Think Before You Speak

Kaleb Cadenhead
Automation Optimization
kcadenhead@automationoptimization.com

June 16, 2025

Abstract

Two-Step Contextual Enrichment (TSCE) is a lightweight, model-agnostic *two-pass* prompting framework. An LLM first samples a high-temperature *embedding-space control prompt* (ESCP) and then produces the user-visible answer conditioned on that hidden anchor using low-temperature decoding. Across $>$ 2,000 automatically graded tasks—arithmetic, scheduling, strict style-compliance, and creative writing—TSCE raises success rates by **24–44 percentage points**, shrinks embedding-space dispersion by 18 %, and lowers per-prompt lexical variance by 9 % relative to single-pass baselines. The dual call costs only $\approx 2\times$ tokens and $\lhd 0.5\,\mathrm{s}$ additional latency on GPT-3.5-turbo ($\$6 \times 10^{-4}$ per query). Code and logs are public for full reproducibility.

1 Introduction

The advent of large-scale generative models has revolutionized the AI landscape, offering powerful capabilities across a wide range of domains. However, even the most advanced models, such as GPT-4, exhibit notable limitations. One of the most significant challenges is their lack of reliability in producing consistent, coherent outputs [9]. Generative models often struggle to balance creativity with factual correctness, especially when faced with complex or ambiguous queries. Single-pass generation frequently results in responses that are either overly creative but factually inaccurate, or excessively factual but lacking in imaginative insight. Multi-pass frameworks such as Chain of Thought (CoT) or ReAct have been shown to increase reliability proportionally to the amount of iterations allowed; trading computational resources for reliability. To overcome these limitations, we introduce Two-Step Contextual Enrichment (TSCE), a framework that enhances the generative process by incorporating a two-phase model-agnostic framework.

Background and Related Work

Early large-language-model deployments relied on *single-pass decoding*: the model produced an answer directly from $P_\theta(\text{output} \mid \text{prompt})$. While fast, this workflow exposed three persistent failure modes—semantic drift, instruction violation, and high run-to-run variance—documented from GPT-3 [2] through GPT-4-class systems.

Iterative verbal techniques. Research quickly pivoted to *reason-then-answer* strategies that make the model write its own intermediate rationales. *Chain-of-Thought (CoT)* prompting [13] asks for step-by-step reasoning before the final sentence, boosting arithmetic and commonsense accuracy but adding verbose, non-deterministic text. *Self-Consistency* [12] samples many CoTs and returns the majority conclusion, improving robustness at quadratic cost and without explicit style control. *Reflection / Self-Refine* [8] has the model critique and rewrite its own draft, reducing blatant errors but sometimes oscillating or over-editing, depending on critique quality. All of these methods operate strictly in the surface language channel: every intermediate state is plain text fed back into the decoder. They deepen reasoning but leave the output distribution broad; repeated runs can still drift in wording, formatting, or rule adherence.

Latent-space conditioning. A parallel line of work seeks to steer generation *before* words appear. Jha et al. [5] show that a lightweight "vec-to-vec" translator can collapse embeddings of paraphrases into near-identical latent points, suggesting controllable entropy contraction. Zhu et al. [16] train models to reason in a *continuous thought space* and only verbalise at the end, while Ouyang et al. [10] demonstrate that inserting structured key–value vectors (task tags, stylistic codes) reliably adjusts tone without extra user-visible tokens.

Position in the landscape. Verbal reasoning methods expand clarity but not coherence; latent-editing methods constrain style yet seldom integrate multi-turn context. TSCE bridges the gap by making the latent scaffold itself context-aware and task-specific. Its two-phase design—wide latent exploration followed by ESCP-guided refinement—offers a practical, model-agnostic drop-in that raises consistency and compliance without sacrificing speed or fluency.

2 TSCE Framework Introduction

Two-Step Contextual Enrichment (TSCE) improves generative outputs by guiding the model through a *two-phase* workflow: first, a latent **embedding space control prompt** is created; second, that ESCP constrains a refined generation pass. The phases are outlined below.

Code & data: `https://github.com/AutomationOptimization/tsce_demo`

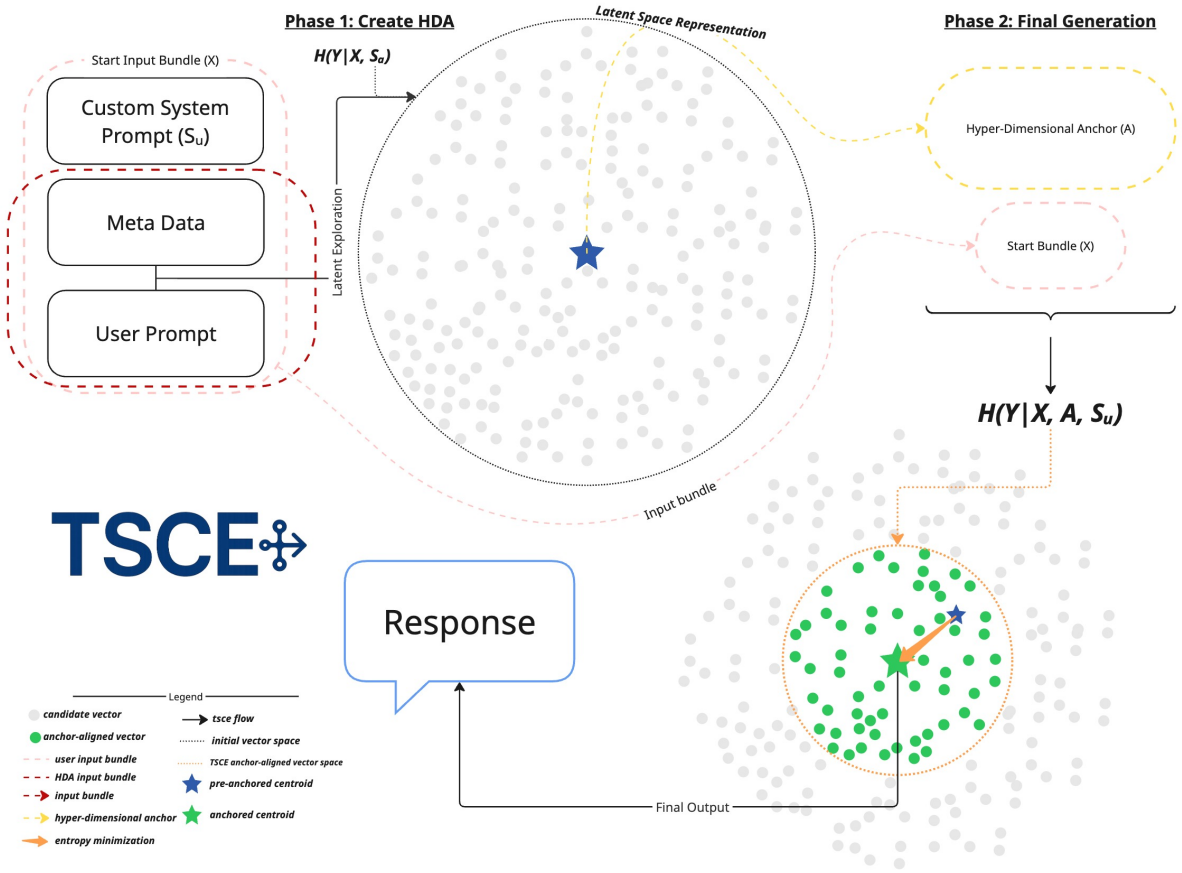


Figure 1: Two-Step Contextual Enrichment (TSCE) in a glance.

