

THE PROBLEM: THE LOOP IS DOMINATED BY AUTOTUNING

Helion kernels are written as tile loops in Python; the compiler lowers to Triton (along with other backends) and the **autotuner** picks block sizes, warps, pipeline stages, loop order and indexing per (shapes, dtype, device).

10–20

min per compile for a full search – unusable while iterating

14–31%

slower kernels when starting from the default config

0

reuse from the autotune cache after any source edit

WHY THE CACHE MISSED

The exact-match key hashes the kernel **source**. Rename a variable, move a line, touch a comment – and prior tuning is gone.

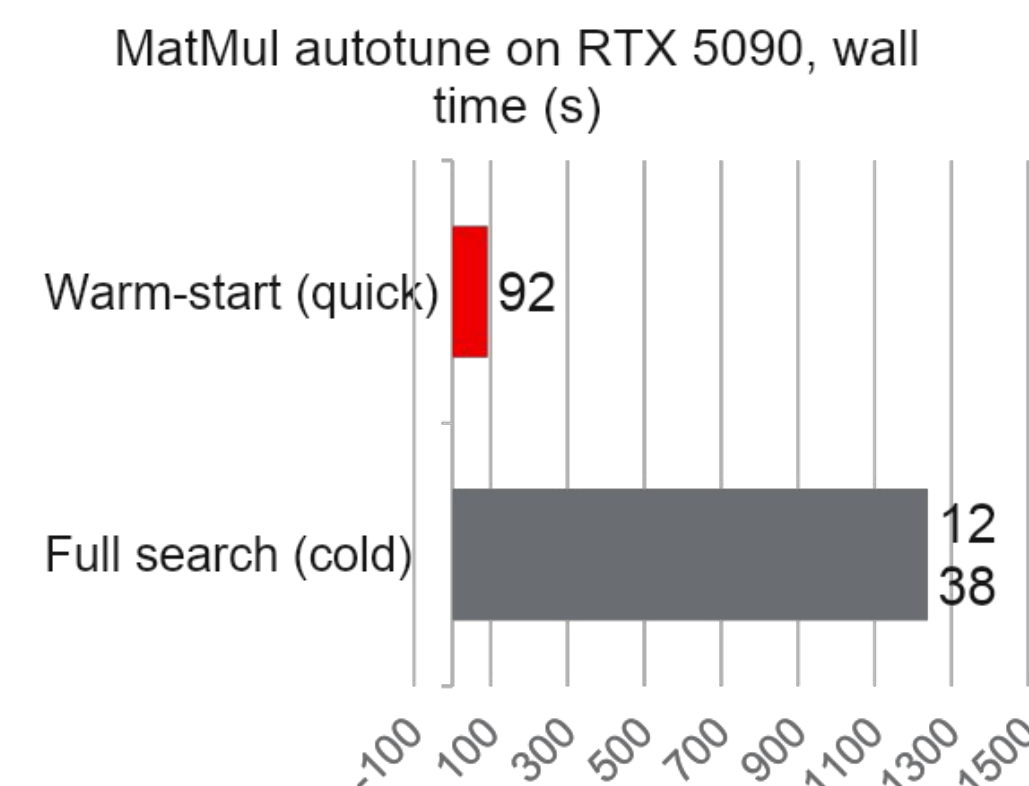
```
LooseAutotuneCacheKey(
    specialization_key = (dtypes, shapes, strides, ...),
    hardware            = "NVIDIA RTX 5090", backend="triton",
    kernel_source_hash = sha256(inspect.getsource(fn)),
    config_spec_hash   = "c069ee1c...", # ↑ any edit
) → sha256(repr(key)) names the .best_config file
```

Exactness is the right contract for **skipping** autotuning: a different kernel must never silently reuse a config. But the winning configuration depends on **shapes, dtypes and device** – not on how the source is spelled.

WHICH EDITS REUSE PRIOR TUNING?

Edit to the matmul kernel	Args	Search space	Result
Fused ReLU / GELU epilogue	same	same	warm start
Row-max normalisation in epilogue	same	same	warm start
Extra load from an existing arg	same	differs	cold start

FULL QUALITY AT QUICK COST



13×

faster autotuning, **1238 s** → **92 s**, with the same winning-configuration quality.

