Deep-Latent Mixture of Models: A Training-Free Architecture for System 2 Reasoning via Latent-Space Collaboration

Lawrence Beckwith, OtherU

Abstract

We present **Deep-Latent Mixture of Models (DL-MoM)**, a training-free architecture that enables multiple large language models to collaborate **primarily in latent space**, approximating System 2–style reasoning without fine-tuning. Unlike traditional mixture-of-experts systems that route discrete text tokens, DL-MoM routes **probabilistic belief representations**—**Soft Belief Packets**—between expert models and performs consensus directly over latent preferences. DL-MoM integrates (i) **probabilistic latent reasoning** (Soft Thinking, Coconut), (ii) **adaptive computation** (SwiReasoning) via trend-based entropy switching, (iii) **multi-agent latent collaboration** (LatentMAS, InterLat) while replacing learned alignment/bridge components with training-free alternatives, and (iv) **training-free KV-cache compression** (KIVI, MiniCache) for efficient *per-expert* context retention. We introduce three key innovations: (1) a **Sparse Belief Packet** protocol enabling cross-model latent communication without learned alignment matrices, (2) a **trend-based entropy controller** with switch-count controls for stable dynamic switching between exploration and exploitation, and (3) a **contrastive TIES-style consensus engine** that merges expert preferences in latent space. We provide an implementation blueprint compatible with off-the-shelf models (e.g., Qwen-2.5, Llama-3) and a reproducible experimental protocol with ablation registries and runner tooling. We further discuss how *optional* learned cache-to-cache (C2C) projectors can enable KV transfer across heterogeneous architectures, but this extension is not training-free.

1. Introduction

Large language models (LLMs) have demonstrated remarkable performance across diverse tasks, yet their reasoning process remains fundamentally constrained by token-by-token autoregressive generation. This "System 1" mode—fast, intuitive, and locally greedy—contrasts with deliberate, exploratory "System 2" reasoning observed in human cognition. Recent work has attempted to bridge this gap through externalized intermediate reasoning (Chain-of-Thought), branching search (Tree-of-Thought), and **latent reasoning** approaches that operate directly in continuous representation spaces.

In parallel, multi-agent LLM systems have emerged as a promising paradigm for complex problem solving, where multiple models collaborate by exchanging messages. However, typical multi-agent frameworks communicate in natural language, imposing repeated encoding/decoding overhead and discarding rich distributional information present in logits, hidden states, and KV caches. This interface forces subsequent agents to reconstruct semantic states from lossy token sequences, limiting bandwidth and efficiency.

We propose **Deep-Latent Mixture of Models (DL-MoM)**, a unified architecture that addresses these limitations by treating latent space as the primary medium for (i) reasoning, (ii) routing, and (iii) consensus. DL-MoM transmits **probabilistic beliefs** instead of argmax token decisions, enabling downstream experts to condition on uncertainty rather than only on committed strings. The system only decodes to text for final outputs (and optionally for fallback bridging when tokenizers are incompatible).

Core Contributions

1. **Soft Belief Packet Protocol.** A sparse representation—top-k token IDs with associated probabilities—that enables latent communication between heterogeneous models **without learned alignment matrices** and without requiring shared embedding vectors.
2. **Trend-Based Entropy Controller.** An adaptive mechanism inspired by SwiReasoning that monitors **block-wise normalized entropy trends** to dynamically switch between latent exploration and explicit exploitation, with switch-count controls to prevent oscillatory overthinking.
3. **Contrastive Consensus Engine (TIES-style).** A principled approach to combining expert preferences in latent space by centering logits into contrastive "preference vectors" and applying trimming, sign election, and disjoint merging.
4. **Training-Free Implementation Blueprint.** A concrete implementation plan and reproducible experiment registry compatible with off-the-shelf models and modern inference stacks.

---

2. Related Work

DL-MoM synthesizes insights from latent reasoning in LLMs, multi-agent collaboration systems, KV-cache compression, and model merging. We organize prior work into a coherent taxonomy and identify the specific contributions each thread provides.

2.1 Latent Reasoning in Large Language Models

We group latent reasoning methods into three categories: **raw hidden-state methods**, **tuned latent reasoning**, and **probabilistic/mixture-based (training-free) methods**.

Raw Hidden-State Methods and Tuned Latent Reasoning **Coconut** [1] introduced "continuous thought" by feeding the last hidden state back as the next input embedding, allowing models to compute intermediate reasoning without emitting tokens. Coconut emphasizes breadth-first exploration in latent space but requires task-specific fine-tuning to produce meaningful continuous trajectories.

SIM-CoT [7] extends the paradigm via step-level supervision, training an auxiliary decoder that is removed at inference time. While strong, the approach depends on specialized training and checkpoints.

Training-Free Probabilistic Latent Reasoning **Soft Thinking** [2] constructs "continuous concept tokens" as probability-weighted mixtures of token embeddings. Given logits z and embedding matrix E, the soft representation is:

$$e_{\text{soft}} = \sum_i \text{softmax}(z)_i \cdot E[\text{token}_i]$$

This preserves uncertainty over multiple candidate continuations and can reduce explicit token generation.

A subsequent analysis [5] identifies a limitation: deterministic Soft Thinking can enter a **greedy feedback loop**, increasingly collapsing onto the top-1 token and suppressing exploration. The proposed remedy—injecting stochasticity (e.g., **Gumbel-Softmax**)—is central to DL-MoM's robust latent exploration.

2.2 Adaptive Computation Controllers

SwiReasoning [3] studies when to reason in latent space versus explicit token space. A key insight is that absolute entropy alone is insufficient; **entropy trends across blocks** provide a more reliable convergence signal:

- Rising/high entropy → exploration/uncertainty → continue latent mode
- Falling/low entropy → convergence → switch to explicit mode

SwiReasoning also includes **switch-count controls** to avoid pathological oscillation and runaway compute.

