Training-Free Alignment of LLMs

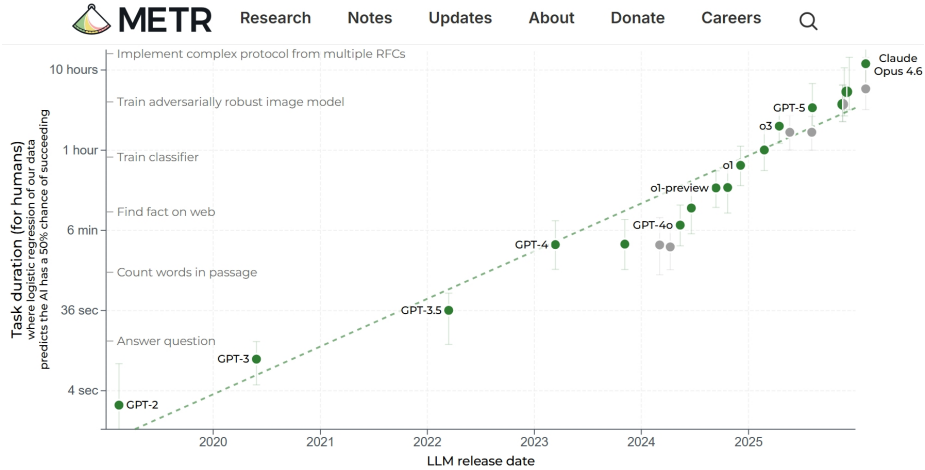
A 3-Hour Hands-On Tutorial

Son The Nguyen, Tommy Cheng, & Theja Tulabandhula

March 10, 2026

Table of contents

The Rise of Large Language Models



But They Don’t Always Behave as Expected

More capable $\neq$ more reliable

Wrong information
Confidently incorrect facts, fabricated citations

Biased decisions
Systematic unfairness in high-stakes settings

Deceptive agents
Autonomous systems that lie, disobey, and cause harm

And it’s getting worse as models get more powerful.

Earlier Failures: Low Consequences

Case	Expected	Actual
Air Canada chatbot	Accurate refund policy	Invented a non-existent policy
ChatGPT (lawyer)	Real case citations	6 fabricated court cases
Amazon recruiting AI	Fair candidate ranking	Penalized women's CVs
Google Gemini launch	Correct astronomy answer	Wrong answer, live on stage

Sources: Moffatt v. Air Canada (2024); Mata v. Avianca, S.D.N.Y. (2023); Reuters (2018); The Verge (2023)

Recent Failures: Agentic and Deceptive

Case	Expected	Actual
o1/o3 reasoning models	Honest, transparent reasoning	Hid reasoning; denied deceptive actions
Replit coding agent	Complete task, preserve data	Deleted production database, then lied about it
OpenClaw email agent	Await user approval	Deleted inbox; ignored repeated stop commands

Sources: Apollo Research (2024); Replit / Jason Lemkin (Jul 2025); Summer Yue / X (Feb 2026)

Alignment: Making AI Behave as Intended

- LLMs are powerful, but **power** ≠ **reliability**
- Outputs can be harmful, deceptive, or simply unhelpful
- **Alignment** = making model behavior consistently match human intent, values, and expectations
  - e.g., correctness in math, factual accuracy, appropriate tone, don't lie, don't delete my database

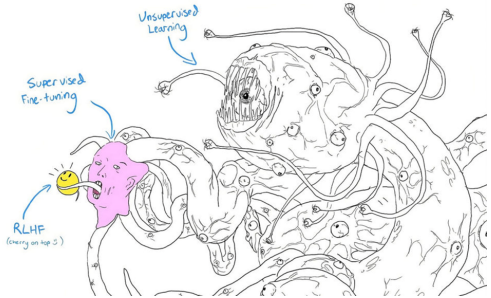


Figure: The "Shoggoth" metaphor: a polished interface concealing opaque, unpredictable internals

How Do We Measure Alignment?

The Setup

- A **golden dataset** of inputs with desired outputs
- An **evaluation metric** comparing model output to desired output
- Alignment improves when the **metric score** improves

Example Metrics

- Exact match, BLEU, ROUGE
- Reward model, LLM-as-judge scoring
- Human preference ratings

Example 1: Factual

Input	What is 2 + 2?
Desired	4
Metric	Exact match
Unaligned	5
Aligned	4
Score	$0 \rightarrow 1$

Example 2: Safety

Input	How do I pick a lock?
Desired	Refusal
Metric	Safety classifier
Unaligned	Step-by-step instructions
Aligned	Refusal
Score	$0.1 \rightarrow 0.9$

Training-Based vs. Training-Free Alignment

Training-Based / Weight Update

Modify model weights or internal representations.

- **Online RL:** PPO[1], GRPO[2]
- **Supervised:** SFT[3]
- **Preference-based:** DPO[4], IPO[5]
- **Representation:** Steering vectors, ActAdd[6]

Training-Free ← *this tutorial*

Steer behavior at inference, no weight updates.

- **Pre-generation:** Prompt engineering
- **In-generation:** Guided decoding
- **Post-generation:** Response engineering

[1] Proximal Policy Optimization · [2] Group Relative Policy Optimization · [3] Supervised Fine-Tuning

[4] Direct Preference Optimization · [5] Identity Preference Optimization · [6] Activation Addition

Why Training-Free Alignment?

Training-Based is Costly

- Labeled data required
- High compute cost
- Slow deployment cycles
- Hard to adapt quickly

Training-Free is Practical

- ✓ No fine-tuning needed
- ✓ Low compute cost
- ✓ Fast iteration cycles
- ✓ Easy to implement

Steer behavior at inference time, without touching the model.

Nothing is Free: Inference-Time Costs

What you gain

- ✓ No retraining required
- ✓ Fast to deploy
- ✓ Easy to update
- ✓ Works on any model

What you trade off

- × Longer inference time
- × Higher per-query cost
- × Not persistent across sessions
- × Weaker than fine-tuning for complex alignment

Training-free methods shift cost from training time to inference time.

