Stateful Adaptation and Memory in Foundation Models

Yong Zhong | Research Interests | September 2026

Research focus

I want to understand how foundation models can adapt during use while retaining information needed later. The immediate question is how a model should alter its computation given its current state; the longer-term question is what it should carry forward when earlier context is no longer available. I want to study these questions under explicit memory and compute budgets. My current work examines state-dependent interventions; next I want to extend that logic to online memory decisions: which KV entries to retain, where to write new information, and which parametric memory to read.

Where the question came from

My first research project, RADAR at SUSTech (IROS 2026, co-first author), is a closed-loop data generation engine for robotic manipulation. A vision-language model proposes tasks, a policy executes them, and the system judges outcomes and recovers from failures. I integrated the policy into the loop, built RLBench scenes, and ran most of the simulation evaluation. The pipeline detected and recovered from failures, but did not use their causes to decide what data to collect next. While working on GeoMIND, I also encountered manipulation failures that appeared to involve losing spatial information across observations, which led me to ask what task-relevant state an embodied model needs to preserve.

Adapting to the current state

Since February 2026 I have been a visiting research student at NUS, advised by Kai Wang and Prof. Yang You, working on inference-time adaptation and parameter generation. In my first-author steering project, fixed representation edits often lagged prompting on the benchmarks we used. I first tested KV-cache edits to reuse previous computation; they underperformed, so the project moved to hidden-state interventions generated per token and per layer from the live residual stream and a target behavior. The next question is whether feedback from an intervention can guide the next one. I would measure locality and stability alongside behavioral control, rather than assuming that stronger interventions are always better.

In a separate masked diffusion language model project, I contribute to timestep-conditioned parameter-efficient adaptation. My early experiments with a mixer and dynamic gating between adapter components did not survive into the final design. They sharpened a question that also appears in steering: which changes in inference state actually require different parameters, and which parts of an adaptation can be shared?

State-aware memory decisions at test time

Long-running inference makes state-aware adaptation a memory-management problem. As new context arrives, the model must decide which KV entries merit exact access and what information to carry beyond the working window. I want to study a test-time controller that decides which KV entries to evict and routes each new block to bounded parametric-memory modules for updating. When a query arrives, it would route the read to relevant modules. A recurrent parameter generator could update a selected LoRA memory module from its previous state and the new block. Unlike appending an adapter for every block, the total memory budget would remain fixed. The open question is whether these decisions can preserve useful history, incorporate corrections, and limit interference without rereading evicted blocks.

What I would test

I would begin with questions whose supporting evidence has left the working window, then add irrelevant blocks, corrected facts, and conflicting information to test retention and updating separately. I would compare learned KV eviction with fixed eviction policies, and routed writing and reading with uniform updates and unconditioned reads. Under matched storage, training, and inference budgets, I would also compare parametric memory with chunk-adapter composition, recurrent states, text summaries, and retrieval. Alongside answer quality, I would report accuracy by evidence distance and update count, update latency, peak memory, and end-to-end inference cost. I would test where bounded memory fails, including possible inconsistency between new adapters and KV cached under earlier ones.

Longer-term direction

Steering, KV eviction, and parametric-memory routing act at different points in a system, but each asks which intervention is warranted by the current state and its future consequences. I would first establish reliable evaluation for bounded-memory updates, then test whether state-aware control improves what the model keeps in KV, writes into memory, and retrieves when needed.