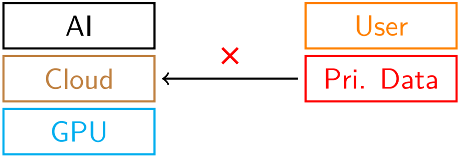
Honeycomb: Secure and Efficient GPU Executions via Static Validation

Haohui Mai Jiacheng Zhao **Hongren Zheng**[†] Yiyang Zhao Zibin Liu
Mingyu Gao Cong Wang Huiming Cui Xiaobing Feng Christos Kozyrakis

PrivacyCore ICT,CAS Tsinghua[†] Stanford IDEA BUPT

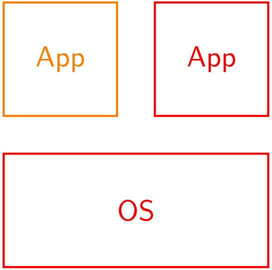
AI on private data needs S&P solutions

- AI is powerful
  - e.g. ChatGPT
- Still security concerns
  - Private data: e.g. medical/financial records
  - User does not trust 3rd party cloud



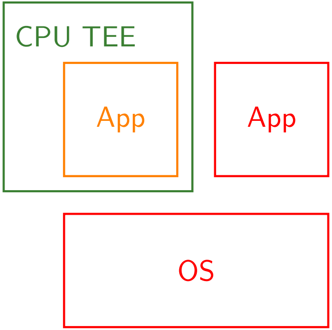
GPU TEE: a pragmatic approach

- In Cloud, User App can be harmed by other Apps and the OS



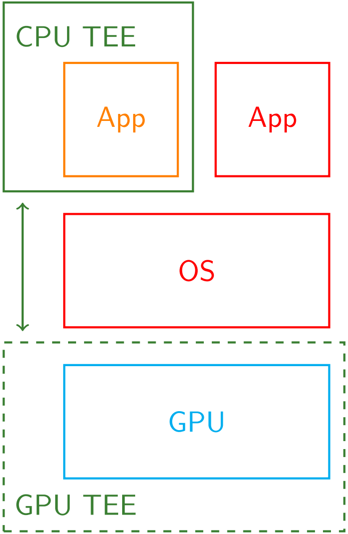
GPU TEE: a pragmatic approach

- In Cloud, User App can be harmed by other Apps and the OS
- Trusted Execution Environment (TEE) provides *isolation*
  - Special CPU hardware: Intel SGX/TDX, AMD SEV
  - Efficient: native speeds within enclave



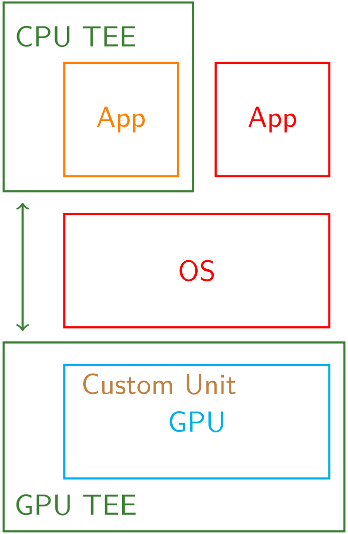
GPU TEE: a pragmatic approach

- GPU is powerful
  - Widely used by AI
- We want GPU TEE
  - Won't leak private data



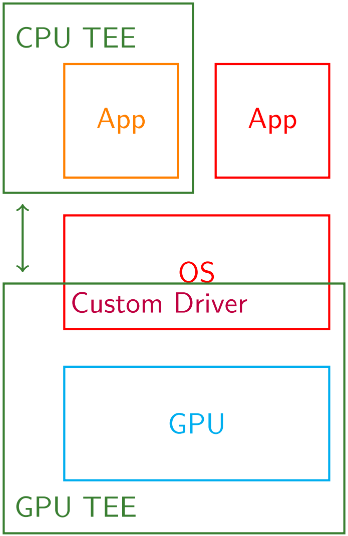
GPU TEE: a pragmatic approach

- GPU is powerful
  - Widely used by AI
- We want GPU TEE
  - Won't leak private data
- Current proposals
  - Hardware modification: slow evolution