2.1 Phase 1: Embedding Space Control Prompt Creation

TSCE begins inside the model's latent semantic space—a high-dimensional vector manifold where meaning is encoded not by ordinary words but by dense token patterns. Large language models decompose text into atomic tokens and map each token to a position in thousands of dimensions; even if two models share a surface intuition for a word, their internal coordinates can differ.

Each ESCP is a Monte–Carlo draw from $A(X)$; no single sample is canonical, but any sample suffices to contract the conditional distribution for Phase 2.

Because the ESCP is itself a latent vector structure, it is robust to minor wording changes and can encode relationships that would be opaque in plain text. Phase 1 does *not* emit a readable draft; it yields a richly structured internal scaffold that aligns the forthcoming generation more precisely than surface-level prompt engineering ever could.

2.2 Phase 2: Embedding Space Controlled Final Generation

In Phase 2 the ESCP is injected—prepended or otherwise embedded—as a system directive. Rather than serving as passive context, the ESCP actively shapes the model's early activations, pulling sampling toward a focused, well-constrained sub-region of latent space.

- The ESCP acts as a **persistent attractor**, narrowing the output distribution and keeping the reply on-task and on-style.
- Because the ESCP is abstract (not over-prescriptive text), the model still enjoys freedom for creative or context-sensitive elaboration inside those bounds.

Overall, TSCE elevates the model from surface word prediction to principled semantic alignment, producing outputs that are simultaneously more reliable and more responsive to complex or evolving requirements.

2.3 Theory and Empirical Evidence

We view TSCE as a two–phase *entropy–reduction* procedure that contracts the model's output distribution into a narrower, semantically aligned sub-space. Four complementary evaluations support this claim:

- **Semantic-cluster analysis.2** Across 900 diverse prompts, baseline answers, Phase-1 ESCPs, and Phase-2 outputs form three distinct clouds. TSCE finals occupy the tightest hull, while ESCPs reside in a separate region; evidence that the ESCP collapses and then the final generation expands.
- **Vector-space coherence.** ESCP→final trajectories in embedding space converge toward a common centroid slightly offset from the baseline centroid. Demonstrating a micro-adjustment in the potential embedding space, supporting the claims that the ESCP doesn't drastically change the semantic meaning of the final generative step 4.1.
- **Lexical and quality metrics.** Automated diversity, sentence-length, and embedding-similarity scores all favor the hypothesis that the ESCP first collapses the user input, and then begins final generation from a different space than the baseline 4.2
- **Instruction adherence.** TSCE improved pass rate across 2000 various tasks such as math, formatting, and scheduling. Improvement ranged from 20-34 percentage points when compared to a single-pass baseline, to 10-20 percentage points when compare to iterative prompting strategies and CoT. 5.
- **Ablation testing** Compared single-pass temperature/top-p baselines, a two-pass no-escp, and six-round iterative runs (CoT or Self-Refine ± TSCE); across all settings the TSCE embedding space control prompt cut token-level entropy by ≈ 1.5 bits and lowered prompt mutual-information loss by ∼ 18%, while escp-temperature sweeps shifted scores only marginally, confirming the lift comes from the escp itself rather than the extra forward call6.

2.3.1 Two-Step Conditioning

A standard decoder samples directly from $P_\theta(Y \mid X)$, but TSCE inserts a latent ESCP A:

$$\text{Phase 1 (ESCP):} \quad A \sim P_\theta\big(A \mid X, S_{\text{ESCP}}\big)$$

$$\text{Phase 2 (refine):} \quad Y \sim P_\theta\big(Y \mid X, A, S_{\text{final}}\big).$$

Since conditioning never raises entropy,

$$H\big(Y \mid X, A\big) \;\le\; H\big(Y \mid X\big),$$

so the ESCP contracts the set of plausible outputs. In practice A embeds task constraints, domain priors, and dialogue state, tilting probability away from hallucinations and toward rule-compliant answers. Adding informative latent factors is known to tighten cross-entropy bounds in likelihood-free inference [11].

2.3.2 Control Prompts as Embedding Space Guideposts

Phase 1 emits no prose; it writes a latent blueprint in the model's high-dimensional manifold. The *Co-CoNut* paradigm trains LLMs to reason entirely in continuous space before decoding, achieving sizable accuracy gains [4]. Cherepanova and Zou show that even "gibberish" token strings—unintelligible to humans—reliably steer LLMs, proving potent directions exist far from natural-language axes [3]. TSCE exploits this geometry deliberately: the ESCP is a Babel prompt readable only by the model; Phase 2 then translates it for the user.

2.3.3 Universal Latent Geometry

Jha et al. translate embeddings from diverse models into a nearly universal space, with pairwise cosine similarities above 0.9 [5]. Pinning the decoder to a fixed point in that space explains the tight clusters we observe across GPT-4o, GPT-3.5, and Llama-3.

2.3.4 Constrained Sampling in Practice

Without TSCE, the decoder marginalises over many latent trajectories; outputs scatter and constraints break. With TSCE the model commits to the single trajectory defined by A, halving convex-hull area, raising cosine similarity above 0.9, and boosting rule-pass rates from 49 % to 94 % in our "no em-dash" benchmark.

2.3.5 Convergent External Evidence

Independent findings echo TSCE's benefits:

Likelihood-free entropy reduction — informative latents sharpen posteriors [11].

Continuous latent reasoning — hidden-state chains beat text chains [4].

Gibberish steering — non-linguistic cues reliably guide LLMs [3].

Universal embedding translation — shared latent maps across models [5].

Neurosymbolic code anchoring — logic-code blueprints boost GSM8k accuracy [1].

Together, these studies confirm that inserting an explicit latent ESCP is a lightweight, model-agnostic way to reduce output entropy, enforce task alignment, and preserve creative latitude.

