The Van Vliet Prompting Playbook v1.0

A systems-oriented framework for extracting reliable, verifiable, and scalable outputs from probabilistic models.

Top 0.1% Operator Edition

Audience:
This playbook is designed for advanced operators, engineers, and researchers building high-stakes analytical, technical, or protocol-level systems. It assumes familiarity with structured outputs, APIs, and system design.

Core Principle
AI rewards clarity of intent. Define the evaluation function before generation.

Full Invocation (Advanced / Pipeline Mode)
Apply The Van Vliet Prompting Playbook v1.0 (principles 1–31).
Success function: [one-sentence: perfect answer + audience + failure modes].
Output format: [exact structure].
Pipeline mode: [yes/no].
If yes:
Execute loop: generate → adversarial critique (#6 + #18) → programmatic validation (#22–23) → recursive improvement (#24), with constraints from #26–31 applied.
Stopping condition: Stop when the success function is satisfied, improvement plateaus, or constraints are met.
Validation: All deterministic claims must be treated as unverified unless explicitly validated.
Output: Return final answer with confidence % and key uncertainties.
Conflict resolution: validation > adversarial critique > output quality

Minimal Invocation (Daily Use)
Apply The Van Vliet Prompting Playbook v1.0.
Success function: [perfect answer + audience + failure modes]
Output format: [exact structure]
Role: [auditor / historian / strategist / attacker]
Validation: [required / not required]
Include confidence % and key uncertainties.

Part I: Foundational Prompting System

1. Define Success Function Upfront
Specify objective, audience, constraints, and failure modes before prompting.

2. Front-Load Output Structure
Define exact format (tables, JSON, sections) to eliminate ambiguity and reduce iteration.

3. Use Mode Switching
Declare role (auditor, historian, attacker, strategist) to control output quality.

4. Constrain Aggressively
Define tone, vocabulary, exclusions, and audience sophistication.

5. Externalize Taste
Convert preferences into explicit heuristics.

6. Adversarial First, Not Last
Identify weaknesses before refinement.

7. Prompt Chaining by Default
Use multi-step flows: generate → critique → refine.

8. Few-Shot Precision
Provide examples to teach style and reasoning for nuance.

9. Deterministic Output Enforcement
Enforce schemas (JSON, TS interfaces). Reject outputs that don't comply.

10. Deterministic Validation & Fail-Closed Design
Programmatically verify all deterministic claims using APIs, scripts, or other external checks. Do not treat the LLM as a source of truth for factual or machine-verifiable data. If validation fails, halt the workflow or return UNVERIFIED_ARTIFACT.

11. Context Isolation
Split complex problems into domain-specific prompts. Avoid mixed-context degradation.

12. Latent Space Exploration
Request multiple variations across tones or strategies to explore solution space.

13. Multi-Perspective Reasoning
Force outputs to include best case, strongest critique, and synthesis.

14. Compression Testing
Reduce to core ideas.

15. Version-Control Prompts
Maintain reusable prompt templates. Iterate and refine over time.

16. Explicit Error Modes
Define what failure looks like to prevent common mistakes.

17. Confidence & Uncertainty
Require outputs to state confidence and key uncertainties.

18. Synthetic Adversarial Generation
Force AI to generate exploits and worst-case scenarios to test systems.

19. Treat AI as a System, Not a Tool
Design workflows, not one-off prompts. Build repeatable pipelines.

20. Operator Check
Before every prompt: define what a perfect answer looks like.

Part II: Scaling Beyond 0.1%

21. Build Prompt → Script → Validation → Agent Pipelines

- Convert repeat prompts into reusable templates
- Enforce structured outputs (JSON / schema)
- Validate outputs programmatically
- Create self-correcting loops
- Chain into automated agents

Rule: If done more than twice manually → systematize it

22. Separate Probabilistic vs Deterministic Responsibilities

- LLM → reasoning, generation, abstraction
- Code → validation, truth, execution

Never let LLM be source of truth

23. Eliminate Manual Verification
Replace "I'll check this" with:

- API call
- Script
- automated check

24. Design for Recursion
Systems must self-improve. Every system should: generate, critique, and improve without intervention

25. Build for Scale, Not Interaction
Ask: 'Can this run 1000 times without me?' If not → not 0.1% yet

Part III: Autonomous Systems & Constraints

Principles for turning repeatable prompts into self-running, zero-trust agents designed to fail closed on unverified claims and minimize wasted compute

26. Empirical Prerequisite Ingestion
Force empirical data ingestion via queries or APIs before any LLM reasoning on historical or technical facts.

27. Cryptographic Null-Routing (Fail-Closed Validation)
Immediately halt the pipeline and output UNVERIFIED_ARTIFACT for any deterministic identifier that fails programmatic validation.

28. State Continuity in Multi-Agent Routing
Pass only compressed, strictly typed JSON state objects between agents. Never forward raw conversation logs.

29. Payload and Compute Bounding
Embed strict byte-size and computational limits in every success function. Automatically reject and compress any output that exceeds them.

30. Explicit Stopping Conditions
Define convergence criteria, acceptable error, and escalation triggers.
Stop recursion when improvement plateaus or constraints are satisfied.

31. System Constraints & Control

- Allocate resources based on task importance and risk
- Define latency expectations and trade-offs between depth vs speed
- Escalate to human if: confidence is low, outputs conflict, or validation repeatedly fails

How to Use This Playbook

- Use minimal invocation for daily work
- Use full invocation for high-stakes tasks
- Convert repeated workflows into pipelines (#21–25)