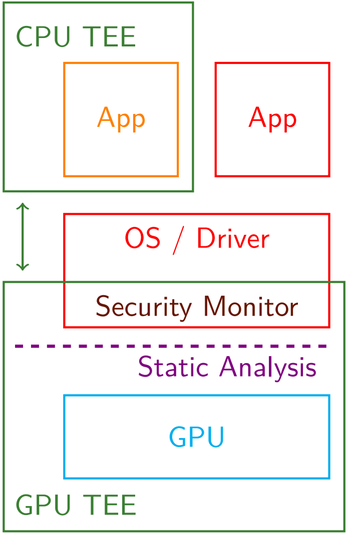
GPU TEE: a pragmatic approach

- GPU is powerful
  - Widely used by AI
- We want GPU TEE
  - Won't leak private data
- Current proposals
  - Hardware modification: slow evolution
  - Driver-based: large TCB (>1M SLOC), error-prone



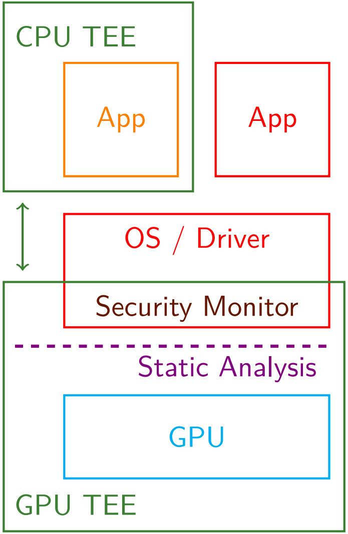
Honeycomb: confining behaviors via *static validation*

- Previous work: keep invariant
  - Either in hardware or driver
- Our work: by static validation
  - On the proper interface - - - - -



Honeycomb: confining behaviors via *static validation*

- Flexible: Complement hardware limitations
- Efficient:
  - Security checks at load time: 2% overheads for BERT / NanoGPT.
  - Modest overall dev. efforts.
- Secure: 18x smaller TCB compared to Linux-based systems

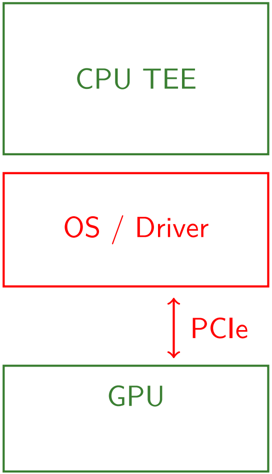


Agenda

Threat Model

Adversary

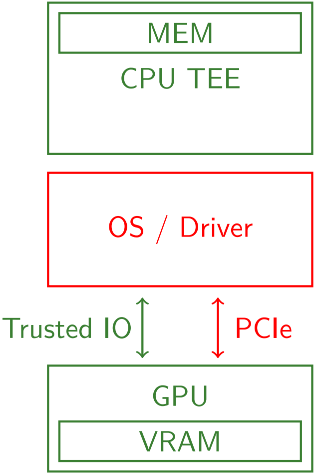
- Controls entire software stacks (OS / compiler / hypervisor)
- Has physical access of the hardware
- Sniffs PCIe traffic
- But cannot tamper the CPU or GPU silicons



Threat Model

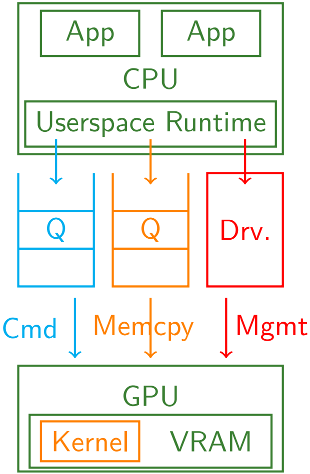
Assumptions

- CPU TEE (e.g., AMD SEV-SNP)
- Discrete GPU with integrated memory
- Trusted I/O paths: detailed in the paper
- Side-channel attacks are out of scope