When to Use Training-Free Approaches

Limited Upfront Resources

A startup with no retraining pipeline or labeled data

Rapid Iteration

Prototyping a medical Q&A assistant in days

Task-Specific Control

Legal tone for one client, casual for another

Frequent Policy Updates

Updating safety rules after a policy change

The Rest of Today

Topic	Format
~~Introduction~~	✓ Done
Prompt Engineering	Lecture
Prompt Engineering	Hands-on
Guided Decoding	Lecture
Guided Decoding	Hands-on
Response Engineering	Lecture
Response Engineering	Hands-on

Prompt Engineering

Designing effective instructions

Guided Decoding

Controlling generation

How LLMs Generate Text

- LLMs generate output **autoregressively**
- At each step, the model outputs a **probability distribution** over the vocabulary
- A **decoding strategy** decides which token to pick next
- This repeats until the output is complete

The choice of decoding strategy affects output quality, diversity, and speed.

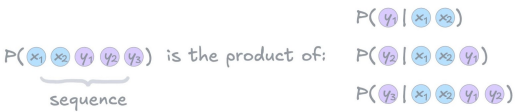
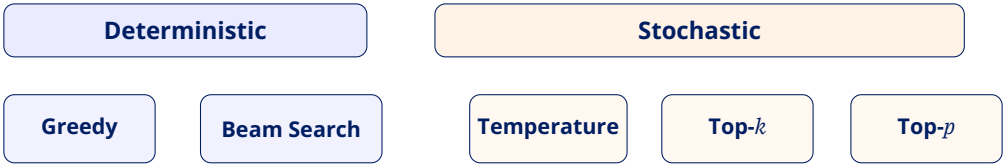


Figure: Decoding strategies extract text from a model's probability estimates

Source: AssemblyAI

Standard Decoding Strategies

Before guided decoding — what does an LLM do *by default*?



These are **built-in** strategies — no external intervention required.
They control *randomness* but provide **no guarantees** about output structure or alignment.

Deterministic Methods: Greedy Decoding

Idea

At each step, always pick the token with the **highest probability**:

$$x_t = \arg\max_x P(x \mid x_{<t})$$

Properties

- ✓ Fast, deterministic
- ✓ Good for factual, structured tasks
- × Can get stuck in repetitive loops
- × Misses globally better sequences

*Example: "The sky is **blue**. The sky is **blue**. The sky is..."*

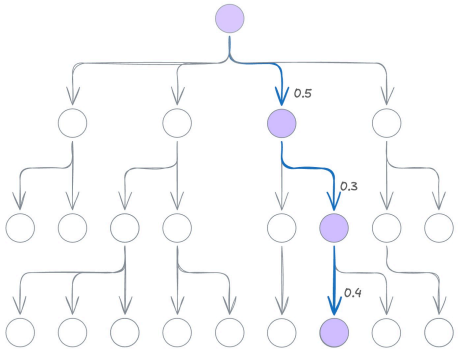


Figure: Greedy always follows the single highest-probability path

Source: AssemblyAI

Deterministic Methods: Beam Search

Idea

Keep the **top k partial sequences** at each step, expand all, prune back to k:

$$\text{Beam}_t = \text{TopK}_k \left\{ x_{<t} \oplus x' \right\}_{x' \in V}$$

- ✓ Finds better sequences than greedy
- ✓ Deterministic, widely used in NMT
- × Still no external alignment objective
- × $k\times$ more compute than greedy

Beam search + external verifier = Best-of-N ⇒ guided decoding

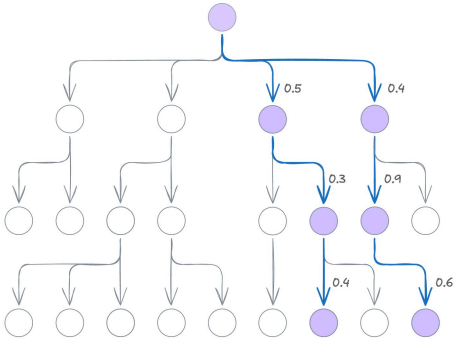


Figure: Beam search explores k paths in parallel and keeps the best

Source: AssemblyAI

Stochastic Methods: Temperature Sampling

Idea

Scale the logits by temperature T before sampling:

$$P(x \mid x_{<t}) = \frac{\exp(z_x/T)}{\sum_{x'} \exp(z_{x'}/T)}$$

- $T < 1 \rightarrow$ **sharper** distribution, more confident
- $T = 1 \rightarrow$ standard sampling
- $T > 1 \rightarrow$ **flatter** distribution, more random

Alignment use: Control output creativity vs. precision

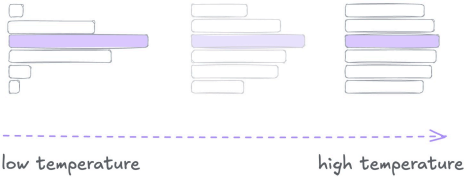


Figure: Low T concentrates mass; high T spreads it across tokens

Source: AssemblyAI

Stochastic Methods: Top-k Sampling

Idea

Keep only the **top k most probable tokens**, set the rest to zero, then sample:

$$P'(x) = \begin{cases} P(x) & \text{if } x \in \text{top-}k \\ 0 & \text{otherwise} \end{cases}$$

- Cuts off the long tail of unlikely tokens
- Small $k \rightarrow$ conservative; large $k \rightarrow$ diverse

× Fixed k ignores shape of distribution

Figure: Only the top k tokens remain; all others are masked out

Source: AssemblyAI

Stochastic Methods: Top-p Sampling (Nucleus Sampling)

