Free tier (5 images); paid tiers quote-based	File level; stops at library boundary
Snyk · Sysdig · Aqua · Prisma · Anchore	Scanning and detection	Per-developer / per-node subscriptions	Find, do not remove