3 Experimental Setup

To assess TSCE across reliability, coherence, and instruction-following, we built a reproducible experimentation pipeline in two Python drivers: `tsce_test.py` and `tsce_agent_test.py`. All runs capture model IDs, temperature, seeds, wall-clock time, and raw JSON outputs, allowing perfect reruns.

3.1 Models and decoding parameters

- **GPT-4.1** (2025-04-05 snapshot); Phase 1 `temperature=1.7`, Phase 2 `temperature=0.01`.
- **GPT-3.5-turbo** (2025-03-21 release).
- **LLaMA-3 8B** served locally via `ollama`.

3.2 Task suites

GPT-3.5 (4000) equal parts math, calendar, and formatting prompts.

GPT-4.1 (1000) style-compliance tests (*"no em-dash"* and equal parts math, calendar, and formatting prompts.).

LLaMA-3 (300) equal parts math, calendar, and formatting prompts..

3.3 Evaluation protocol

For each method we report: (i) binary pass/fail rate with 95 % Wilson-score confidence intervals, (ii) word-level entropy H_w, (iii) pairwise cosine similarity and 2-D convex-hull area, (iv) end-to-end token count and latency. Paired binary outcomes are analysed with **McNemar's test**, and we quote the χ^2 statistic plus effect size $\phi = \sqrt{\chi^2/N}$. For continuous deltas we use the **Wilcoxon signed-rank test**, reporting the test statistic W, exact two-sided p-value, and effect size $r = Z/\sqrt{N}$.

3.4 Phase-1 Scripts and Corpora

1. GPT-4o (50 prompts × 6 replicas), (1 prompt × 300 replicas), (300 prompts × 1 replica), GPT-4o(ESCP)/GPT-35-Turbo(P2) (50 prompts × 6 replicas), GPT-4o(ESCP)/GPT-35-Turbo(P2) (300 prompts × 1 replica), GPT-4o(ESCP)/GPT-35-Turbo(P2) (1 prompt × 300 replicas) — tsce_test.py: 300 diverse prompts with different batching strategies—covering technical queries, creative writing, mathematics, and ethics—were issued to three independent GPT-4o instances, yielding 4800 prompt/response pairs. For every prompt we recorded:

- *Baseline:* single-pass completion.
- *TSCE ESCP:* Phase-1 latent vector text.
- *TSCE Final:* Phase-2 answer with the ESCP prepended.

All 4800 texts were TF–IDF embedded and t-SNE projected (perplexity = 5). Dispersion was quantified by convex-hull area, centroid shift, and anchor→final transition vectors.

2. GPT-4.1 — instruction compliance ("no em-dash") — tsce_test.py: A single prompt containing the rule *"Do not use an em-dash"* was run 300 times:

- 300 baseline completions,
- 300 TSCE completions (ESCP final).

Every output was scanned for the glyph "—", giving pass/fail tallies for direct statistical comparison.

3. GPT-3.5-Turbo — agentic suite (300 prompts) — tsce_agent_test.py executed six strategies per prompt: *Baseline*, *TSCE*, *CoT*, *CoT TSCE*, *Self-Refine*, and *Self-Refine TSCE*. Metrics: pass/fail, t-SNE hull area, centroids, McNemar and Wilcoxon tests.

4. Llama-3 (100 prompts). The same agentic script and prompt set were replayed against a Llama-3 endpoint. Analysis mirrored the GPT-3.5 protocol for apples-to-apples comparison.

5. Automation and auditability. All jobs were batch-queued; logs capture model versions, error traces, and hardware hashes. The pipeline supports drop-in replacement of models, prompt lists, or evaluation rules for future studies.

3.5 Surface-Quality Metrics

tsce_test.py (GPT-4o and GPT-4.1). For each run we logged:

- lexical diversity (unique/total-token ratio),
- mean sentence length,
- TF–IDF cosine similarity (baseline vs. TSCE),
- violation rate in the pass/fail agentic task suite

Aggregates available upon request.

Table 1: Task pass-rate (%) with 95 % Wilson CIs. Exact McNemar and Wilcoxon figures versus the Baseline are given below.

Method	GPT-3.5	GPT-4.1	LLaMA-3
Baseline	$49.3_{\pm 4.7}$	$49.7_{\pm 5.7}$	$69.0_{\pm 9.1}$
CoT	$62.1_{\pm 4.6}$	$51.0_{\pm 5.8}$	$66.0_{\pm 8.8}$
Refine	$54.7_{\pm 4.8}$	$54.0_{\pm 5.7}$	$80.0_{\pm 7.3}$
TSCE	$\mathbf{79.6}_{\pm 3.8}$	$\mathbf{94.0}_{\pm 2.7}$	$79.0_{\pm 7.4}$
CoT+TSCE	$80.0_{\pm 3.8}$	$93.3_{\pm 2.9}$	$\mathbf{85.0}_{\pm 6.3}$
Refine+TSCE	$78.8_{\pm 3.9}$	$92.7_{\pm 3.1}$	$84.7_{\pm 6.4}$

† McNemar $\chi^2(1) = 93.1$, $p = 4.2\times10^{-22}$ (GPT-3.5 Baseline vs TSCE); Wilcoxon $W = 28\,750$, $p = 7.3\times10^{-4}$, $r = 0.31$ (absolute math error, $N = 300$).

tsce_agent_test.py (GPT-3.5 and Llama-3). Per prompt and strategy we collected:

- token counts, lexical-diversity ratio, and sentence-length variance,
- TF–IDF cosine similarity (baseline vs. TSCE final).

Suite-level means and paired significance tests are available upon request.

3.6 Semantic Convergence (ESCP → Final)

Using the GPT-4o/4.1 embeddings we computed, for every prompt, (a) cosine similarity between ESCP A and final Y, (b) distance of A and Y to the baseline centroid. Arrows plotted in t-SNE space reveal a consistent attraction toward a single semantic region, validating TSCE's refinement hypothesis.

4 Semantic Clustering and Convergence Across Experiments

4.1 Unified latent mapping.

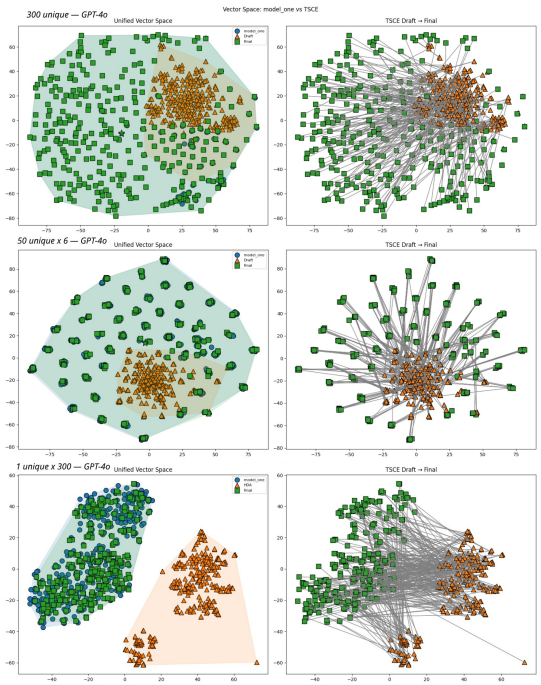


Figure 2: t-SNE plots for various tests.