FROM_BEST_AVAILABLE: SEED THE SEARCH

Before

- Population = 30 random configs
- Evolve for N generations
- Every run starts from nothing, even on the same shapes and GPU

FROM_BEST_AVAILABLE

- Population = user seeds, compiler seeds, default config, + **cached winners**
- The autotuner still searches
Default config always pinned – a bad seed can never make it worse

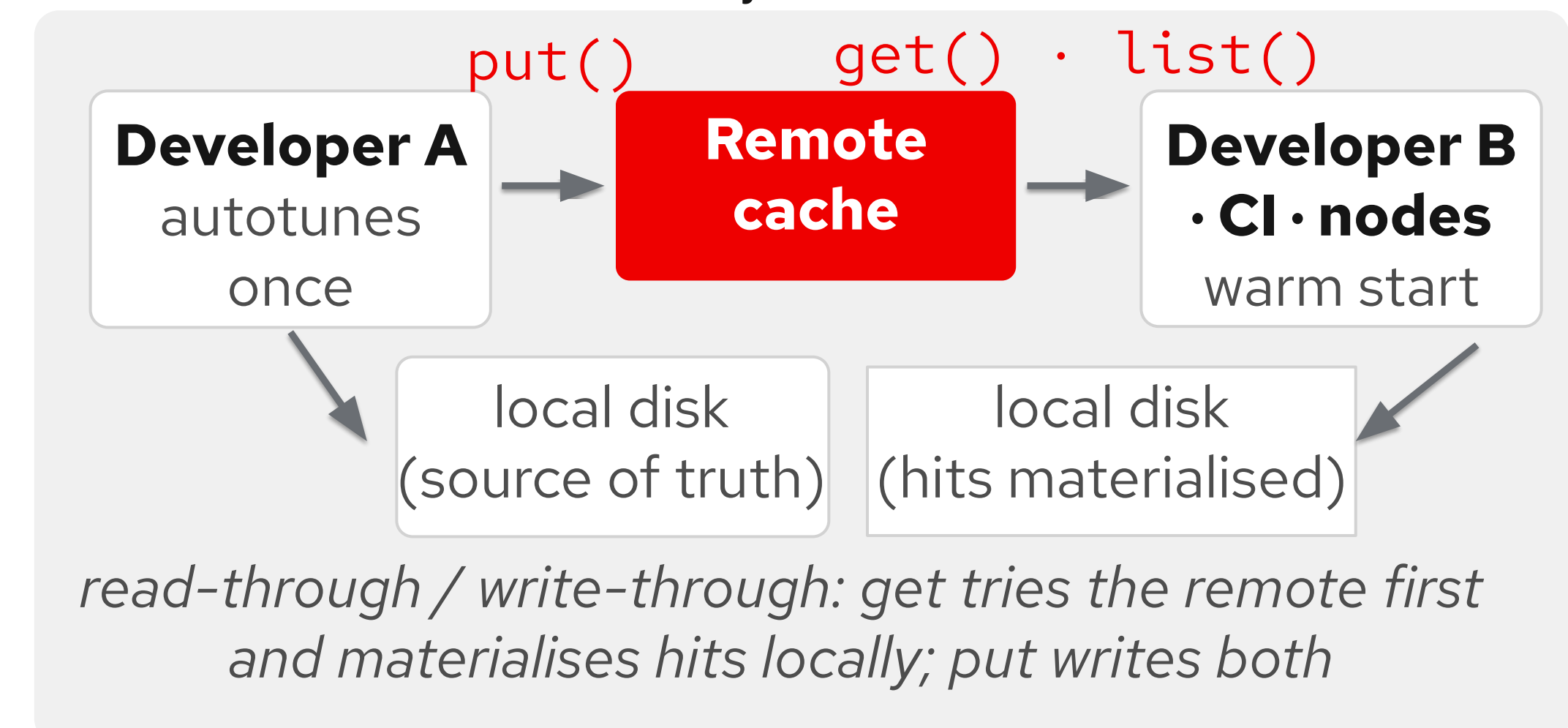
THE LOOSE-MATCH KEY

A cached entry is reusable only if it passes all three. The source hash is **deliberately not** one of them.

- Same hardware**
A B200 winner is not an RTX 5090 winner.
- Same input characteristics**
Specialization key: dtypes, shapes, strides, static indices. Code objects are normalised.
- Compatible search space**
config_spec_hash – a structural fingerprint of the knobs.

THE REMOTE CACHE

Helion ships **RemoteCacheBackend**, an ABC – not a store. Implement **get** / **put** and optionally **list()** over Redis, S3, HTTP or a shared filesystem. Local disk stays the source of truth on every machine.



Warm-start via list(). Fewer than 20 compatible local entries → merge remote entries, same three filters, deduplicated against local.

Degrades gracefully. Remote read or list failure → warning, local only. Put failure → warning; the local write already succeeded.

Two independent switches. The backend env var enables warm-start; add `HELION_AUTOTUNE_CACHE=RemoteAutotuneCache` to also publish winners.

TRY IT



Scan for examples

A Redis backend in ~25 lines, warm-start walkthroughs and benchmark scripts:
github.com/fulvius31/helion-cache-examples

Helion: github.com/pytorch/helion · PRs #1365, #2317, #2453