2.3 Multi-Agent Latent Collaboration

LatentMAS [6] formalizes multi-agent collaboration in latent space and introduces an alignment operator W_a (estimated via ridge regression) to align agent representations for stable message passing. While effective, pairwise alignment becomes cumbersome for large ensembles and heterogeneous models.

InterLat [4] explores inter-agent latent communication with trainable adapters and compressed latent message protocols, but the core mechanism is not training-free due to learned bridges (e.g., LayerNorm + Linear). DL-MoM draws inspiration from InterLat's probability-preserving debates but substitutes training-free communication via sparse belief packets.

2.4 Training-Free KV-Cache Compression

Efficient latent collaboration benefits from compressing the KV cache transmitted between agents.

- **KIVI** [9] proposes tuning-free asymmetric 2-bit quantization of KV cache.
- **MiniCache** [8] compresses KV cache in the depth dimension via cross-layer redundancy merging.
- **ChunkKV** [10] introduces chunk-based compression that preserves semantic coherence for long context inference.

DL-MoM treats KV compression as an orthogonal systems layer: it affects feasibility and throughput while the belief packet governs semantic communication.

2.5 Model Merging and Consensus

TIES-Merging [11] merges parameter deltas from fine-tuned checkpoints via: - **Trim** small-magnitude noise, - **Elect** sign directions via voting, - **Merge** only agreeing directions.

DL-MoM adapts the *principle* of directional conflict resolution to runtime consensus over token preferences, with modifications appropriate for logits (raw scores, not deltas).

---

3. DL-MoM Architecture

Core philosophy: Standard MoE/MoM systems route *tokens* to experts. DL-MoM routes *beliefs* (latent preferences). Experts think, communicate, and merge preferences in latent space, decoding to text only for final outputs (or optional fallback bridging).

3.1 System Overview

DL-MoM consists of five interconnected components:

1. **Lookahead Latent Router** (training-free expert selection)
2. **Soft Thinking Engines** (belief generation)
3. **Trend-Based Entropy Controller** (adaptive switching)

4. **Latent Communication Bus** (belief exchange; optional KV transfer; compression)
5. **Contrastive Consensus Engine** (TIES-style merging of preferences)

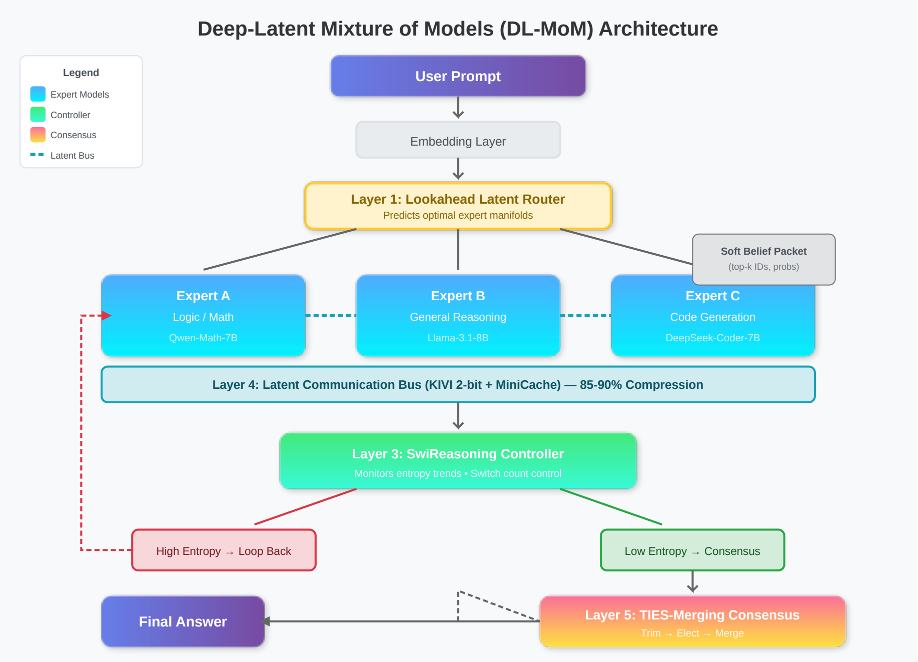
Deep-Latent Mixture of Models (DL-MoM) Architecture
Legend
Expert Models
Controller
Consensus
Latent Bus
User Prompt
Embedding Layer
Layer 1: Lookahead Latent Router
Predicts optimal expert manifolds
Soft Belief Packet
(top-k IDs, probs)
Expert A
Logic / Math
Qwen-Math-7B
Expert B
General Reasoning
Llama-3.1-8B
Expert C
Code Generation
DeepSeek-Coder-7B
Layer 4: Latent Communication Bus (KIVI 2-bit + MiniCache) — 85-90% Compression
Layer 3: SwiReasoning Controller
Monitors entropy trends • Switch count control
High Entropy → Loop Back
Low Entropy → Consensus
Final Answer
Layer 5: TIES-Merging Consensus
Trim → Elect → Merge

Figure 1: DL-MoM architecture overview

3.2 Layer 1: Lookahead Router via Perplexity Probing

Standard routers rely on keyword heuristics or trained classifiers. DL-MoM uses **perplexity probing** to select experts without training:

1. Generate a short "preview" continuation (e.g., 5 tokens) using a fast pass.
2. Decode the preview to text.
3. Compute perplexity of that preview text under each expert model.
4. Route to the expert with lowest perplexity (lowest surprise).

This selects an expert whose distribution aligns with the evolving reasoning trajectory (e.g., a math expert tends to assign lower perplexity to symbolic/mathematical continuations).

3.3 Layer 2: Soft Thinking Engines and Soft Belief Packets

Each expert generates **beliefs** rather than committing to discrete tokens at every step.

