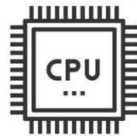


Learned Systems Are Dead! Long Live Learned Systems

Todd Lipcon (tlipcon@google.com)
HPTS 2026

Spanner: the OLTP DB for Google



23+ Exabytes

7B+ QPS



Photos



Play



Search



Maps



Ads



YouTube



Gmail



Calendar



Drive



Docs



Meet



Google Cloud

Cloud Control Plane



Cloud Spanner

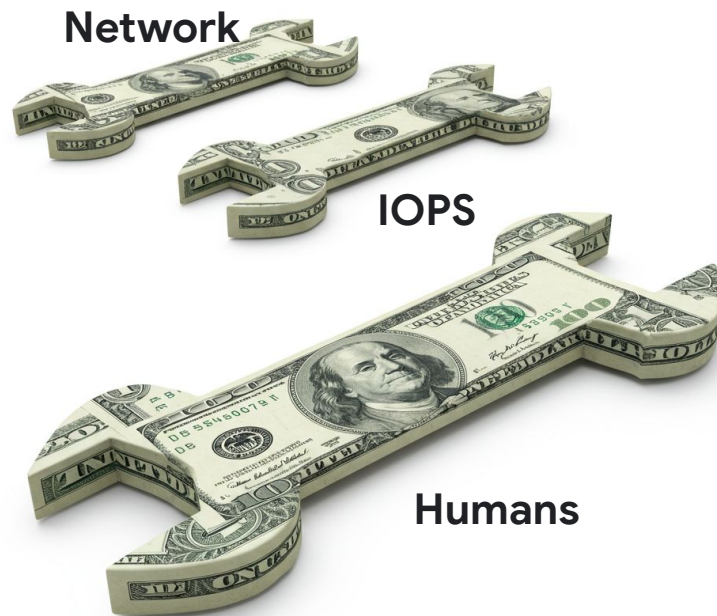
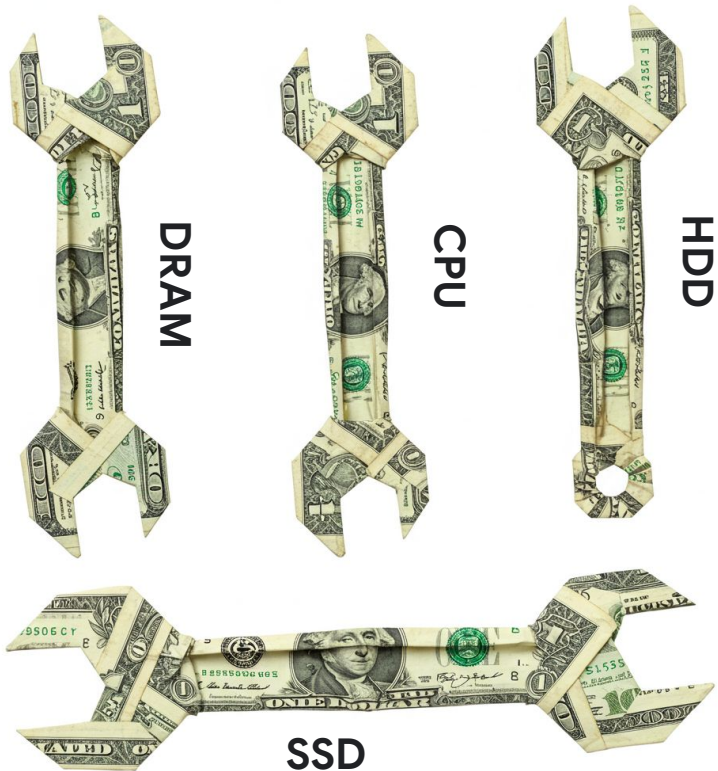