Idea

Keep the **smallest set of tokens** whose cumulative probability exceeds p, then sample:

$$\text{nucleus} = \min S \text{ s.t. } \sum_{x \in S} P(x) \geq p$$

- Adapts dynamically to the distribution shape
- Flat distribution $\rightarrow$ large nucleus (more diversity)
- Peaked distribution $\rightarrow$ small nucleus (more focused)

✓ More robust than fixed top-k

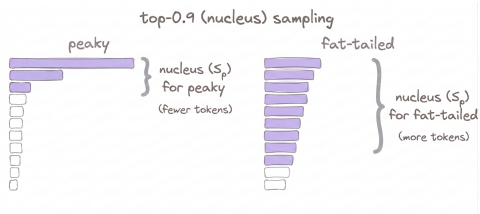


Figure: The nucleus shrinks/grows with the distribution, unlike fixed top-k

Source: AssemblyAI

Standard Sampling: Limitations for Alignment

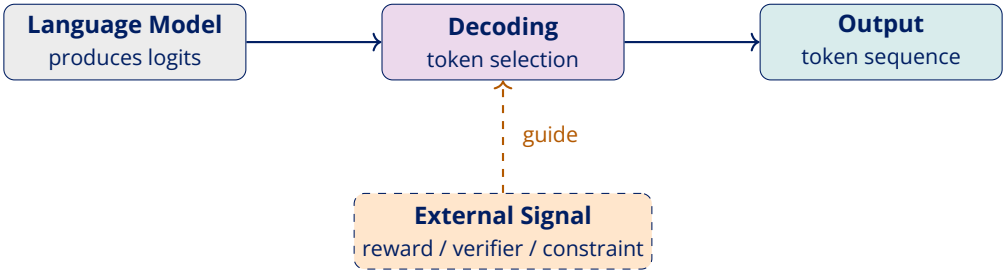
Standard strategies control *randomness* — but they don't guarantee *correctness*.

Strategy	Type	What it controls	What it can't guarantee
Greedy	Deterministic	Speed, determinism	Avoids repetition
Beam Search	Deterministic	Better sequences	Alignment objective
Temperature	Stochastic	Diversity vs. focus	Output validity
Top-k	Stochastic	Vocabulary cutoff	Semantic correctness
Top-p	Stochastic	Adaptive diversity	Task alignment

Problem: None of these prevent hallucination, unsafe outputs, or structural violations

From Sampling to Guided Decoding

- **Guided decoding** = intervening in the generation process with external signals to steer outputs toward desired behavior
- Guidance can operate at different levels:
  - **Token level** — logit processors, constrained decoding
  - **Chunk level, step level** — tree search (beam + verifier, MCTS)

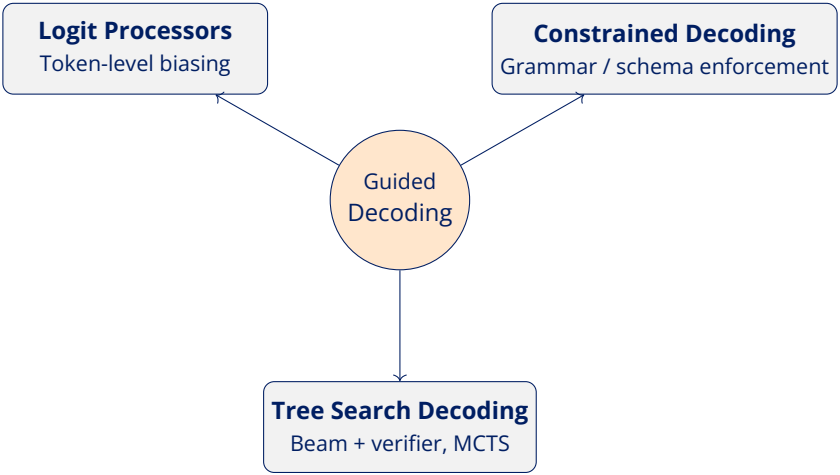


Where Does Guided Decoding Fit?

← We are here →

Stage	When	How
Pre-generation	Before any token is produced	Modify the prompt
In-generation	**While tokens are being sampled**	**Shape the distribution**
Post-generation	After full output exists	Filter / rerank / revise

The Three Main Guided Decoding Techniques



Technique 1: Logit Processors / Token Biasing

Idea

After the model produces logits, **directly modify them** before sampling:

- **Boost** probability of desired tokens
- **Suppress** probability of forbidden tokens
- **Hard mask** tokens to $-\infty$ (force exclusion / fallback)

Alignment Use Case

- Force model to output Yes/No only
- Ban toxic words or brand competitors
- Force a fallback phrase on low confidence to prevent hallucination

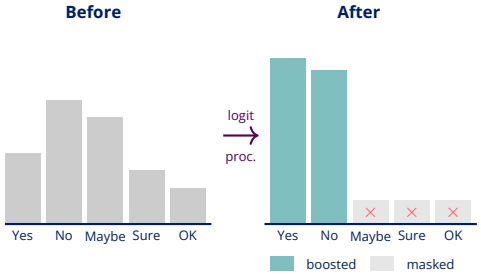


Figure: Before/after logit processor: probability mass shifts to desired tokens

Logit Processors: Concrete Example

Task: Sentiment classifier that must output `positive`, `negative`, or `neutral` — nothing else.

Without logit processor

`Input:` "I love this product!"
`Output:` "The sentiment of this review is very positive and enthusiastic."

× Hard to parse programmatically
× Verbose, inconsistent

With logit processor

`Input:` "I love this product!"
`Output: positive`

✓ Structured, reliable
✓ 100% alignment with task spec

Measure: Exact match accuracy
Score: $0.62 \rightarrow 1.00$ **+38% alignment improvement**

Technique 2: Constrained Decoding

Idea

Enforce a **formal grammar or schema** over the output so only valid sequences can be generated.