GPU is a discrete accelerator

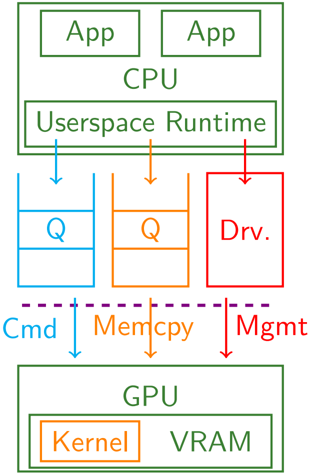
- Userspace queues
  - MemOp queue: Memcpy kernel / data
  - Cmd queue: Launch kernel
- Kernel space driver
  - Initialize hardware and address space
  - Alloc/Multiplex device memory / queues



Agenda

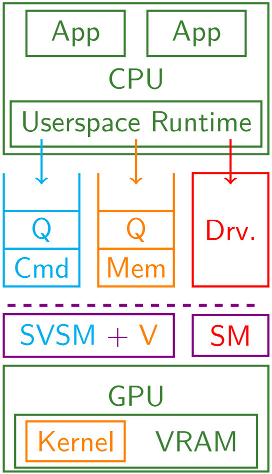
Overview of Honeycomb

- Keep invariant by static validation
  - On the proper interface - - - - -
  - Regulating high-level semantics



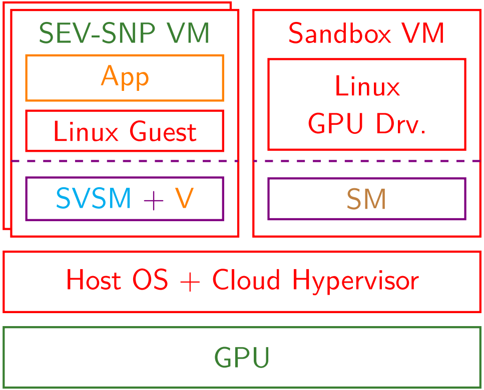
Overview of Honeycomb

- Keep invariant by static validation
  - On the proper interface - - - - -
  - Regulating high-level semantics
- SVSM: on Queue
  - Secure VM Service Module
- Validator: ensure *safe* GPU kernel
- Securiy Monitor: on Driver



Architecture of Honeycomb

- Honeycomb and hardware are TCB
- SVSM for SEV-SNP VM
  - SEV-SNP is an AMD CPU TEE feature
  - Regulating user app behavior
- Validator
  - Ensure *safe* GPU kernels
- SM for Sandbox VM
  - Regulating GPU driver
- System-wide invariant: Efficient IPC
  - Between GPU kernels of Apps



SVSM / SM: intercepting at *lowest* level

- Validate queue cmd / MMIO
- To ensure
  - Validated kernels
  - Init sequences
  - Memory isolation
  - Secure memcpy
- Remove OS kernel / GPU driver / runtime from the TCB

```
void check_launch_kernel(
  AddrSpace *addr,
  DispatchPkt *p) {
  if(!validated(addr,
      p->kernel_object))
    abort_user();
...
```

SVSM / SM: intercepting at *lowest* level

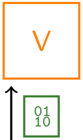
- Validate queue cmd / MMIO
- To ensure
  - Validated kernels
  - Init sequences
  - Memory isolation
  - Secure memcpy
- Remove OS kernel / GPU driver / runtime from the TCB
- Challenge: Recover sematics from MMIO
- BTW: Found 5 new bugs in AMDGPU, deployed in Linux 5.19

```
void check_launch_kernel(
  AddrSpace *addr,
  DispatchPkt *p) {
  if(!validated(addr,
      p->kernel_object))
    abort_user();
...
```

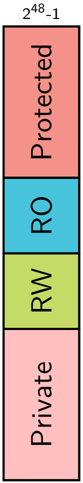
Validator: analysing *binaries*

- Invariant: partitioned addr. space
- Integrity: analyse memory access range
  - e.g. No write to protected region
- Validate GPU kernel at load time
  - Modest overhead
- On binaries: remove compiler from TCB

$$\text{blockDim.x} == 256$$
$$0 \leq \text{blockIdx.x} < 128$$
$$0 \leq \text{threadIdx.x} < 256$$