Soft Belief Packet Protocol (Sparse Belief Packets) A major barrier to heterogeneous latent communication is representational mismatch. Even if models share a tokenizer, their embedding matrices differ. Transmitting raw embeddings from model A to model B is not meaningful in general.

DL-MoM transmits a **Soft Belief Packet**:

$$\text{Packet} = \{(\text{token\_id}_1, p_1), \dots, (\text{token\_id}_k, p_k)\}$$

The receiver reconstructs a soft input embedding using its **own** embedding matrix E_{recv}:

$$e_{\text{soft}} = \sum_{i=1}^{k} p_i \cdot E_{\text{recv}}[\text{token\_id}_i]$$

Constraint (Tokenizer Compatibility). The packet protocol assumes a compatible vocabulary. For mismatched tokenizers (e.g., Qwen vs Llama), DL-MoM uses a **Text-Bridge Fallback**: decode a small representative string (e.g., top-1 token or short span) and re-encode on the receiver side. This preserves system structure but reduces “soft” information on that link.

Stochastic Soft Thinking (Avoiding Greedy Collapse) To prevent deterministic greedy feedback collapse [5], DL-MoM supports stochastic soft tokens using **Gumbel-Softmax**:

$$\tilde{p}_i = \text{softmax}\left(\frac{\log(p_i) + g_i}{\tau}\right)$$

where $g_i \sim \text{Gumbel}(0, 1)$ and τ controls randomness. The next soft input becomes:

$$\tilde{e}_t = \sum_{i \in \text{top-}k} \tilde{p}_i \, E(i)$$

3.4 Layer 3: Trend-Based Entropy Controller

DL-MoM decides when to remain latent (explore) versus when to emit explicit tokens (exploit).

Normalized Entropy Raw entropy varies with vocabulary size and calibration. We use normalized entropy:

$$H_{\text{norm}} = \frac{H}{\log|V|}$$

Trend Detection Maintain a sliding window of recent entropies and compute: - slope (linear regression over window), - variance (stability indicator).

Decision Logic (High-level)

- Decreasing trend + low variance → converging → switch to explicit output
- Flat/high or rising trend → uncertain/exploring → continue latent reasoning
- Switch count exceeds `cap` → force explicit mode to prevent oscillation

---

3.5 Layer 4: Latent Communication Bus (Beliefs + KV State)

Agents communicate through: 1. **belief packets** (semantic preferences), and optionally 2. **KV state reuse/transfer** (working memory / context), compressed for feasibility.

3.5.1 What Is Training-Free: Local KV Reuse + Compression KV caches are model-internal attention state. In DL-MoM's training-free core, each expert maintains and reuses *its own* KV cache across latent steps (i.e., `past_key_values` within that expert), and optionally compresses it (KIVI/MiniCache) to reduce memory bandwidth and enable longer internal rollouts.

This is distinct from *cross-model KV transfer*: no attempt is made to perform byte-wise cache copying between heterogeneous models.

3.5.2 Optional Extension: Learned Cache-to-Cache (C2C) Projection for Heterogeneous Transfer Direct KV transfer between heterogeneous models (different hidden sizes, head counts, RoPE conventions, or architectures) is not meaningful without bridging. A practical approach is a lightweight **cache-to-cache (C2C) projector**: a small learned adapter that maps the source model's per-layer key/value tensors into the target model's KV space, typically with gating to mitigate semantic interference [22]. This requires freezing base models and training only the projector on a small alignment dataset; it is therefore **not training-free**.

If avoiding pairwise projectors, a "universal latent" design can be used: each model learns an encoder to a canonical KV format and a decoder back to its own format (reducing pairwise $O(N^2)$ adapters). For intra-family models (same tokenizer and similar attention geometry), simpler linear alignment or layer-wise stitching is often sufficient, but still involves learned or estimated alignment parameters.

Training-Free KV Compression Stack

- **KIVI** [9]: asymmetric 2-bit KV quantization
- **MiniCache** [8]: depth-dimension merging across layers
- **ChunkKV** [10] (optional): chunk-based semantic grouping for long contexts

These methods reduce memory bandwidth requirements and enable multi-expert KV sharing on limited hardware.

3.6 Layer 5: Contrastive Consensus Engine (Runtime TIES-Style)

Naive averaging of logits is brittle because experts may disagree strongly and logits are raw scores, not parameter deltas. DL-MoM adapts a TIES-like procedure to **contrastive preference vectors**.

Centered "Preference Vectors" Convert raw logits into contrastive preferences by centering:

$$\mathbf{v}_i = \text{logits}_i - \mu(\text{logits}_i)$$

so $v > 0$ indicates above-average preference and $v < 0$ indicates below-average preference for that expert.

Consensus Procedure (Trim–Elect–Merge)

1. **Trim:** set small-magnitude elements to 0:

$$v_{i,j} \leftarrow 0 \quad \text{if } |v_{i,j}| < \theta$$

2. **Elect:** vote a direction per token:

$$s_j = \text{sgn}\left(\sum_i \text{sgn}(v_{i,j})\right)$$

3. **Merge:** average only experts agreeing with the elected sign:

$$v_j^{\text{final}} = \frac{\sum_i v_{i,j} \cdot \mathbb{I}[\text{sgn}(v_{i,j}) = s_j]}{\sum_i \mathbb{I}[\text{sgn}(v_{i,j}) = s_j] + \epsilon}$$

The merged preferences are converted into the next belief packet via softmax + top-k extraction.

---

4. Implementation

DL-MoM requires no fine-tuning—only custom inference-time logic.