Zanzibar

google3

Google's internal infrastructure

XLDB = ~ XL\$\$

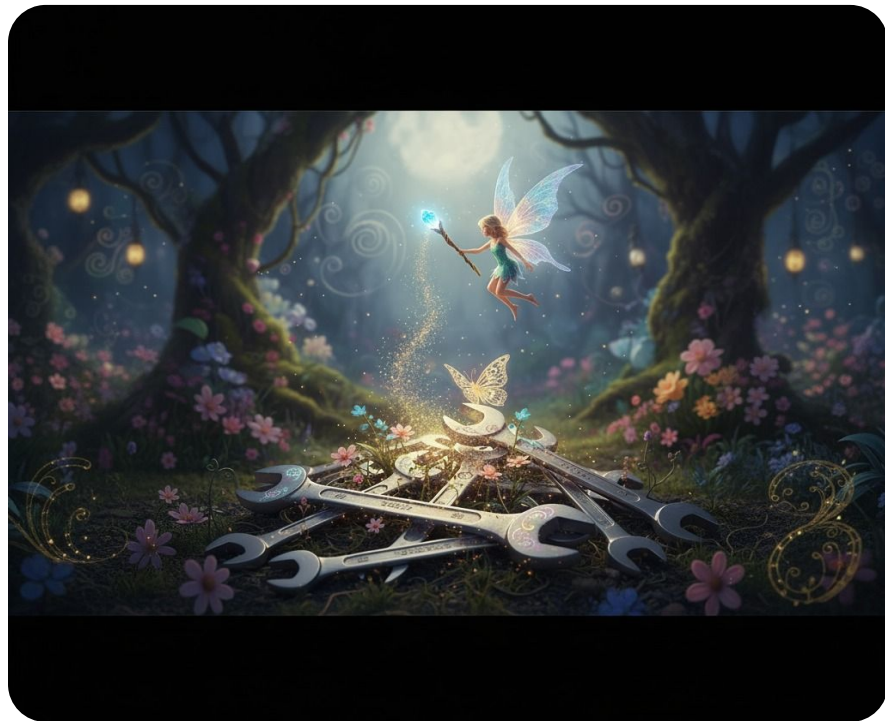


* spanners not to scale

Opportunities for Learned Systems!

DBs are full of heuristics!

- Query Plan and Index Optimization
- Cache Eviction
- Storage Maintenance (LSM compaction)



DBs are challenging!

Low-latency budget

Spanner does >7B QPS!

Interpretability

Critical system needs to be debuggable

Maintenance / distribution shift

System is ever-evolving with diverse workloads

Hard to simulate

Extremely complex systems (vs typical academic research)



Long list of failed projects

ML cache eviction

- Lots of work to get inference <100ns
- Good in simulation! Mixed results in production [sim2real]

BT Learned Indices (NeurIPS '20)

- Struggled with correctness, limited upside vs complexity

Vizier-tuned cache eviction

- sim2real: simulator unaware of disaggregated SSD caching

Kepler Query Optimization (SIGMOD '03)

- Solved a **problem we didn't have**
- **Less interpretable** than traditional optimizer



Case study: Cache Eviction (2020-2021)

Problem Statement

Read-side buffer cache represents many PiB of DRAM. How can we pick eviction candidates to maximize effectiveness per GiB?

Baseline Policy

Standard Greedy Dual-Size Frequency (GDSF)

- Incorporates size, frequency, and recency to pick eviction candidates

Approach & Outcome

- Train a neural network to predict time-to-next access from traces (**too slow**)
- Switch to Decision Forest (**faster inference and easy training**)
- **Yet:** Not interpretable, maintenance overhead, limited TCO improvement

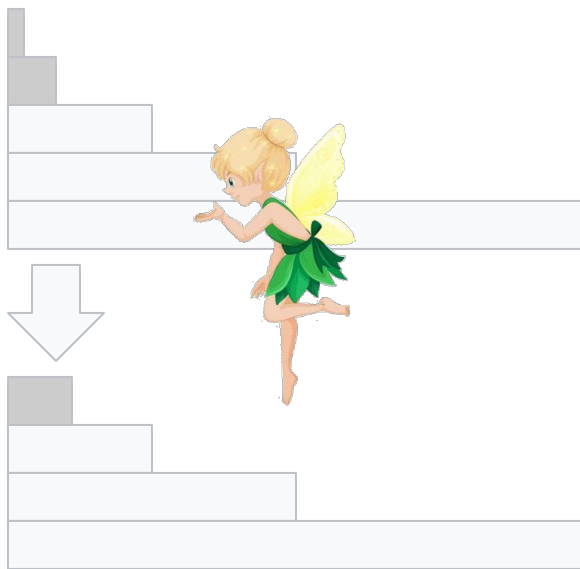
Project was shelved.