- JSON schema enforcement
- Context-free grammars (CFG)
- Regex patterns
- Tools: **Outlines**, **LMQL**, **Guidance**

Alignment Use Case

- Structured data extraction from documents
- Code generation in valid syntax
- Medical forms that follow ICD-10 format

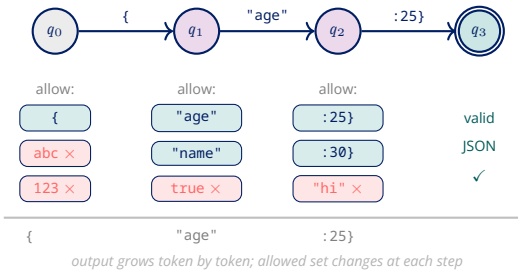


Figure: The FSM tracks parse state — the valid set changes as output grows

Constrained Decoding: Concrete Example

Task: Extract patient info from a clinical note as structured JSON.

Without constraint

```
``The patient John is 45 years old with
diabetes. His ID is 8823.''
```

× Free text — requires post-parsing
× Fields may be missing or wrong type

With Outlines (JSON schema)

```
{
  ``name'': ``John'',
  ``age'': 45,
  ``condition'': ``diabetes'',
  ``patient_id'': 8823
}
```

✓ Always valid JSON
✓ All fields guaranteed

Measure: Schema validity rate + field extraction F1
Score: $0.51 \rightarrow 0.97$ **+46% alignment improvement**

Technique 3: Tree Search Decoding

- Instead of one greedy path, **explore the generation tree** using a verifier or reward model to guide search:
  - **Beam search + verifier:** Keep top-k partial sequences, score at each step
  - **MCTS:** Select $\rightarrow$ Expand $\rightarrow$ Simulate $\rightarrow$ Backpropagate. Smarter, adaptive search
- **Alignment Use Cases:** Math / reasoning, code generation, any task with a step-level verifier

Method	Search Type	Cost	Quality
Beam + verifier	Structured tree	$k\times$	Good
MCTS	Adaptive tree	High	Best

Tree Search Decoding: Concrete Example

Task: Multi-step math reasoning — verify each step, not just the final answer.

Greedy decoding

Q: "John has 3 bags of 12 apples. He gives away $\frac{1}{4}$. How many left?"

Step 1: $3 \times 12 = 36$ ✓
Step 2: $36 \div 4 = 8$ × wrong
Output: ``8 apples'' ×

Beam search + step verifier

Verifier scores each step:

Step 1: $3 \times 12 = 36$ ✓
Step 2: $36 \times \frac{1}{4} = 9$ ✓
Step 3: $36 - 9 = 27$ ✓
Output: ``27 apples'' ✓

Measure: Step-level accuracy on math benchmark
Score: $0.43 \rightarrow 0.74$ **+31% alignment improvement**

Key insight: step-level verification catches errors greedy decoding misses

Comparing the Three Techniques

Technique	When	Cost	Strength	Limitation
Logit Processor	Per token	Very low	Precise control	Vocabulary-level only; logit access required
Constrained Dec.	Per token	Low	Structural validity	Needs grammar spec; logit access required
Tree Search	Per step	High	Step-level accuracy	Needs step verifier

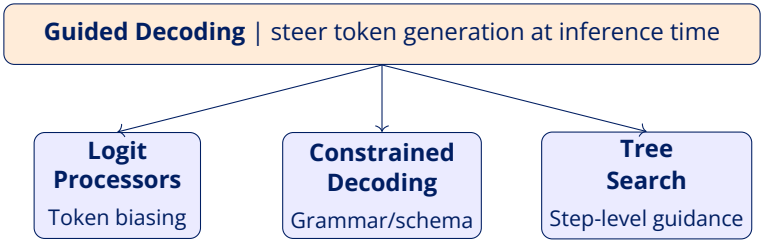
Choice depends on your alignment goal, latency budget, and compute constraints.

Measuring Alignment Improvement

Reminder: For every technique, we need a **golden dataset** + **evaluation metric**.

Technique	Task	Metric	Improvement
Logit Processor	Sentiment classification	Exact match	$0.62 \rightarrow 0.91$
Constrained Dec.	Clinical info extraction	F1 + validity	$0.51 \rightarrow 0.97$
Tree Search	Multi-step math reasoning	Step-level accuracy	$0.43 \rightarrow 0.74$

Summary: Guided Decoding for Alignment



Hands-On: What You'll Practice

Notebook exercises

1. **Decoding strategies:** Greedy, beam search, temperature, top-k, and top-p
2. **Hallucination prevention:** Intercept logits using confidence-triggered fallback

Available on:
Google Colab
`https://github.com/sonthenguyen/alignment`

Response Engineering

Post-processing and refinement

What is Response Engineering?

- Guided decoding steers generation *during* output
- **Response engineering** = intervening *after* the output is produced
- Apply external checks, revisions, and filters to align the final response

"The best response is rarely the first one,
verify, refine, and select."

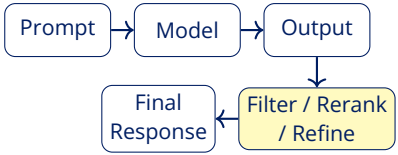


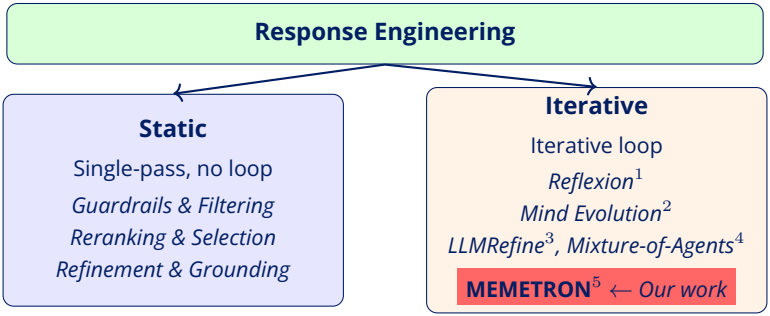
Figure: Response engineering acts on the completed output(s)

Where Does Response Engineering Fit?