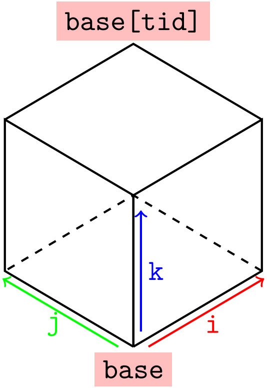


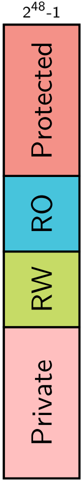

```
void fill( int *base ,
    int n, int b ) {
  u32 dim = blockDim.x ;
  u32 gid = blockIdx.x;
  u32 lid = threadIdx.x;
  u32 tid = gid * dim + lid;
  if (tid < n)
    base[tid] = b;
```



Polyhedral Analysis + GEMM: ☺

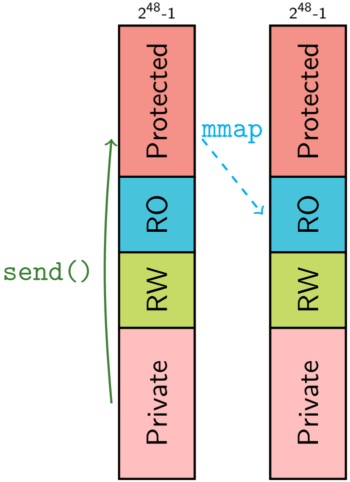
- Range checks using polyhedral analysis
  - Techniques from auto parallelization
  - e.g. Given `base`, conclude `base[tid]`
- **Minimal** overheads for ML workloads
  - Mostly regular
- Complex programs: add runtime checks
  - e.g. indirect heap references `a[b[i]]`
- Impl challenge: complexity of analysing directly on binaries





Efficient IPC: secure *direct memcpy*

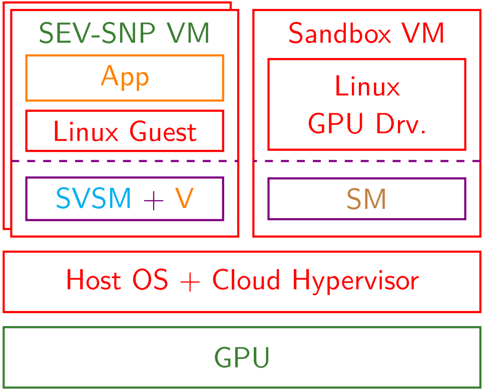
- Useful primitive for multi-stage pipelines
  - Components from multiple vendors
- Validation enforces proper IPC region
  - Trusted primitive `send()`
  - Sender's Protected
  - `mmap` to receivers's RO
  - Avoid double encryption/decryption across the boundary of enclaves



Agenda

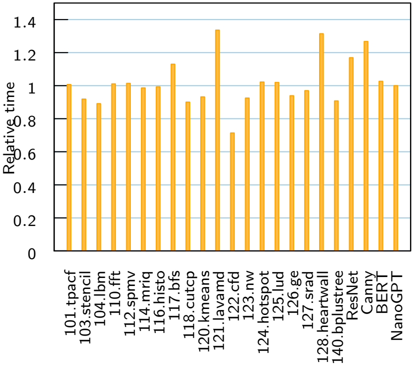
Security Monitors minimize TCB

- 2 EPYC 7433 CPUs, 1 AMD RX6900XT GPU
- Linux 5.17, ROCm 5.4.0
- TCB of Honeycomb (~82 KLOC): 18x smaller
  - Linux kernel ~1.5 MLOC
    - Core functionalities
    - Drivers (AMDGPU) and libraries (DRM & TTM)
  - Userspace runtime (ROCm) ~400 KLOC



Static validation is efficient

- 5 benchmark suites, HPC, CV, ML (DNN/Transformer). 23 apps in total
- Relative perf from 0.71-1.31 compared to Linux stack
  - breakdowns in paper
- Efficient on ML workloads
  - 2% overheads for BERT / NanoGPT
- Spent most time on GEMM kernels
  - polyhedral analysis works well
- Modest dev. effort to pass validations



Conclusion

- Honeycomb supports secure and efficient GPU executions
- Static analysis (Validation) is a practical and flexible technique for GPU apps
  - Honeycomb enhances security via co-designing validation + OS support
  - Efficient on real-world workloads
- The end-to-end SW/HW stack for GPU evolves quickly
  - A promising technique to explore novel designs