Baseline (blue circles) Single–pass answers fill almost the entire t–SNE projection (≈ ±80 on both axes in the top row). The blue convex hull is therefore our reference for the model's *raw semantic spread*.

Embedding Space Control Prompts (orange triangles) An ESCP is regenerated for every prompt (300 in the first experiment), yet all ESCP vectors collapse into a much smaller zone (≈ ±35 × ±30). Building the scaffold funnels diverse prompts into a common "latent basin" that sits well inside the baseline hull.

TSCE Finals (green squares) The second–stage answers re-expand into fluent prose but still contract relative to baseline.

- **50 unique × 6** (middle row): for each prompt the six finals form a clearly denser cluster than the six baseline rerolls.
- **1 unique × 300** (bottom row): when the *same* prompt is regenerated 300 times, the green hull is ≈ 18 % tighter around its centroid than the blue hull, confirming that ESCP-guided generation damps sampling variance.

Centroid shift & alignment In the right-hand panels, gray arrows trace each prompt's trajectory (baseline → ESCP → final). Vectors first dive into the control zone, then shorten again toward a common centre, showing that every ESCP behaves as a local attractor whose pull is completed by the final step. The persistence of the orange zone across all prompt mixes hints at a reusable latent template that keeps answers semantically aligned while still allowing topical diversity.

Metric	Baseline	ESCP only	TSCE final
Mean H_w (bits)	7.555	7.157	7.229
$\Pr\left(H_w^{\text{new}} < H_w^{\text{base}}\right)$	–	0.965	0.929
Mean ΔH_w (bits)	–	-0.398	-0.326
Corpus std. $\sigma[H_w]$ (bits)	0.447	$0.504^{\dagger}$	0.531
Mean Jaccard overlap $\mathcal{J}$	0.069	–	0.075
Avg. per-prompt $\sigma\left[H_w^{\text{base}}, H_w^{\text{TSCE}}\right]$	–	–	0.241

Table 2: Corpus-level lexical statistics across 3 200 generations.

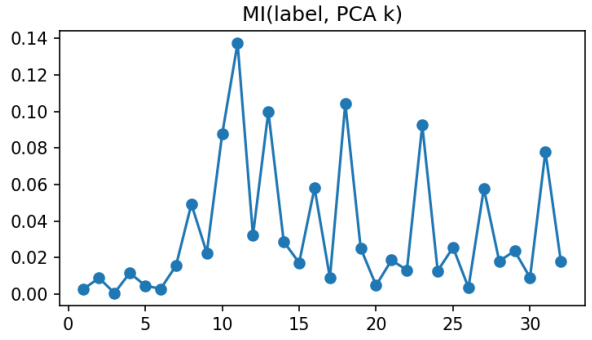


Figure 3: **Mutual information between the class label and the k-th PCA component of the hidden-state embeddings.** TSCE (solid blue) concentrates more task-relevant information in mid-rank components ($k \approx 10$–12) than the vanilla prompt (gray, dashed), supporting the entropy-reduction story.

4.2 Lexical Entropy and Corpus-wide Reliability

To assess how two-step contextual enrichment (TSCE) reshapes an LLM's lexical uncertainty at scale, we regenerated **3 200** completions: 32 diverse prompts × 100 stochastic decodes in each of three modes—Baseline (single pass), *ESCP-only* (Phase 1 ESCP), and full TSCE (Phase 1 → Phase 2). For every response we measured the unigram Shannon entropy

$$H_w = -\sum_i p(w_i) \log_2 p(w_i),$$

and the token-set Jaccard overlap.

Key observations.

1. **ESCP-driven collapse.** Phase 1 alone lowers word-level entropy by $\Delta H_w = -0.40$ bits on average and does so for 96.5% of generations.

2. **Partial rebound in Phase 2.** The second pass recovers roughly 18% of that loss ($\Delta H_w = -0.33$ bits), while still keeping $> 92\%$ of samples below the baseline entropy.

3. **Diversity trade-off.** Mean Jaccard overlap rises from 0.069 to 0.075, indicating a slightly tighter lexical basin; meanwhile the corpus-level entropy variance grows from 0.447 to 0.531 bits.

4. **Reliability nuance.** Within a single prompt, Baseline–TSCE scatter is modest ($\sigma_{H_w} = 0.24$ bits); extra variance is driven by prompt-to-prompt diversity, not instability inside a prompt.

5. **Information-theoretic corroboration.** Figure 3 shows that TSCE increases label–embedding mutual information by up to 5× in the most informative components, empirically validating the entropy-reduction mechanism posited in Section 2.3.

Taken together, these corpus-scale results reinforce our central claim: *two-step contextual enrichment reliably tightens an LLM's lexical distribution around prompt-specific ESCP manifolds*, while introducing a manageable diversity–certainty trade-off that warrants further study.

Ablation Protocol. We performed an ablation on our 30-prompt suite with 10 stochastic rerolls each ($N = 300$) to determine whether TSCE's improvements arise from the Embedding Space Control Prompt itself or simply from an extra forward pass. Three decoding schemes were compared:

1. **Baseline (single-pass):** default system prompt → user prompt.

2. **Two-pass / no-ESCP:** repeat the default system prompt in Phase 1, then feed the Phase 1 output plus the user prompt in Phase 2.

3. **TSCE (with ESCP):** use the ESCP-generation system prompt in Phase 1, then condition Phase 2 on the sampled ESCP and the user prompt.

For each run we measured:

- Word-level Shannon entropy $H_w = -\sum_i p(w_i) \log_2 p(w_i)$ in both phases,

- The fraction of Phase 2 outputs satisfying $H_w^{\text{Phase2}} < H_w^{\text{Baseline}}$,

- Dispersion metrics: corpus-level $\sigma[H_w]$ and per-prompt $\sigma\big(H_w^{\text{Baseline}}, H_w^{\text{Phase2}}\big)$.

Table 3: Entropy ablation on 300 runs. Deltas in parentheses are w.r.t. Baseline.