4.1 Core Algorithm (High-Level)

Algorithm 1: DL-MoM Latent Collaboration Loop

```
Input: input_ids, experts[1..K], max_steps
Output: final text

packet = InitPacket(input_ids)              // start with hard tokens
kv_cache = [None] * K
entropy_window = []
switch_count = 0

for step = 1..max_steps:
  outputs = []
  for i = 1..K:                             // parallel expert thinking
    x_i = ReconstructSoftInput(packet, expert_i)
    y_i = expert_i.forward(inputs_embeds=x_i, past_key_values=kv_cache[i])
    H_i = NormalizedEntropy(y_i.logits[-1])
    outputs.append({logits=y_i.logits[-1], kv=y_i.kv, entropy=H_i})

  H_avg = mean_i(H_i); entropy_window.append(H_avg)

  if ControllerConverged(entropy_window, alpha, w, cap, switch_count):
    break

  merged = ContrastiveTIES([o.logits for o in outputs], trim=theta)
  packet = BeliefPacketFromLogits(merged, top_k=k)

  if stochastic_soft:
    packet = GumbelOrDirichletPerturb(packet, tau, lambda)

return DecodeFinal(packet)
```

4.2 Key Engineering Details

- **Detaching latent buffers:** store belief packets in fp16/bf16 and detach to avoid memory growth.

- **Positions:** when using `inputs_embeds` with `past_key_values`, ensure correct `position_ids` (cumulative length including cached tokens).
- **Tokenizer compatibility:** belief packets assume compatible vocabularies; otherwise use Text-Bridge Fallback.
- **KV transfer scope:** KV caches are only directly reusable within the *same* model instance/configuration. Heterogeneous KV transfer requires an explicit learned projector (C2C) or should be disabled in favor of belief/text bridging.
- **Overhead (soft embedding reconstruction):** reconstructing $e_{\text{soft}} = \sum_{i \in \text{top-}k} p_i \, E[t_i]$ has cost $O(kd)$ per expert per latent step (embedding gather + weighted sum). For a representative setting ($k = 50$, $d = 4096$), this corresponds to ≈ 0.4 MFLOPs and ≈ 0.4 MB of embedding reads in fp16. Relative to a transformer forward step, this overhead is typically small; in many implementations, `topk` selection and the expert forward dominate end-to-end latency. To minimize overhead, keep `ids/probs` on-device and implement gather+sum as a single GPU operation where possible.

4.3 Recommended Stack

Models (example ensemble): - Math: `Qwen/Qwen2.5-Math-7B` - General reasoning: `meta-llama/Llama-3.1-8B-Instruct` - Code: `deepseek-ai/deepseek-coder-7b-instruct-v1.5`

Infrastructure: - Transformers + Accelerate (baseline) - vLLM (optional) for high-throughput KV handling - Compression: KIVI/MiniCache implementations (custom or integrated) - Consensus: lightweight runtime tensor ops (no parameter merging required)

---

5. Theoretical Analysis

5.1 Information-Theoretic Foundation (Belief Packets)

We view belief packets through the information bottleneck lens [15, 16]. Let p be the full next-token distribution and $\hat{p}_k$ the top-k renormalization with tail mass $\tau = 1 - \sum_{i \in \text{top-}k} p_i$. The identity

$$\mathrm{KL}(\hat{p}_k \| p) = -\log(1 - \tau) \tag{1}$$

quantifies truncation distortion in terms of discarded mass. When τ is small (e.g., $\tau < 0.05 \Rightarrow \mathrm{KL} < 0.051$ nats), the renormalized top-k distribution remains close to the original in KL divergence [17]. This supports belief packets as a bandwidth-limited representation that preserves uncertainty structure compared to argmax messaging. When tokenizers are shared, reconstructed embeddings $\tilde{e} = \sum_i p_i E[t_i]$ lie in the convex hull of the receiver's token embeddings, remaining in-distribution by construction.

5.2 Consensus Stability and Convergence

Logit consensus can be cast as a single-iteration robust consensus propagation step [18, 19]. Let centered preference vectors be $v_i = z_i - \bar{z}_i$. The TIES-style merge (Trim–Elect–Merge) produces v^* with bounded deviation from the mean:

$$\|v^* - \bar{v}\|_2 \leq \sqrt{K}\,\rho\,\max_i \|v_i - \bar{v}\|_2,$$

where ρ is the fraction of entries with sign disagreement. Trimming suppresses low-magnitude noise; sign election avoids destructive cancellation. Calibration matters: per-expert temperature scaling (or z-scoring) before merging reduces bias from miscalibrated experts.

5.3 Superposition View of Latent Reasoning

Soft belief packets implement a superposition state $e_{\text{soft}} = \sum_i p_i E[t_i]$ that encodes multiple continuations simultaneously. This aligns with continuous-chain-of-thought theory showing superposition enables implicit parallel search vs. sequential argmax expansion [20]. Text MAS collapses to one branch per exchange; DL-MoM preserves a frontier distribution in one message.

5.4 Entropy-Guided Adaptive Computation

Normalized entropy $H_{\text{norm}} = H/\log|V|$ is a convergence proxy: under deterministic updates, H_{norm} decreases in expectation as evidence accumulates. A trend gate over a window w reduces false positives vs. a single threshold by $O(\sqrt{w})$ variance reduction, and a switch-count cap bounds oscillations (total switches $\leq$ cap), ensuring compute stays $O(S)$.

5.5 TIES Adaptation in Logit Space

Centering logits is distribution-preserving ($\text{softmax}(z) = \text{softmax}(z - \bar{z})$) and exposes directional preferences. Applying TIES on centered logits amplifies agreement while suppressing conflicts [11]; expected consensus norm grows when disagreeing signs are excluded, yielding sharper merged preferences. A spectral variant (low-rank projection before TIES) can further denoise high-rank logits [21]. Calibration alignment (temperature/variance) should precede merging in heterogeneous ensembles.

---

6. Proposed Experimental Evaluation

We outline a comprehensive experimental protocol to validate **DL-MoM**. Unless otherwise stated, all experiments report: **task accuracy**, **tokens generated** (explicit + latent), **wall-clock time**, **peak memory**, and **failure/drift**

rate. For stochastic variants, results are averaged over **3 seeds** and reported with **mean ± std**.