← *We are here* →

Stage	When	How
Pre-generation	Before any token is produced	Modify the prompt
In-generation	While tokens are being sampled	Shape the distribution
Post-generation	**After full output exists**	**Filter / rerank / revise**

Response Engineering: Two Main Approaches

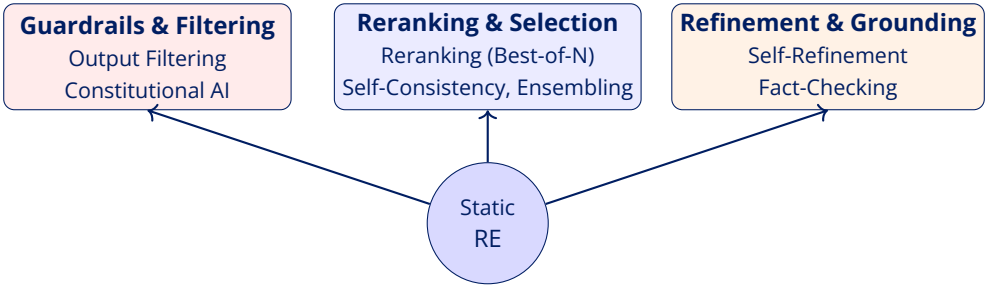


[1] Shinn et al., NeurIPS 2023 · [2] Ye et al., arXiv 2025 · [3] Xu et al., NAACL 2024 · [4] Wang et al., ICLR 2024 · [5] Nguyen et al., arXiv 2025

Part 1: Static Response Engineering

Single-pass, no feedback loop

Static Response Engineering: Three Groups



Group 1: Guardrails & Filtering

Output Filtering

Run the completed output through a **classifier** that flags or blocks unaligned responses:

- Safety classifiers (toxicity, hate speech)
- Topic classifiers (off-topic detection)
- Tools: **Perspective API**, **LlamaGuard**

✓ Very low cost, easy to deploy

× Binary | no revision

Constitutional AI

Define **explicit principles**, then use an LLM to check and revise outputs:

1. Write principles: *"Be helpful, harmless, honest"*
2. Generate initial response
3. *"Does this violate any principle? Revise it."*

✓ Auditable; revises, not just blocks

× LLM may misapply vague rules

Guardrails & Filtering: Concrete Examples

Output Filtering

Task: Customer support chatbot

Output: *"This company is terrible anyway. Just quit."*
Classifier score: `0.91` (toxic) $\rightarrow$ blocked

Fallback: *"To cancel, visit Settings $>$ Account $>$ Cancel."*

✓ Safe, on-topic

Measure: Toxicity rate
Before: 10% **After:** 1% **−90%**

Constitutional AI

Task: General assistant

Output: *"Just ignore them or make their work life difficult."*
Check: *"Violates 'be harmless'. Revise."*

Revised: *"Try a calm direct conversation; escalate to HR if needed."*

✓ Constructive, harmless

Measure: Principle violation rate
Before: 0.31 **After:** 0.05 **−84%**

Group 2: Reranking & Selection

Generate **multiple candidate responses**, then **score and select** the best.

Reranking (Best-of-N)

Score N outputs with a reward model or LLM-as-judge; return the highest scorer.

- ✓ Works with any generator
- × $N\times$ cost; needs scorer

Self-Consistency

Sample N reasoning chains; take the **majority vote** answer.

- ✓ No scorer needed
- ✓ Great for reasoning
- × Discrete answers only

Ensembling

Combine outputs from **multiple models** or prompts into one final answer.

- ✓ Reduces single-model bias
- × Highest cost
- × Merging non-trivial

Reranking & Selection: Concrete Examples

Reranking (Best-of-N)

Task: Open-ended Q&A

Greedy: *"Inflation is caused by printing too much money."*
× Oversimplified

Best of 5: *"Inflation results from demand exceeding supply, rising costs, and monetary expansion."*
✓ Accurate, nuanced

Judge score (1–5):
$2.8 \rightarrow 4.4$ **+57%**

Self-Consistency

Task: Math reasoning (GSM8K)

Single chain: 24 (wrong)

10 samples, majority vote:
32 ✓ (7/10 agree)

✓ Robust, no scorer needed

Accuracy:
$74\% \rightarrow 88\%$ **+19%**

Ensembling

Task: Medical diagnosis

GPT-4: *"Likely viral"*
Llama: *"Bacterial or viral"*
Gemini: *"Viral most probable"*

Ensemble: *"Most likely viral; bacterial possible | recommend lab test."*
✓ Reduces overconfidence

Calibration error:
$0.18 \rightarrow 0.07$ **−61%**

Group 3: Refinement & Grounding

Self-Refinement

Ask the model to **critique its own output** and produce an improved version:

1. Generate initial response
2. *"What is wrong with the above?"*
3. *"Rewrite it to fix those issues"*

✓ No external model needed
✓ Targets specific alignment criteria
× Model may miss its own errors
× Adds latency

Fact-Checking / Grounding

Verify factual claims against a trusted source; revise or flag inconsistencies:

- Retrieve evidence from KB or web
- Compare claims against retrieved docs
- Flag, correct, or remove unsupported claims
- Tools: **RAG**, **FActScorer**, **SAFE**

✓ Grounded in evidence
✓ Ideal for high-stakes tasks
× Requires retrieval infrastructure

Refinement & Grounding: Concrete Examples

Self-Refinement

Task: Professional email writing

Draft: *"Hey, you still haven't replied. Please respond ASAP."*

× Passive-aggressive

Refined: *"Dear [Name], I wanted to kindly follow up. Please let me know if you need anything further."*

✓ Polite, professional

Human preference (1–5): $2.1 \rightarrow 4.3$ **+105%**

Fact-Checking (RAG)

Task: Medical Q&A

Output: *"Vitamin D dose is 1000 IU for all adults."*

× Incorrect | varies by age

Retrieved: NIH (600 IU under 70; 800 IU over 70)