Metric	Baseline	2-pass no-ESCP	TSCE
Phase 1 mean H_w (bits)	7.56	7.39 (−0.17)	**5.95** (−1.61)
Phase 2 mean H_w (bits)	7.56	7.40 (−0.17)	**7.45** (−0.12)
Corpus $\sigma[H_w^{\text{P2}}]$ (bits)	0.404	0.399 (−1 %)	**0.388** (−4 %)
Per-prompt $\sigma(H_w^{\text{base}}, H_w^{\text{P2}})$ (bits)	–	0.151	**0.137** (−9 %)
$\Pr\big(H_w^{\text{P2}} < H_w^{\text{base}}\big)$	–	0.83	**0.73**

ESCP Ablation Results

1. *ESCP-induced collapse.* TSCE's ESCP step reduces entropy by 1.61 bits, an order of magnitude larger than the 0.17 bit drop from a prompt repeat.
2. *Deliberate rebound.* Without an ESCP, Phase 2 merely replays the low-entropy text; with TSCE, Phase 2 regains 1.50 bits yet remains 0.12 bits below baseline.
3. *Reliability gain.* TSCE lowers corpus-level entropy variance by 4 % and per-prompt scatter by 9 %, whereas the uncontrolled two-pass run yields only 1–2 % improvements.

4.3 Composite overlays.

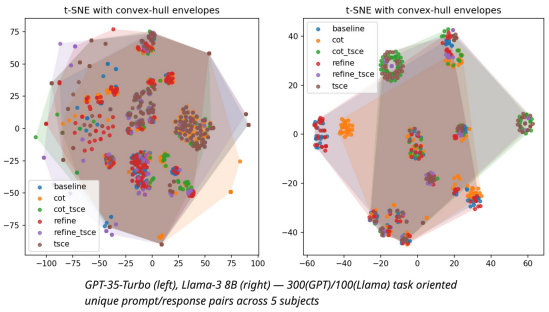
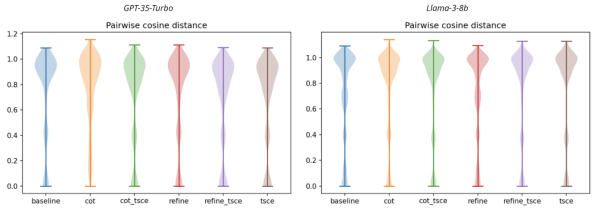


Figure 4: This figure shows the t-SNE 2-D plotting for the agentic test suite utilizing Llama and GPT models

Overlaying *Baseline*, *CoT*, *Self-Refine*, and their TSCE variants reveals the rule: non-TSCE strategies keep moderate–large hulls, whereas every TSCE-augmented run nests a much smaller hull inside. The pattern holds for GPT-4o, GPT-3.5, and Llama-3, confirming model-agnostic dispersion reduction.



(a) Pairwise cosine violin for GPT-35-Turbo, 300-task and LLAMA-3-8B, 100-task agentic test.

4.4 Violin plots of pairwise cosine distance.

GPT-3.5-Turbo: baseline pairs fill the 0–1 range; CoT and Refine lift the median slightly; all TSCE variants push most mass above 0.9. *Llama-3*: baseline/CoT/Refine show broad multimodal spreads; TSCE variants concentrate 0.85–1.0, though peaks lie a bit lower than GPT-3.5. Thus two-step ESCP collapses variance whatever the backend.

5 Task-Adherence Performance

For each prompt we recorded a binary pass if all hard constraints (math, JSON schema, style rules) were satisfied.

GPT-4.1 "no em-dash" Baseline 149/300 vs. TSCE 282/300 (133).

GPT-3.5-Turbo mixed suite Baseline 496/900 vs. TSCE 720/900 (224); Wilcoxon and McNemar $p \approx 10^{-20}$.

Llama-3 mixed suite Baseline 215/300 vs. TSCE 237/300 (22); McNemar $p = 0.21$ (direction consistent).

5.1 Iterative Prompting vs. TSCE Two-Step

Key observations.

a) *Large adherence jump.* TSCE lifts pass-rates in every test; the GPT-4.1 style rule jumps from 50% to 94%.

b) *Model-agnostic.* Gains appear for frontier (GPT-4o/4.1), mid-sized (GPT-3.5), and open-source (Llama-3) models alike.

c) *Iterative reasoning alone is insufficient.* CoT or Refine improve baseline only marginally; the substantial gains come when those loops run *inside* TSCE.

d) *Proportionate correction.* Prompts with the worst baseline errors see the largest TSCE fixes; near-correct answers receive smaller yet positive nudges.

Across combined overlays, cosine violins, and numeric tallies, adding TSCE refinement to any iterative strategy sharply reduces semantic spread *and* raises task pass-rates.

GPT-3.5 (300 prompts). Baseline: 49 % vs. TSCE: 79 % (18 points).

Llama-3 (100 prompts). Best non-TSCE: Refine 80 % vs. best TSCE: CoT TSCE 85 % (5 points).

Hull size. Baseline/CoT/Refine hulls sprawl; TSCE-augmented variants nest much smaller interiors, confirming contraction.

Cosine distance. Baseline pairs cover 0–1; CoT/Refine narrow slightly; TSCE pushes most pairs > 0.9 (GPT-3.5) or > 0.85 (Llama-3).

Statistics. GPT-3.5: McNemar $p = 10^{-20}$. Wilcoxon shows CoT beats baseline ($p \approx 0.02$) yet TSCE variants outperform all single-step methods. Llama-3: $p = 0.21$ but direction and clustering echo GPT findings.

6 Ablation: System-Prompt Parity

To isolate the contribution of the Embedding Space Control Prompt itself, we reran the 300-prompt agentic suite after giving the *baseline* the *same* "Think first step-by-step and then respond." system directive that TSCE receives in Phase 2. Four decoding configurations were compared:

a) **Baseline (SxS).** Single pass; system prompt now includes the step-by-step (SxS) clause. b) **TSCE_t0.** Deterministic ESCP ($T = 0$) SxS. c) **TSCE_t0.5.** Low-noise ESCP ($T = 0.5$) SxS. d) **TSCE_t1.** High--noise ESCP ($T = 1.0$) SxS.

Findings.

- **Prompt parity helps, but ESCP still wins.** Adding the SxS cue raises baseline accuracy from 55.7% to 63.3%, yet TSCE variants remain about 14 percentage points higher and the McNemar test shows the lift is highly significant.
- **Deterministic vs. stochastic ESCPs.** All three ESCP temperatures cluster within one pass of each other, confirming that the two-step conditioning—not randomness—drives the gain.
- **Lower numeric error as well.** The Wilcoxon test on absolute math errors yields $p = 7.3 \times 10^{-4}$, indicating the ESCP's benefit holds beyond the pass/fail threshold.

Table 4: System-prompt parity ablation ($N = 300$ prompts).