6.0 Experimental Controls

Models. Primary results use a single strong instruction-tuned LLM (e.g., 7–14B class). To support "model-agnostic" claims, replicate the full suite on a second model family when feasible.

Prompting. All variants share identical prompts and output formats. Latent-mode variants are required to emit a final explicit answer string for scoring.

Agent topology. Fix: number of agents (e.g., 2 or 3), roles (Generalist→Specialist), and maximum handoffs. Only ablation knobs change.

Budgets. Fix `max_steps`, `max_switches`, and `max_handoffs` across conditions (except where explicitly ablated).

6.1 Ablation Studies

Each ablation changes *one axis* relative to a **Unified Default**:

- Deterministic soft tokens
- Threshold gate (single-step entropy)
- Embedding-vector payload (same-model)
- No KV compression (fp16)

A1. Soft Token Variants Goal: test exploration vs stability trade-offs in continuous "soft concept" generation.

1. **Deterministic top-k mixture** *(baseline)*

$$\tilde{e}_t = \sum_{i \in \text{top-}k} p_i \, E(i)$$

2. **Gumbel-Softmax mixture**, $\tau \in \{0.5, 1.0, 2.0\}$ Evaluate both accuracy and variance across seeds.
3. **Dirichlet noise injection** on top-k probabilities $p' \sim \text{Dirichlet}(\lambda p)$ with λ chosen so expected $\text{KL}(p' \| p)$ matches a small target (e.g., 0.05–0.1). Report noise level explicitly.

Report additionally: top-k mass $\sum_{i \in \text{top-}k} p_i$ (as a bandwidth/coverage diagnostic).

A2. Controller Variants Goal: validate that the DL-MoM controller improves both efficiency and robustness.

1. **Single-step entropy threshold** *(naive baseline)* latent if $H_t > \alpha$, else explicit
2. **Trend-based switching**, window sizes $w \in \{3, 5, 10\}$ Gate on statistics over the last w steps: mean entropy, slope, and variance.
3. **Trend-based + switch-count limits**, cap $\in \{2, 5, 10\}$ Report "switches per sample" distribution (p50/p90) and timeout reductions.

Recommended logging: per-step entropy, mode, and handoff triggers.

A3. Communication Protocols Goal: isolate whether robustness comes from *representation* (soft beliefs) vs *bandwidth*.

1. **Raw embedding vectors** *(baseline; same-model only)* transmit fp16/bf16 embeddings for each latent step.
2. **Soft Belief Packets**, $k \in \{10, 50, 100\}$ transmit `(token_ids, probs)` and reconstruct embeddings on the receiver side.
3. **Full logit transmission** *(upper bound; systems stress test)* transmit logits (or top-k logits) and reconstruct probabilities exactly.

Bandwidth reporting: bytes communicated per latent step and per solved sample.

A4. Compression Configurations Goal: quantify memory/throughput gains vs drift.

1. **No compression** *(fp16 baseline)*
2. **KIVI 2-bit only**
3. **MiniCache only**
4. **KIVI + MiniCache combined** Evaluate both orderings if they differ (merge→quantize vs quantize→merge).

Drift requirement: report logits divergence vs uncompressed baseline (see §6.4).

6.2 Benchmark Evaluation

We evaluate on three capability groups. For each dataset, we fix the evaluation script, split, and answer-extraction rules, and we publish all prompts.

Mathematical Reasoning

- GSM8K
- MATH
- AIME (pin exact years/rounds used)

Code Generation

- HumanEval
- MBPP
- LiveCodeBench (pin snapshot/date)

General Reasoning

- ARC-Challenge
- HellaSwag
- MMLU

6.3 Baseline Comparisons

Single-model baselines - Direct prompting (no CoT) - Chain-of-Thought prompting [13] - Self-consistency (majority vote across N samples; report N and compute multiplier)

Structured reasoning baselines - Tree-of-Thought (fix branching factor + depth + evaluation heuristic) [14]

Multi-agent baselines - Text-based multi-agent debate / standard MAS (same agent roles + same budget)

Optional - Raw latent bus baseline (LatentMAS-style hidden-state passing) to directly quantify stability gains.

6.4 Metrics, Drift, and Failure Definitions

Token accounting - `explicit_tokens`: decoded tokens emitted - `latent_steps`: number of soft-token steps - `total_steps = explicit_tokens + latent_steps`

Wall-clock - per-sample latency (p50/p90) - throughput (samples/sec, steps/sec)

Drift metrics (mandatory) Compare each variant to a fixed reference pipeline (Unified Default, no compression): - **Logits KL drift:** mean KL between next-token distributions at matched step indices - **Cosine drift:** cosine distance between logits vectors

Count a drift event if drift exceeds a threshold for >= M consecutive steps.

If enabling any learned KV projection (C2C), report drift both (i) versus the no-transfer reference and (ii) versus a same-model KV reuse baseline, since the projector can introduce systematic bias.

Failure modes (mandatory) 1. NaN/Inf in logits/entropy/embeddings 2. Timeout: exceeded `max_steps` or `max_handoffs` without a scorable answer 3. Decode/format failure: no parseable final answer 4. Code runtime failure: compilation error, timeout, runtime exception

Report failure rate per benchmark and per ablation.