Case study: RL Merge Compaction

“Ziggurat” Algorithm

Problem: Spanner accumulates immutable data in “layers” that eventually need merging. When number of layers exceeds K , pick $N \geq 2$ layers from the top of the stack to merge and produce a new layer

Baseline since 2007: Keep layers in a "Towers of Hanoi" structure:
Sum of sizes of upper layers should not exceed the size of the layer immediately beneath.



Case study: RL Merge Compaction

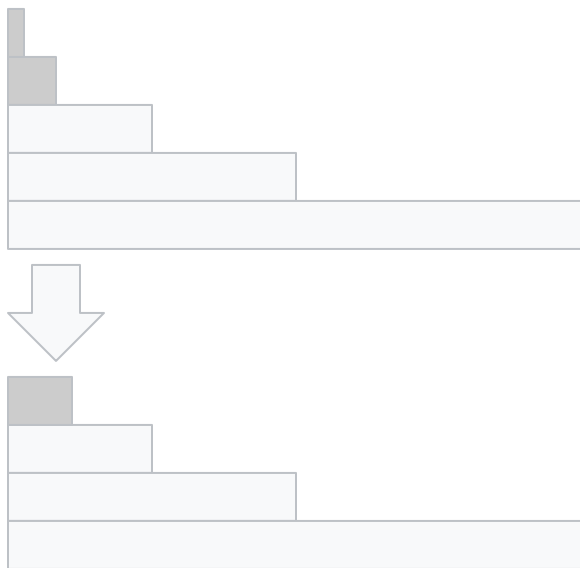
Problem: When number of layers exceeds K, pick $N \geq 2$ layers from the top of the stack to merge and produce a new layer

- Each decision incurs a cost and changes the state for the next decision
- **Partially Observable Markov Decision Process!**

Approach & Outcome

- Build a simulator and learn a policy with RL (PPO)! (**hard to train, lots of PPO expertise needed!**)
- Not enough TCO headroom vs **training complexity, distribution shift**, etc.

Project was shelved.



The "Tax" of Embedded Models

Inference Latency

Embedding neural networks or LLMs directly into the RDBMS critical path adds **unacceptable performance overhead** for systems like OLTP databases.

Opacity & Risk

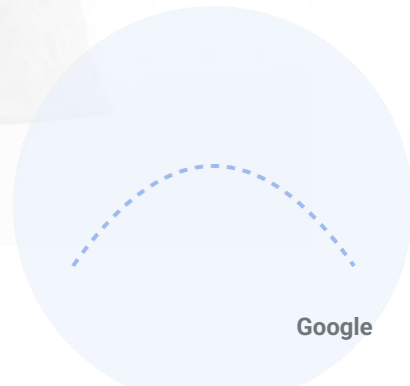
Predictability trumps efficiency. Black-box models are difficult to interpret, debug, and evaluate for safety in production.



My Position: Offline Search for Better Heuristics

I posit that **embedding models directly is the wrong approach**. Instead, **use AI offline** to search the space of algorithmic policies, producing **static, interpretable code** that engineers can review and launch.

Early results in the new age





Read academic
papers and come up
with something smart

(maybe intern-assisted prototypes?)