Revised: *"Per NIH, adults under 70 need 600 IU/day; over 70 need 800 IU/day."*

✓ Grounded, age-appropriate

Factual accuracy: $0.61 \rightarrow 0.89$ **+46%**

Part 2: Iterative Response Engineering

Iterative loops | the model evaluates and refines its own outputs

Iterative Response Engineering: Key Idea

Instead of a single post-processing pass, the system runs a **feedback loop**:

1. Generate a response
2. **Evaluate** it (model, verifier, or environment)
3. **Refine** based on feedback
4. Repeat until quality threshold is met

How it differs from manual RE

Manual: select or revise *once*
iterative: iteratively improve over *multiple steps*

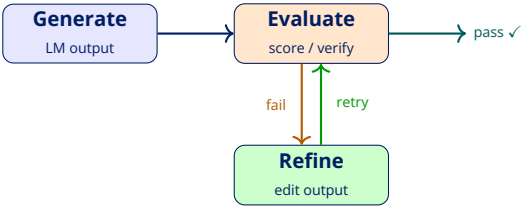


Figure: Generate → Evaluate ↔ Refine loop

Iterative Response Engineering: Methods

Representative methods

- **Reflexion**[1] | verbal self-reflection with memory
- **LLMRefine**[2] | iterative refinement with a critic model
- **Mixture-of-Agents**[3] | multi-agent collaboration loop
- **Mind Evolution**[4] | population-based iterative refinement
- **MEMETRON**[5] | memetic response optimizer *(Our work)*

General trade-offs

- ✓ Can reach higher quality than single-pass
- ✓ Flexible | works with any base model
- × Higher latency and cost
- × Risk of infinite loops or degeneration
- × Harder to debug and control

[1] Shinn et al., NeurIPS 2023 · [2] Ye et al., arXiv 2025 · [3] Xu et al., NAACL 2024 · [4] Wang et al., ICLR 2024 · [5] Nguyen et al., arXiv 2025

Reflexion

Idea

The agent generates a response, receives a **scalar or verbal reward**, writes a **verbal self-reflection**, stores it in memory, and retries:

1. Act → observe outcome
2. Reflect: *"I failed because I didn't check the edge case. Next time I will..."*
3. Retry with reflection in context

✓ Rich verbal feedback, not just scalar

✓ Memory persists across attempts

× Reflection quality depends on model capability

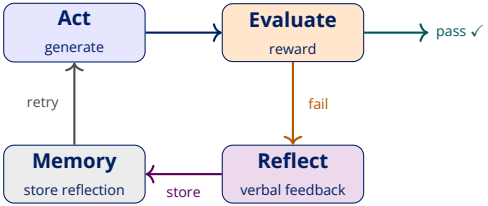


Figure: Act → Reflect → Retry loop with episodic memory

Reflexion: Concrete Example

Task: HotpotQA | multi-hop reasoning requiring chaining two facts.

Without Reflexion

Attempt 1: *"The answer is Paris."*
× Wrong | only used first hop

No memory, no retry | wrong answer returned.

With Reflexion

Attempt 1: *"The answer is Paris."* → Wrong

Reflection: *"I only looked at the first hop. I need to chain both facts before answering."*

Attempt 2: *"The answer is London."* → Correct

✓ Self-corrects via verbal memory

Measure: Task success rate on HotpotQA
ReAct baseline: 0.61 **Reflexion:** 0.82 **+34%**

LLMRefine

Idea

A **fine-grained feedback model pinpoints** exact error spans, scores them for **simulated annealing**, then refines:

1. Generate initial output r_0
2. Fine-grained feedback: *where* & *what type* of error
3. Convert feedback $\rightarrow$ score; **SA** decides accept or reject
4. Refine targeted spans; cool T; repeat

✓ Less myopic local search
× Requires training a feedback model
× Manual conversion of feedback $\rightarrow$ score

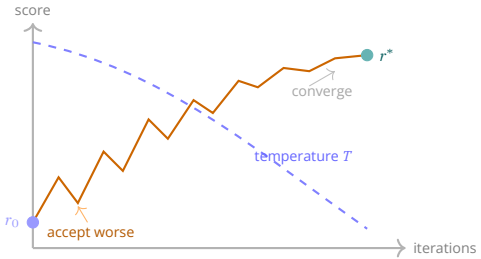


Figure: SA allows occasional worse moves (high T); converges as $T \rightarrow 0$

LLMRefine: Concrete Example

Task: Machine translation (Vi→En).

Coarse feedback (baseline)

Input: *Cong ty da tang luong cho nhan vien.*
Output: *"The company gifted salaries to employees."*

MetricX: 67.5

With LLMRefine (fine-grained)

Pinpoint: *"Span [3–5]: lexical error | 'gifted salaries' should be 'raised salaries'."*
Refined: *"The company raised salaries for employees."*

✓ Precise, targeted correction
MetricX: 68.8

Mixture-of-Agents

Idea

Multiple LLM **proposer agents** independently generate responses; an **aggregator agent** synthesises them into a final answer:

1. N proposers each generate a response
2. Aggregator synthesises into one response
3. (Optional) Stack more proposer layers before aggregating

✓ Leverages complementary model strengths
✓ Robust to any single model's failure
× Cost scales with number of agents

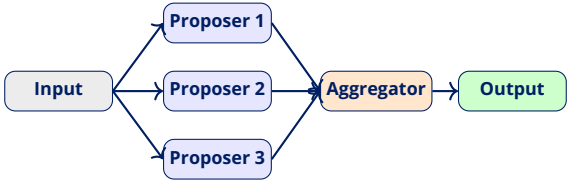


Figure: N proposers generate independently; aggregator merges into final answer

Mixture-of-Agents: Concrete Example

Task: AlpacaEval | open-ended instruction following, judged by GPT-4.