6.5 Experiment Registry and Reproducible Runner

6.5.1 Canonical Config Fields All ablation cells use the following normalized configuration fields:

- `alpha`: normalized entropy threshold in $[0, 1]$, where $H_{\text{norm}} = H/\log|V|$
- `w`: trend window size (steps); `null` if unused
- `cap`: max number of mode switches per sample; `-1` = unlimited
- `k`: top-k for mixture/belief packets; `-1` = "full vocab" (logits transmission)
- `tau`: Gumbel-Softmax temperature; `null` if unused
- `lambda`: Dirichlet concentration multiplier; `null` if unused

Global defaults (unless overridden): - `alpha=0.60`, `w=null`, `cap=10`, `k=50`, `tau=null`, `lambda=null`

6.5.2 Experiment ID Tables (Exact Fields)

A1 — Soft Token Variants

ID	Variant	alpha	w	cap	k	tau	lambda
A1.1	Deterministic top-k mixture	0.60	null	10	50	null	null
A1.2	Gumbel-Softmax	0.60	null	10	50	0.5	null
A1.3	Gumbel-Softmax	0.60	null	10	50	1.0	null
A1.4	Gumbel-Softmax	0.60	null	10	50	2.0	null

ID	Variant	alpha	w	cap	k	tau	lambda
A1.5	Dirichlet noise injection	0.60	null	10	50	null	20
A1.6	Dirichlet noise injection	0.60	null	10	50	null	50
A1.7	Dirichlet noise injection	0.60	null	10	50	null	100

A2 — Controller Variants

ID	Variant	alpha	w	cap	k	tau	lambda
A2.1	Threshold gate (naive)	0.60	null	-1	50	null	null
A2.2	Trend gate	0.60	3	-1	50	null	null
A2.3	Trend gate	0.60	5	-1	50	null	null
A2.4	Trend gate	0.60	10	-1	50	null	null
A2.5	Trend gate + switch cap	0.60	5	2	50	null	null
A2.6	Trend gate + switch cap	0.60	5	5	50	null	null
A2.7	Trend gate + switch cap	0.60	5	10	50	null	null

A3 — Communication Protocols

ID	Variant	alpha	w	cap	k	tau	lambda
A3.1	Raw embedding vectors (same-model)	0.60	null	10	50	null	null
A3.2	Soft belief packets	0.60	null	10	10	null	null

ID	Variant	alpha	w	cap	k	tau	lambda
A3.3	Soft belief packets	0.60	null	10	50	null	null
A3.4	Soft belief packets	0.60	null	10	100	null	null
A3.5	Full logits transmission (upper bound)	0.60	null	10	-1	null	null

A4 — KV / Cache Compression

ID	Variant	alpha	w	cap	k	tau	lambda
A4.1	No compression (fp16/bf16)	0.60	null	10	50	null	null
A4.2	KIVI 2-bit only	0.60	null	10	50	null	null
A4.3	MiniCache only	0.60	null	10	50	null	null
A4.4	KIVI + MiniCache combined	0.60	null	10	50	null	null

6.5.3 Minimal Runner CLI Spec We define a single entrypoint `dlmom` with three subcommands.

Run a suite (auto-enumerate + execute)

```
dlmom run \
  --suite A1 \
  --model Qwen/Qwen2.5-7B-Instruct \
  --bench gsm8k \
  --out runs/2025-12-15_A1 \
  --device hip \
  --dtype bf16 \
  --seeds 0 1 2 \
  --max-steps 256 \
  --max-handoffs 2 \
  --max-switches 10 \
  --batch-size auto \
  --resume
```

Required behavior

- Enumerate experiment IDs for the suite (A1.* etc.)
- Materialize `runs/.../configs/<ID>.yaml`
- Execute each experiment and write:
  - `runs/.../raw/<ID>.jsonl` (per-sample)
  - `runs/.../summary/<ID>.json` (aggregate)
- Run a fixed **reference** configuration first (`REF` = Unified Default, no KV compression) for consistent drift computation.

Aggregate results (CSV + LaTeX-ready)

```
dlmom aggregate \
  --in runs/2025-12-15_A1 \
  --out runs/2025-12-15_A1/aggregate.csv
```

Generate plots (paper figures)

```
dlmom plot \
  --in runs/2025-12-15_A1 \
  --out runs/2025-12-15_A1/figs \
  --format png
```

Required plots

1. `accuracy_vs_total_steps.png` (Pareto)
2. `accuracy_vs_wallclock.png` (Pareto)
3. `drift_histogram.png` (KL or cosine drift vs REF)
4. `switch_count_cdf.png` (A2 only)
5. `bandwidth_vs_accuracy.png` (A3 only)
6. `memory_vs_throughput.png` (A4 only)

6.5.4 Grid Specification (`configs/suites.yaml`)

```yaml
defaults:
  alpha: 0.60
  w: null
  cap: 10
  k: 50
  tau: null
  lambda: null

suites:
  A1:
    - id: A1.1
      soft_mode: deterministic
    - id: A1.2
      soft_mode: gumbel
      tau: 0.5
    - id: A1.3
      soft_mode: gumbel
      tau: 1.0
    - id: A1.4
      soft_mode: gumbel
      tau: 2.0
    - id: A1.5
      soft_mode: dirichlet
      lambda: 20
    - id: A1.6
      soft_mode: dirichlet
      lambda: 50
    - id: A1.7
      soft_mode: dirichlet
      lambda: 100

  A2:
    - id: A2.1
      gate: threshold
      cap: -1
    - id: A2.2
      gate: trend
      w: 3
      cap: -1
    - id: A2.3
      gate: trend
      w: 5
      cap: -1
    - id: A2.4
      gate: trend
      w: 10
      cap: -1
    - id: A2.5
      gate: trend
      w: 5
      cap: 2
    - id: A2.6
      gate: trend
      w: 5
      cap: 5
    - id: A2.7
      gate: trend
      w: 5
      cap: 10

  A3:
    - id: A3.1
      comm: embed
    - id: A3.2
      comm: belief
      k: 10
    - id: A3.3
      comm: belief
      k: 50
    - id: A3.4
      comm: belief
      k: 100
    - id: A3.5
      comm: logits
      k: -1

  A4:
    - id: A4.1
      kv: none
    - id: A4.2
      kv: kivi2bit
    - id: A4.3
      kv: minicache
    - id: A4.4
      kv: minicache+kivi2bit
```

For `--batch-size auto`, the runner should probe batch size and record the chosen value in `summary/<ID>.json`.