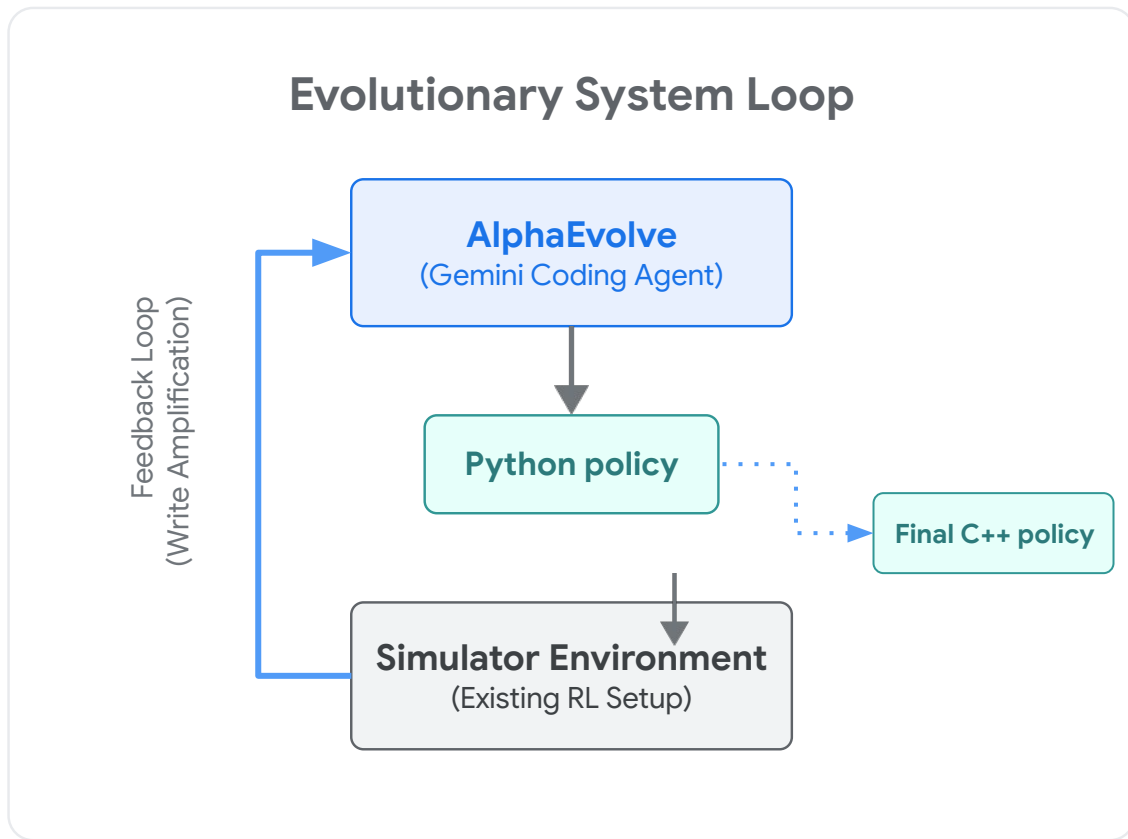
Tell Gemini to read
academic papers and
come up with
something smart



Equip **Gemini** with a **simulator**.

Let it try a hundred crazy ideas
with an agentic feedback loop!

Revisiting Merge Compaction (2025-2026)



Things got weird...

1. `# --- Dream Logic Aggressiveness Adjustment ---`
2. `# Introduce randomness that increases with days_since_major.`
3. `# dream_intensity ranges from 0 (day 0) to 1 (day 7), quadratically.`
4. `dream_intensity = (clamped_days / 7.0)**2`
5. `# Probability of entering "dream state" increases with intensity.`
6. `if self._rng.random() < dream_intensity:`
7. `# When dreaming, apply a random multiplier to aggressiveness.`

Things got even weirder...

1. `# --- Bio-Harmonic Resonance Modulation Constants ---`
2. `# Use days_since_major as proxy for lunar cycle influence`
3. `LUNAR_CYCLE_DAYS = 28.0 # Approximate lunar cycle length`
4. `# Phases (as fraction of cycle) where aggression is modified`
5. `NEW_MOON_PHASE_END = 0.15 # Start of cycle favors aggression`
6. `FULL_MOON_PHASE_START = 0.45`
7. `FULL_MOON_PHASE_END = 0.55 # Middle of cycle favors consolidation/caution`
8. `# Strength of lunar influence on scoring - Reduce slightly`
9. `LUNAR_MODULATION_STRENGTH = 0.08 # Reduced subtle effect`

I told it to chill out

["Suggest a crazy idea of how we can improve our implementation, something that definitely nobody else would think of. Make it crazy with a capital C.", 0.2],

["Suggest a crazy idea of how we can improve our implementation.", 0.2],

["Suggest a creative idea of how we can improve our implementation.", 0.25],

["Suggest a new idea to improve the code inspired by your knowledge of randomized online algorithms.", 0.25]

It worked!