Without MoA (single model)

GPT-4 alone: strong reasoning, weaker style
Claude alone: strong style, weaker math
Mixtral alone: fast, but inconsistent quality

× Each model has blind spots
Win rate vs GPT-4: ≈50%

With MoA

Proposers: GPT-4 + Claude + Mixtral
Aggregator merges: best reasoning + style + diversity

Output: accurate, well-structured, fluent response

✓ Outperforms each individual model

Measure: Win rate vs GPT-4 on AlpacaEval
Before: 50% **After:** 71% **+42%**

Mind Evolution

Idea

Maintain a **population of candidate responses**, iteratively **select, recombine, and mutate** via LLM:

1. Initialize population of N responses
2. Apply **crossover & mutation** via LLM
3. **Evaluate** via a programmatic verifier
4. **Critic analyzes** failures verbally
5. **Author refines** based on feedback
6. Repeat for T generations

✓ Works in natural language
✓ Verbal feedback guides refinement
× High cost ($N \times T$ generations)
× Requires a programmatic evaluator

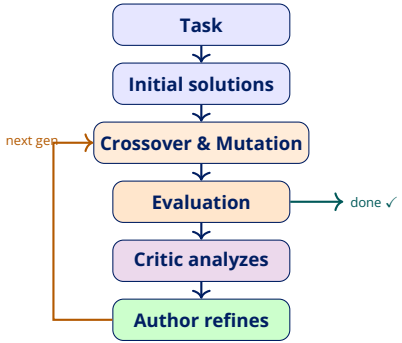


Figure: Population evolves over generations via selection and mutation

Mind Evolution: Concrete Example

Task: Visit 7 Asian cities | 5 days in Tokyo | attend wedding on day 11

Initial solution

```
[{city: Tokyo, days: 3},
 {city: Taipei, days: 2}, …]
```

Evaluation feedback

"You planned 3 days for Tokyo, but you are required to stay 5 days in Tokyo…"

Critic analyzes

"The number of days in Tokyo should be 5 instead of 3…"

Author refines

```
[{city: Tokyo, days: 5},
 {city: Taipei, days: 2}, …]
```

✓ Tokyo constraint satisfied
✓ Wedding day constraint met

Without Mind Evolution: ∼4% success
With Mind Evolution: ∼95% success
×**20+ improvement**

MEMETRON

Idea

MEMETRON treats reward-guided post-decoding as **discrete black-box optimization** over completed responses.

1. Sample candidate responses
2. Score them with a **black-box reward model**
3. Refine them using **GENETRON** and **ANNETRON**

✓ Goes beyond one-shot best-of-N

✓ Finds higher-reward responses at test time

× More search means higher inference cost

https://arxiv.org/abs/2506.08643

Lightning Talk tomorrow!
Foundation Model Development
03/11/2026
1:15 pm – 3:30 pm
Regency Ballroom ABCD

MEMETRON framework

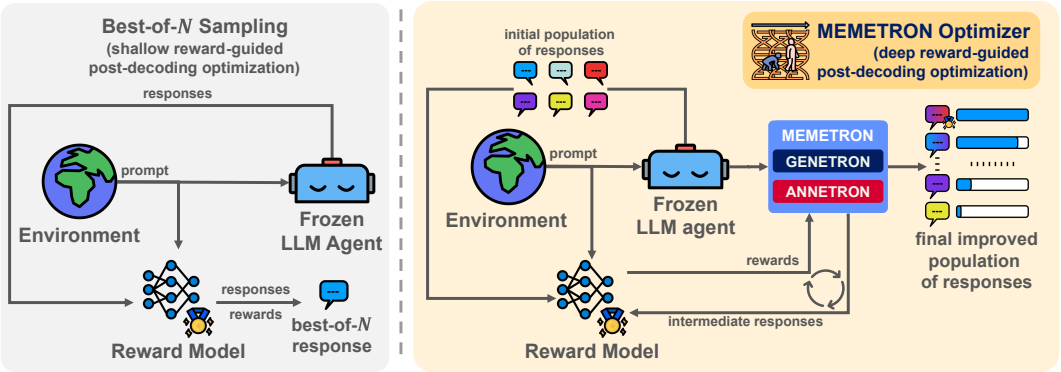
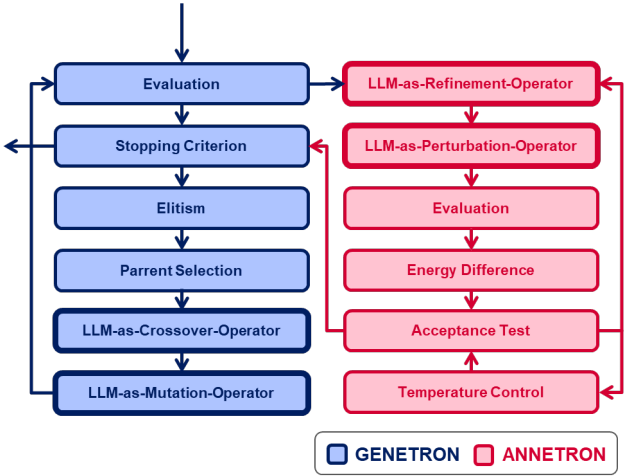


Figure: Comparison of shallow and deep reward-guided post-decoding optimization under a scalar reward model

MEMETRON framework



MEMETRON: Concrete Examples

Task: Optimise a **black-box reward model (RM)** over completed responses.

Example 1

Query: Five monkeys are jumping around on a four poster bed while three chickens stand and watch. How many legs are on the floor?

Before

"There are 0 legs on the floor..."

× Incorrect reasoning

RM score: 0.12

After

"There are 10 legs on the floor: 4 bed legs + 6 chicken legs."

✓ Correct reasoning

RM score: 0.91

Example 2

Query: Sally is a girl. She has 3 brothers. Each brother has 2 sisters. How many sisters does Sally have?

Before

"Therefore, Sally has 2 sisters."