---

7. Implementation Plan for gfx1151 + 128GB RAM

7.1 Hardware and Runtime Assumptions

- gfx1151 (ROCm/HIP backend)
- 128GB system RAM enables larger KV caches and higher batch sizes, but peak device memory must still be measured to avoid hidden paging.

7.2 Execution Plan

1. **Environment pinning**
   - Record: `uname -r`, `rocminfo`, `rocm-smi`, `torch.__version__`, `torch.version.hip`
   - Keep `torch.compile` off for stability
   - Fix seeds unless stochastic ablation requires sampling
2. **Config-driven harness**
   - Store `configs/<ID>.yaml` per experiment
   - Write per-sample JSONL + per-run summary JSON
3. **Instrumentation**
   - per sample: timestamps, explicit/latent counts, switch/handoff counts, peak device mem, peak RSS, drift stats, failure label
   - export traces (entropy + mode) for a small qualitative subset
4. **Scheduling**
   - Run order: baselines → reference → A1 → A2 → A3 → A4
   - Pilot runs (32–128 items) before full splits
5. **Fair budgeting**
   - Fix `max_steps`, `max_handoffs`, `max_switches` for all comparable conditions
   - For self-consistency and ToT, report compute multiplier explicitly

---

8. Limitations and Future Work

8.1 Current Limitations

- **Tokenizer dependency.** Soft Belief Packets require compatible vocabularies. Text-Bridge fallback preserves structure but reduces soft information on incompatible links.
- **Memory overhead.** Maintaining KV caches for multiple experts remains expensive; practical deployment on consumer GPUs may require aggressive quantization and limiting expert count.
- **Heterogeneous KV transfer is not training-free.** Cross-architecture KV-cache transfer requires learned bridging (e.g., cache-to-cache projection) or must be restricted to intra-family settings where lightweight alignment suffices. DL-MoM's training-free guarantee applies to belief routing + consensus + controller + compression, not to heterogeneous KV transfer.

- **Evaluation scope.** This paper provides architecture + protocol + reproducibility tooling. Full empirical validation is defined but not yet reported here.

8.2 Future Directions

- **Learned routing (optional).** Lightweight learned routers may improve expert selection while maintaining low overhead.
- **Hierarchical ensembles.** Tree-structured expert organizations with meta-experts coordinating sub-groups.
- **Multimodal extension.** Extend belief packets to VLM token spaces and multimodal preference representations.
- **Theoretical foundations.** Formal bounds on approximation error induced by top-k truncation and consensus merging.

---

9. Conclusion

DL-MoM is a training-free architecture for latent-space collaboration among LLM experts. By routing **probabilistic belief packets** and merging expert preferences via a **contrastive TIES-style consensus**, DL-MoM preserves uncertainty that text-based multi-agent systems discard and enables adaptive compute via trend-based entropy switching. We provide a complete implementation blueprint and a reproducible experimental registry to evaluate accuracy–efficiency–stability trade-offs across benchmarks.

References

[1] Hao, S., et al. (2024). *Training Large Language Models to Reason in a Continuous Latent Space.* arXiv:2412.06769.

[2] Zhang, Y., et al. (2025). *Soft Thinking: Unlocking the Reasoning Potential of LLMs in Continuous Concept Space.* arXiv:2505.15778.

[3] Shi, W., et al. (2025). *SwiReasoning: Switch-Thinking in Latent and Explicit for Pareto-Superior Reasoning LLMs.* https://swireasoning.github.io/

[4] *Enabling Agents to Communicate Entirely in Latent Space.* arXiv:2511.09149 (2025).

[5] *LLMs are Single-threaded Reasoners: Demystifying the Working Mechanism of Soft Thinking.* arXiv:2508.03440 (2025).

[6] Zou, H., et al. (2025). *Latent Collaboration in Multi-Agent Systems.* arXiv:2511.20639.

[7] *SIM-CoT: Supervised Implicit Chain-of-Thought.* OpenReview (2025).

[8] Zhang, A., et al. (2024). *MiniCache: KV Cache Compression in Depth Dimension for Large Language Models.* arXiv:2405.14366.

[9] Liu, Z., et al. (2024). *KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache.* arXiv:2402.02750.

[10] *ChunkKV: Semantic-Preserving KV Cache Compression for Efficient Long-Context LLM Inference.* arXiv:2502.00299 (2025).

[11] Yadav, P., et al. (2023). *TIES-Merging: Resolving Interference When Merging Models.* NeurIPS 2023.

[12] Shazeer, N., et al. (2017). *Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer.* ICLR 2017.

[13] Wei, J., et al. (2022). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models.* NeurIPS 2022.

[14] Yao, S., et al. (2023). *Tree of Thoughts: Deliberate Problem Solving with Large Language Models.* NeurIPS 2023.

[15] Tishby, N., Pereira, F. C., & Bialek, W. (1999). *The Information Bottleneck Method.* arXiv:physics/0004057.

[16] Shwartz-Ziv, R., & Tishby, N. (2017). *Opening the Black Box of Deep Neural Networks via Information.* arXiv:1703.00810.

[17] *A Mathematical Theory of Top-k Sparse Attention via Total Variation Distance.* arXiv:2512.07647 (2025).

[18] Weiss, Y., & Freeman, W. T. (2001). *Correctness of Belief Propagation in Gaussian Graphical Models of Arbitrary Topology.* Neural Computation, 13(10), 2173–2200.

[19] Moallemi, C. C., & Van Roy, B. (2006). *Consensus Propagation.* IEEE Transactions on Information Theory, 52(11), 4753–4766.