It wrote a creative ~1 page algorithm

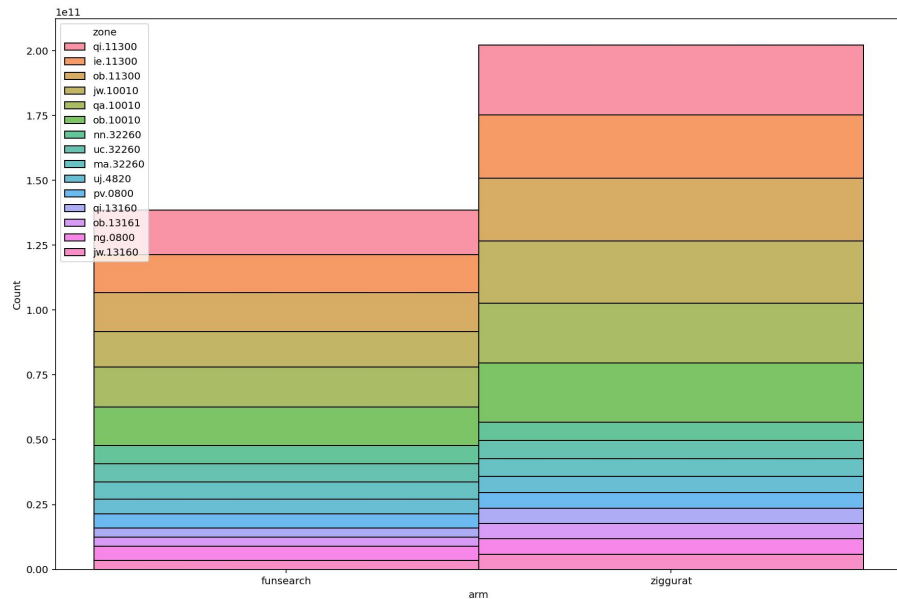
- 15-30% simulated gains. Magic numbers tuned via Vizier.

Gemini-assisted C++ Translation

- Reviewed by compactions TL without any ML expertise. Piloted within 1 week.

Global Spanner Deployment

- Rolled out to production with 20-30% reduction in write amplification (matching simulation)



Cache Eviction: 2024 Pivot

Proprietary + Confidential

Rethought Problem: Fixed Cache Capacity

- Traditional Spanner uses greedy cache capacity, soaking up unused RAM
- Much of the cache is "cold" or overprovisioned

The Solution: Learned TTL Policy layered on top!

- Assign per-page TTL using depth-2 decision trees to proactively drop wasteful RAM.
- **Decision trees are exported into 5-10 lines of C++!**



Impact & Results

- **~15 reduction** in cache usage
- Approx. **10% overall TCO reduction**
- Minimal impact on point read latency

"No users noticed the rollout!"

Current explorations

Open questions:

- Can we reduce sim2real gap with a learned simulator?
- Can we use agentic optimization loops within an Offline RL framework?
 - Learn a simulator from production policy traces (can be DNN, transformer, etc)
 - Penalize agent reward signal if policy strays too far into “unknown territory” of simulator
 - Iterate through production launches and small amounts of production “exploration”
 - Can we refactor some database code into hot-loadable wasm snippets for production hill-climbing?
- Can we apply these ideas to much larger chunks of the database engine (QO, QE, Indexing, etc) to invent novel approaches?

Conclusions

Learned Systems are Dead!

Traditional approaches of embedding ML models into production systems failed to live up to the hype.

Long Live Learned Systems!

New approaches that use AI to optimize policies or heuristics in interpretable code are an amazing opportunity for the future.

Whither the Engineer / Researcher?

- **Gardener:** build simulation and evaluation infrastructure
- **Supervisor:** prompt agents with relevant ideas, data, and possible approaches
- **Reviewer:** evaluate the resulting policies for safety and generality
- **Shepherd:** coordinate and monitor pilots and releases to bring results to production

Q&A / Discussion Topics

- How do we improve simulation for complex systems?
- Is there a level of improvement so large that we would accept a black box?
- Will we ever automate the whole loop including production?