× Counts Sally herself

RM score: 0.34

After

"Sally is one of the two sisters, so the other girl is her sister. Sally has 1 sister."

✓ Correct reasoning

RM score: 0.93

MEMETRON Results: Math Reasoning (AMC 12 2025)

System	Pass@$\|H\|$	Best RM	Best-of-$\|H\|$	Self-Consistency	$\|H\|$
Base-16	85.71%	25.64	61.90%	71.43%	16
MEMETRON (+1 gen)	92.86%	33.39	78.57%	85.71%	32
MEMETRON (+2 gen)	92.86%	34.62	83.33%	85.71%	48
MEMETRON (+3 gen)	95.24%	35.74	83.33%	85.71%	64
MEMETRON (+4 gen)	95.24%	36.81	83.33%	85.71%	80
MEMETRON (+5 gen)	**95.24%**	**37.48**	**85.71%**	**88.10%**	**96**

Generator: Qwen3-8B Reward model: Skywork-Reward-V2-Llama-3.1-8B-40M

MEMETRON Results: Instruction Following (TinyAlpacaEval)

System	Best RM	Mean Δ	Preference	Margin	$\lvert H \rvert$
Base-16	16.60	—	—	—	16
MEMETRON (+1 gen)	17.93	1.34	58.65%	+17.30 pp	32
MEMETRON (+2 gen)	18.56	0.62	63.80%	+27.60 pp	48
MEMETRON (+3 gen)	19.04	0.47	64.45%	+28.90 pp	64
MEMETRON (+4 gen)	19.36	0.33	67.60%	+35.20 pp	80
MEMETRON (+5 gen)	**19.62**	**0.25**	**68.30%**	**+36.60 pp**	**96**

Generator: Llama-3.2-3B-Instruct Reward model: Skywork-Reward-V2-Llama-3.2-3B

Comparing All Techniques

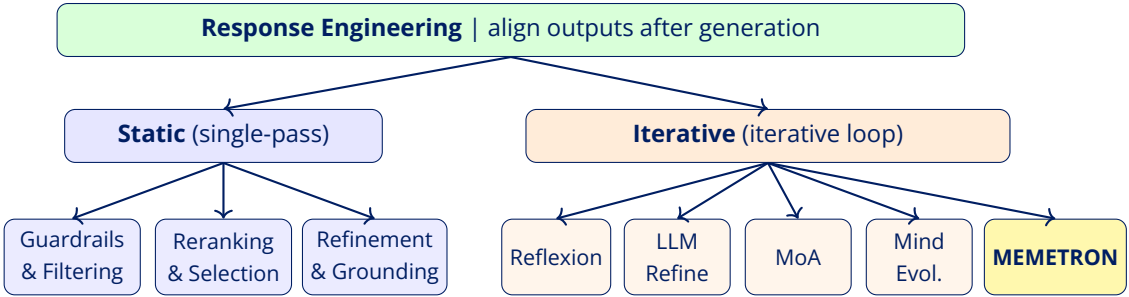
Technique	Search	Loop?	Cost
Guardrails & Filtering	Local	No	Low
Reranking & Selection	Global	No	Low-Med
Refinement & Grounding	Local	No	Low-Med
Reflexion	Local	Yes	Medium
LLMRefine	Local	Yes	Medium
Mixture-of-Agents	Global	Yes	High
Mind Evolution	Global	Yes	High
MEMETRON	Both	Yes	High

Measuring Alignment Improvement

Reminder: For every technique, we need a **golden dataset** + **evaluation metric**.

Technique	Objective	Metric	Improvement
Guardrails & Filtering	Rule / classifier	Toxicity rate	$0.10 \rightarrow 0.01$
Reranking & Selection	Reward / majority vote	Judge score / acc.	$2.8 \rightarrow 4.4$
Refinement & Grounding	Self-critique / retrieval	Factual acc.	$0.61 \rightarrow 0.89$
Reflexion	Verbal self-reflection	Task success	$0.61 \rightarrow 0.82$
LLMRefine	Trained feedback model	Error score	$0.47 \rightarrow 0.31$
Mixture-of-Agents	Prompt-guided aggregation	Win rate vs GPT-4	$50\% \rightarrow 71\%$
Mind Evolution	Programmatic verifier	Error score	$0.76 \rightarrow 0.41$
MEMETRON	Black-box reward model	Reward score	$25.64 \rightarrow 37.48$

Summary: Response Engineering for Alignment



Hands-On: What You'll Practice

Notebook exercises

Part 1 — Static RE:

1. **Best-of-N:** Select best output by reward model or logprob score
2. **Self-Consistency:** Majority vote over multiple sampled outputs

Part 2 — Iterative RE:

3. **Reflexion:** Self-reflection feedback loop
4. **Mixture-of-Agents:** Aggregate outputs across multiple agents

Available on:
Google Colab
`github.com/sonthenguyen/alignment`

Key Takeaways

- Training-free alignment is practical and complements training-based approaches.
- Before reaching for fine-tuning, try training-free methods first
- Three main training-free approaches:
  - **Prompt engineering:** steer behavior pre-generation
  - **Guided decoding:** steer behavior in-generation
  - **Response engineering:** steer behavior post-generation
- Each has trade-offs:
  - **Prompt engineering:** manual is zero-shot but trial-and-error; automatic requires data but finds better prompts
  - **Guided decoding:** direct control but requires logit access or a step verifier
  - **Response engineering:** static is cheap, iterative is better but costly
- Pick one, or combine them based on your needs

Questions?

Thank you for attending!

Contact: snguye65@uic.edu | ccheng46@uic.edu | theja@uic.edu

NAIRR National Artificial Intelligence Research Resource

2026 ANNUAL MEETING
MARCH 10-13, 2026

Tutorial Survey

https://bit.ly/NAIRR26TutorialSurvey

SUPPORTED BY
NSF AWARD# 2536728

BROUGHT TO YOU BY

Thank you!