[20] Zhu, H., et al. (2025). *Reasoning by Superposition: A Theoretical Perspective on Chain of Continuous Thought.* arXiv:2505.12514.

[21] *Spectral Structure of Task Vectors for Model Merging and Ensembling.* arXiv:2412.12153 (2024).

[22] Fu, T., Min, Z., Zhang, H., Yan, J., Dai, G., Ouyang, W., & Wang, Y. (2025). *Cache-to-Cache: Direct Semantic Communication Between Large Language Models.* arXiv:2510.03215. https://github.com/thu-nics/C2C

Appendix A: Comparison with Existing Approaches

Feature	Text MAS	LatentMAS	InterLat	Soft Thinking	DL-MoM (Ours)
Bandwidth	Low	High	High	Medium	High
Training-Free	Yes	No*	No	Yes	Yes
Heterogeneous	Yes	Limited	Limited	No	Yes†
Reasoning	Linear	Linear	Linear	BFS-like	BFS-like
Consensus	Voting	Alignment (W_a)	Adapters	N/A	Contrastive TIES
Adaptive Depth	No	No	No	No	Yes

* LatentMAS requires alignment estimation and does not provide a fully training-free heterogeneous protocol.

† Requires compatible tokenizers; Text-Bridge fallback for mismatched vocabularies.

Appendix B: Hyperparameter Recommendations

Parameter	Recommended	Notes
`alpha` (entropy threshold)	0.60	Use normalized entropy (H/log|V|)
`w` (entropy window)	5	Trend window for slope/variance
`cap` (max switches)	5	Prevent oscillation / overthinking
`top_k` / `k` (belief packet)	50	Balance information vs sparsity
`max_latent_steps`	40	Upper bound on latent depth
`max_handoffs`	2	Avoid runaway multi-agent loops
`gumbel_temperature` / `tau`	1.0	Lower = more deterministic

Parameter	Recommended	Notes
`dirichlet_lambda` / `lambda`	50	Higher = less noise; tune via target KL
`trim_threshold`	0.1	TIES noise removal
`convergence_threshold`	0.15	Normalized entropy for exit

Appendix C: Python Implementation Sketch (Minimal)

```python
import torch
import torch.nn.functional as F

class DeepLatentMoM(torch.nn.Module):
    def __init__(self, experts, tokenizer, top_k=50):
        super().__init__()
        self.experts = torch.nn.ModuleList(experts)
        self.tokenizer = tokenizer
        self.top_k = top_k

    def reconstruct_soft_input(self, ids, probs, expert_idx):
        expert = self.experts[expert_idx]
        emb = expert.get_input_embeddings()(ids)          # [B, K, D]
        x = torch.matmul(probs.unsqueeze(1), emb).squeeze(1)  # [B, D]
        return x.unsqueeze(1)                             # [B, 1, D]

    def normalized_entropy(self, logits):
        p = F.softmax(logits, dim=-1)
        h = -torch.sum(p * torch.log(p + 1e-9), dim=-1)
        return h / torch.log(torch.tensor(logits.shape[-1], device=logits.device))

    def contrastive_ties(self, logits_list, trim_thresh=0.1, eps=1e-6):
        centered = [l - l.mean(dim=-1, keepdim=True) for l in logits_list]
        stacked = torch.stack(centered, dim=0)                 # [K, B, V]
        mask = (stacked.abs() > trim_thresh).float()
        trimmed = stacked * mask

        vote = torch.sign(torch.sign(trimmed).sum(dim=0))       # [B, V]
        agree = (torch.sign(trimmed) == vote).float()

        count = torch.clamp(agree.sum(dim=0), min=1.0)
        merged = (trimmed * agree).sum(dim=0) / (count + eps)
        return merged                                           # [B, V]

    def belief_packet_from_logits(self, logits):
        probs = F.softmax(logits, dim=-1)
        top_probs, top_ids = torch.topk(probs, self.top_k, dim=-1)
        top_probs = top_probs / top_probs.sum(dim=-1, keepdim=True)
        return top_ids, top_probs
```

```python
def forward_step(self, packet, kv_cache):
    ids, probs = packet
    outputs = []
    for i, expert in enumerate(self.experts):
        x = self.reconstruct_soft_input(ids, probs, i)
        out = expert(inputs_embeds=x, past_key_values=kv_cache[i], use_cache=True)
        logits = out.logits[:, -1, :]
        outputs.append((logits, out.past_key_values, self.normalized_entropy(logits)))

    merged = self.contrastive_ties([o[0] for o in outputs])
    next_packet = self.belief_packet_from_logits(merged)
    avg_entropy = torch.stack([o[2] for o in outputs]).mean()
    next_kv_cache = [o[1] for o in outputs]
    return next_packet, next_kv_cache, avg_entropy
```

Appendix D: Proof Sketches (Selected)

- **Top-k KL bound (Eq. 1).** For the renormalized top-k truncation $\hat{p}_k$ with tail mass τ, the divergence admits the closed form

$$\mathrm{KL}(\hat{p}_k \| p) = \log \frac{1}{1-\tau},$$

  since $\hat{p}_k = p/(1-\tau)$ on its support. This can be related to total variation via $\mathrm{TV}(p, \hat{p}_k) = 1 - e^{-\mathrm{KL}}$.
- **Consensus deviation bound (§5.2).** Trimming sets $|v_{i,j}| < \theta$ to zero, and sign election partitions experts into sign-consistent sets. Disjoint merging averages within each set, yielding a deviation bound controlled by within-set dispersion and the fraction of sign conflicts ρ.
- **Trend gate variance reduction (§5.4).** Under an additive-noise model $H_t = H^* + \epsilon_t$ with i.i.d. ϵ_t, a windowed slope estimator over w steps has variance $O(\sigma^2/w)$, reducing premature stopping relative to a single-sample threshold test.