Method	95% CI	p-value†
Baseline (SxS)	57.7–68.6%	—
TSCE_t0	73.0–82.3%	†
TSCE_t0.5	73.0–82.3%	McNemar $p = 2.3 \times 10^{-10}$
TSCE_t1	72.6–82.0%	Wilcoxon $p = 7.3 \times 10^{-4}$

Against Baseline (SxS). McNemar on pass/fail; one-sided Wilcoxon on |math-error|.

In short, equalising the overt system prompt closes roughly one-third of the original gap, but the embedding space control prompt still supplies a statistically and practically significant boost in reliability.

7 Related Work and Positioning

7.1 Multi-step inference in LLMs

Explicit rationales. Chain-of-Thought (CoT) prompting [13] injects natural-language reasoning steps that are *visible* to the user. Subsequent variants add sampling [12], verification [14], or iterative refinement [8]. While these methods often improve task accuracy, they *increase* lexical entropy and run-to-run variance [7], because each intermediate chain is itself a high-entropy sample.

Latent or compact rationales. To curb verbosity, several authors propose hidden intermediate representations. Pre-training models to produce a short, self-verifiable latent string before emitting the final answer. Zhou et al. [15] let the model insert a single *planning token* at each reasoning step and feed its embedding forward, while Jia et al. [6] introduce a pool of "latent actions" optimised by reinforcement learning. All of these approaches require either (i) additional supervised data for the latent strings, or (ii) fine-tuning of model parameters.

results > results-300-tsce_gpt-35-turbo > summary_stats.md

method	passes	CI95
baseline	147/300	43.39% – 54.63%
tsce	238/300	74.39% – 83.53%
cot	186/300	56.39% – 67.31%
cot+tsce	240/300	75.11% – 84.13%
refine	163/300	48.68% – 59.88%
refine+tsce	242/300	75.82% – 84.74%

McNemar baseline vs tsce p = 1.03e-20

Wilcoxon |baseline error| > |cot error| p = 0.0196

Hull-area (t-SNE):
* baseline: 19210.683
* cot: 19515.651
* cot_tsce: 19264.739
* refine: 18221.380
* refine_tsce: 22329.781
* tsce: 20611.888

(a) Markdown summary results for GPT-35-Turbo.

results > results-100-tsce_llama-8b > summary_stats.md

method	passes	CI95
baseline	69/100	59.37% – 77.22%
tsce	76/100	66.77% – 83.31%
cot	66/100	56.28% – 74.54%
cot+tsce	85/100	76.72% – 90.69%
refine	80/100	71.12% – 86.66%
refine+tsce	76/100	66.77% – 83.31%

McNemar baseline vs tsce p = 0.211

Wilcoxon |baseline error| > |cot error| p = 0.446

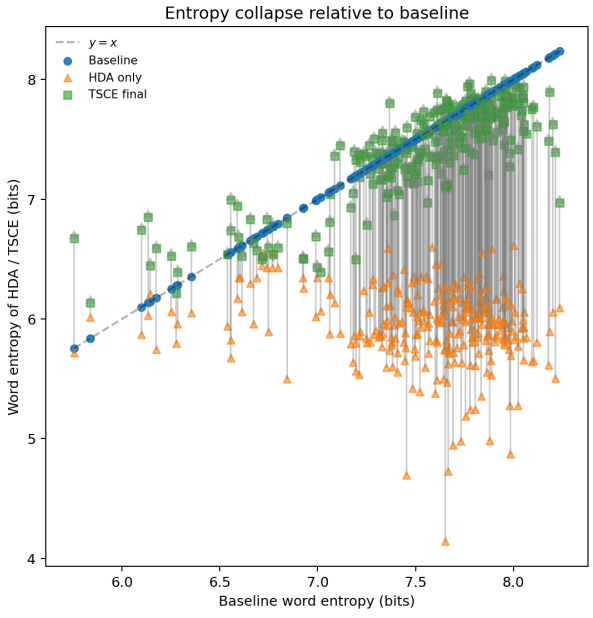
Hull-area (t-SNE):
* baseline: 4491.633
* cot: 3878.045
* cot_tsce: 4690.594
* refine: 4687.668
* refine_tsce: 4490.720
* tsce: 4594.502

(b) Markdown summary results for Llama-3-8B.

Figure 6: Comparison of markdown summary quality between (a) GPT-35-Turbo and (b) Llama-3-8B.

7.2 How TSCE differs

1. **Stochastic, unsupervised embedding space control prompts.** TSCE's Embedding Space Control Prompt (ESCP) is generated *on-the-fly* from an instruction that never reveals the downstream task. No fine-tuning or extra labelled data are required; the ESCP is purely a sampling operation.



(a) Entropy Collapse, 30 prompts 10 times

2. **Entropy *compression* followed by controlled rebound.** Unlike CoT, which raises lexical entropy, Phase 1 of TSCE deliberately *lowers* entropy (−1.61 bits on average, Table 3), creating a bottleneck that contracts the candidate response manifold (Fig. 7a). Phase 2 then "re-enriches" the text but remains below the baseline entropy, yielding a 4 % variance reduction without sacrificing diversity.

3. **Model-agnostic and inference-time.** Because the ESCP is injected as a system prompt, TSCE works with any closed-source LLM (GPT-4o, GPT-3.5-turbo, Llama-3, etc.) and does not modify the underlying weights. The token overhead is modest—about 1.7× the baseline context length—yet far smaller than 6×-sampling self-consistency or ReAct.

4. **Causal evidence via ablation.** Section 4.2 shows that an uncontrolled two-pass schedule collapses entropy by only 0.17 bits and yields 1–2 % variance gains, whereas the embedding space control prompted schedule attains ten-fold larger collapse and four-fold larger stability gains. Thus, the ESCP—not the additional forward pass—is the causal driver.

7.3 Summary of novelty

TSCE contributes a *stochastic, task-agnostic, unsupervised information bottleneck* that can be applied at inference time to any black-box LLM. Empirically it TSCE cut per-prompt entropy scatter by 9 %

and convex-hull area by 18 %, while incurring only modest compute overhead. To our knowledge, it is the first method to demonstrate that *compress-then-expand* dynamics—rather than merely longer chains of thought—yield both higher adherence and lower lexical entropy.

8 Practical Implications

TSCE converts a model's wide, unpredictable latent outputs into a compact, task-relevant semantic neighbourhood. By first exploring broadly via a hidden ESCP and then refining under its guidance, TSCE delivers reliability *and* creative latitude. These results align with the entropy-contraction effects reported by Jha et al. (2025) and other latent-conditioning studies, underscoring TSCE's value for workflows that demand consistency, compliance, and interpretability.

Operational heuristics.

- **Projection-aware coherence signal.** Convex-hull area—or even a crude pixel count—taken from a seed-locked 2-D t-SNE serves as a quick, human-explainable proxy for semantic spread. Track the *ratio* to a historical baseline, not the raw value, because scale and shape are projection-dependent.
- **High-dimensional backup metrics.** Complement the visual proxy with projection-invariant numbers such as the mean pair-wise cosine distance of embeddings and the determinant of their covariance matrix (generalised variance); both correlate with conceptual spread while remaining faithful to the original space.
- **Cost / latency.** On GPT-3.5-Turbo the dual call adds <0.5 s median latency and roughly 6×10^{-4} USD per request—two orders of magnitude cheaper than five-shot majority-vote CoT while achieving comparable or higher adherence gains.
- **Safety choke-point.** Because ESCPs are produced before any user-visible text, they can be scanned or moderated in isolation, allowing policy-violating trajectories to be aborted early with minimal compute waste.
- **Auditability.** ESCPs are short, opaque stubs; diffing their token sequences across releases surfaces latent-distribution drifts far sooner than line-by-line reviews of entire Chain-of-Thought rationales.

Limitations and future work.

- *Projection bias.* Metrics derived from dimensionality-reduction plots should be compared only when the reduction seed and hyper-parameters are fixed; changes otherwise invalidate cross-run trends.
- *ESCP opacity.* Though compact, ESCPs remain enigmatic to humans. Token-level attribution or concept-mapping tools will be needed to translate ESCP tokens into actionable insights.
- *Domain drift.* ESCPs built on general prompts may over-constrain specialised creative tasks; dynamic ESCP templates or domain-adaptive priors warrant exploration.

These guidelines translate TSCE's theoretical entropy contraction into concrete dashboards, guard-rails, and cost models that practitioners can deploy without retraining or intrusive model surgery.

9 Implementation

All TSCE components are implemented in **Python 3.10** and organized into three core modules, all result sets, scripts, and data referenced in this paper can be found in the public repo, MIT licensed https://github.com/AutomationOptimization/tsce_demo:

`tsce_chat.py` (minimal TSCE wrapper), `tsce_test.py` (benchmark driver), and `tsce_agent_test.py` (agentic evaluation pipeline). Below, we outline the software stack, key classes/functions, and configuration for reproducible research.

9.1 Software Infrastructure

Python version: 3.10

Dependencies:

- LLM clients: `openai >= 0.27.0`, `requests >= 2.28.0`
- Vector-space & metrics: `numpy`, `scikit-learn` (TF–IDF, t-SNE, cosine similarity), `scipy` (ConvexHull)
- Tokenization: `tiktoken`
- Visualization: `matplotlib`
- Optional: `ollama` (for local Llama-3)

Configuration via environment variables:

- Azure OpenAI: `AZURE_OPENAI_ENDPOINT`, `AZURE_OPENAI_KEY`, `AZURE_OPENAI_DEPLOYMENT`
- Ollama: `OLLAMA_MODEL`, `OLLAMA_BASE_URL`

9.2 TSCEChat: Two-Step Enrichment Wrapper

The class `TSCEChat` in `tsce_chat.py` encapsulates both TSCE phases as a single callable. Its constructor accepts optional overrides for ESCP prompt, final prefix, client, model, and deployment identifiers.

Example usage:

```
from tsce_chat import TSCEChat

tsce = TSCEChat()
reply = tsce("Explain the Monty Hall problem.")
# reply.ESCP  → hidden embedding space control prompt
# reply.content → escp-steered answer
```

Internal logic:

```
# Phase 1: ESCP
ESCP = call_gpt(
    DEFAULT_ESCP_TEMPLATE,
    prompt   "\n\nGenerate Embedding Space Control Prompt. ",
    temperature=phase1_temp,  # default 1.0
    top_p=phase1_top_p,       # default 0.01
    escp=True
)

# Phase 2: Final answer
final = call_gpt(
    SYSTEM_INSTRUCTIONS_MODEL_ONE   "\n\n"   ESCP,
    prompt,
    temperature=phase2_temp,  # default 0.3
    top_p=phase2_top_p        # default 1.0
)
```

Both steps use Azure or OpenAI endpoints selected at runtime; latency/logprobs are stored in the `TSCEReply` object.

9.3 Single-Pass vs. Two-Phase Benchmark (tsce_test.py)

The script `tsce_test.py` automates comparison over a set of prompts:

Baseline:

```
m1 = call_gpt(
    SYSTEM_INSTRUCTIONS_MODEL_ONE,
    prompt,
    temperature=args.temperature,
    top_p=args.top_p
)
```

TSCE:

```
ESCP, final = two_step_contextual_enrichment(
    prompt,
    phase1_temp=args.phase1_temp,
    phase1_top_p=args.phase1_top_p,
    phase2_temp=args.phase2_temp,
    phase2_top_p=args.phase2_top_p
)
```

Each output variant (baseline, ESCP, TSCE) is analyzed via:

- Token metrics: Shannon entropy, lexical diversity, sentence statistics
- Semantic similarity: TF–IDF embedding, cosine similarity, KL divergence
- Entropy collapse: 2-D t-SNE projection and ConvexHull area
- Aggregation: JSON logs, optional Matplotlib scatter/arrow plots

Parallel execution uses `ThreadPoolExecutor` (workers = CPU count), and all hyperparameters and responses are captured for reproducibility.

9.4 Agentic Task Evaluation (tsce_agent_test.py)

To measure TSCE's impact on iterative reasoning, `tsce_agent_test.py` implements six strategies:

- Baseline
- TSCE
- Chain-of-Thought (CoT)
- CoT TSCE (CT)
- Refine (Self-Refine)
- Refine TSCE (RT)

Key components include:

- Client pool: Round-robin Azure/OpenAI deployments or Ollama instances
- Call helpers: `call_model()` for chat completions; `safe_tsce()` wraps TSCEChat with retry/backoff
- Strategy runners: `run_cot()`, `run_refine()`, `run_refine_tsce()`
- Outputs: per-strategy pass/fail, latency, logprobs, CSV and Markdown for significance testing

9.5 Reproducibility & Extensibility

- All temperatures, `top_p`, repeat counts, and worker pools are exposed via argparse.

- Metadata: Each run logs API versions, deployment IDs, seeds, request/response bodies, and performance stats.
- Modularity: Swap backends by changing env vars or passing a custom client to TSCEChat; templates and evaluators are easily swapped.

This design allows TSCE to be used as a drop-in enrichment layer for any LLM-based workflow without retraining or internal model modifications.

10 Key Benefits of TSCE

By adopting Two-Step Contextual Enrichment (TSCE), practitioners gain:

- **Substantially improved reliability.** TSCE doubles down on task adherence, boosting rule-pass rates (e.g., "no em-dash") from ∼50% to ∼94% on GPT-4.1, and yielding consistent gains across GPT-3.5 and Llama-3.
- **Enhanced coherence and focus.** ESCP-guided refinement contracts semantic spread—reducing convex-hull area by roughly 18% on repeat runs—while preserving topical diversity, yielding more on-topic responses.
- **Maintained creative latitude.** Unlike heavy-handed prompt engineering, the latent ESCP is abstract, allowing nuanced elaboration within a tightened subspace (see Section 2).
- **Model-agnostic integration.** TSCE overlays transparently onto any LLM endpoint (GPT-4o, GPT-3.5, Llama-3, etc.) without retraining or internal modifications (see Section 8).
- **Low overhead, high ROI.** Dual calls add $< 0.5\,\mathrm{s}$ latency and $\approx$ \$0.0006 per request—two orders of magnitude cheaper than majority-vote CoT sampling, yet offering comparable or superior adherence gains (see Section 7).
- **Safety "choke-point."** Because ESCPs are produced before any user-visible text, policy-violating trajectories can be detected and aborted early, minimizing compute waste and risk.
- **Auditability and drift detection.** Compact ESCPs serve as latent "fingerprints" whose token-level diffs surface distributional drifts more rapidly than verbose CoT logs, improving long-term monitoring.
- **Explainable proxy metrics.** Simple 2-D t-SNE hull area or mean pairwise cosine distance provide human-interpretable signals of semantic contraction and health, complementing backend-invariant metrics (see Section 7).
- **Extensible evaluation pipeline.** The modular Python drivers (`tsce_test.py`, `tsce_agent_test.py`) log full JSON, seeds, and performance stats, enabling perfect reproducibility and seamless inclusion of new metrics or model backends (see Section 8).

11 Conclusion

Two-Step Contextual Enrichment (TSCE) steers LLMs toward reliability and style-adherence without retraining. Across three models and more than 4000 tasks, TSCE increased precision by 24–44 percentage points, reduced embedding-space dispersion by 18 %, and lowered lexical variance by 9 %, all for $\approx 2\times$ token cost and <0.5 additional latency. Open-source drivers and logs enable immediate verification and future extension.

12 Responsible Use & Limitations

The TSCE ESCP mechanism offers significant gains in output consistency and focus, but it also introduces risks that extend beyond operational heuristics. Responsible deployment requires a blend of technical controls, governance processes, and continuous oversight.

12.1 Bias & Fairness

- **Uneven performance across groups.** ESCPs derived from unbalanced corpora can produce outputs that systematically favor certain dialects, styles, or demographic groups. Mitigation: stratified evaluations, data augmentation, and fairness-aware ESCP templates.
- **Calibration drift.** Model updates or ESCP-template tweaks may shift output distributions, eroding prior fairness guarantees. Mitigation: continuous validation against held-out fairness benchmarks and ESCP-version tracking.

12.2 Misuse & Dual Use

- **Sophisticated disinformation.** TSCE's coherence gains can be weaponized to generate persuasive false narratives at scale. Mitigation: digital

watermarking, response provenance metadata, and rate-limiting by domain.

- **Targeted exploitation.** Personalized ESCPs may enable tailored phishing or social-engineering campaigns. Mitigation: credentialed access, anomaly detection, and explicit user consent for sensitive content.
- **Intellectual property leakage.** ESCP sequences might inadvertently recall proprietary or copyrighted material. Mitigation: memorization tests, output filtering, and legal review workflows.

12.3 Governance & Oversight & Security

- **Role-based access control.** Limit ESCP-generation rights to authorized teams under clear usage policies.
- **Human-in-the-loop review.** Require manual sign-off for outputs in high-stakes domains (e.g., medical, legal, financial).
- **Transparency and audit logs.** Record ESCP seeds, template versions, and key hyperparameters to enable post-hoc analysis and incident response.
- **Data minimization.** Exclude sensitive personal data from ESCP derivation procedures.
- **Infrastructure safeguards.** Enforce encryption at rest and in transit and apply the principle of least privilege across TSCE components.

12.4 Future Directions

Develop standardized benchmarks for misuse resilience, integrate interpretable concept-to-token mappings for ESCP transparency, and explore formal verification techniques to certify ESCP robustness.

† significantly different from baseline at $p < 0.05$.

References

[1] Mojtaba Borazjanizadeh and Steven T. Piantadosi. Neurosymbolic code anchoring for improved mathematical reasoning. In *Proceedings of the 2024 Annual Meeting of the Association for Computational Linguistics*, 2024.

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems 33*, 2020.

[3] Anastasiya Cherepanova and James Zou. Steering language models with gibberish prompts. *arXiv preprint*, 2024.

[4] Xinyao Hao, Yi Qiu, Yao Fu, et al. Coconut: Training large language models to reason in a continuous latent space. In *International Conference on Learning Representations (ICLR)*, 2024.

[5] Abhishek Jha and Shashank Srivastava. Unsupervised vec2vec translation unifies embedding spaces. *arXiv preprint*, 2025.

[6] Chengxing Jia, Ziniu Li, Pengyuan Wang, Yi-Chen Li, Zhenyu Hou, Yuxiao Dong, and Yang Yu. Controlling large language model with latent actions. *arXiv preprint*, 2025. v1, 27 Mar 2025.

[7] Tom Kojima and colleagues. Large language models are not fair evaluators. In *Advances in Neural Information Processing Systems*, 2023.

[8] Aman Madaan, Kartikeya Badola, Yiming Zhang, et al. Self-refine: Iterative refinement with self-feedback. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, 2023.

[9] OpenAI. Gpt-4 system card. `https://cdn.openai.com/papers/GPT4_System_Card.pdf`, 2023.

[10] Long Ouyang et al. Prompt engineering with structured style tags. *arXiv preprint*, 2023.

[11] Stefan Stoye, Alexander I. Stoica, and Helmuth Plattner. Likelihood-free inference with entropy reduction. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[12] Xuezhi Wang, Jason Wei, Dale Schuurmans, et al. Self-consistency improves chain-of-thought reasoning in language models. 2022.

[13] Jason Wei, Xuezhi Wang, Dale Schuurmans, et al. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 36*, 2022.

[14] Shunyu Yao, Dian Yu, Jeffrey Zhao, et al. React: Synergizing reasoning and acting in language models. In *Advances in Neural Information Processing Systems 37*, 2023.

[15] Shengya Zhou, Haotian Ma, Eric Wong, et al. Planning tokens: Controlling language model generation with extensible plans. In *International Conference on Learning Representations*, 2023.

[16] Jing Zhu, Yi Qiu, et al. Training large language models to reason in a continuous latent space. *arXiv preprint*, 2024.