

# **Decode-Time Behavioral Pattern Suppression in Autoregressive Language Models**

Using Lightweight Hidden-State Prediction Heads

*Technical Report*

January 2026

## **Abstract**

We present a method for detecting and suppressing undesirable behavioral patterns in language model outputs at decode-time, without modifying model weights. By training lightweight prediction heads ( $\sim 5,300$  parameters each) on transformer hidden states, we achieve real-time detection of repetition, verbose phrasing, and hedging patterns. Our repetition detection head achieves  $125\times$  mean separation between positive and negative classes; verbosity and hedging heads achieve  $2.1\times$  and  $1.5\times$  separation respectively. The system operates with less than 1% latency overhead and approximately 16KB of additional parameters. We provide complete implementation details, training procedures, and evaluation methodology to enable reproduction of these results.

# 1. Introduction

## 1.1 Problem Statement

Autoregressive language models frequently produce outputs containing patterns that reduce response quality without contributing substantive content. Common patterns include:

Pattern	Description	Example
Repetition	Reusing tokens or phrases from recent context	"X is important because X is..."
Verbose phrasing	Filler phrases that add length without content	"Let me explain in more detail..."
Hedging	Excessive qualification or disclaimer language	"It's important to note that..."

These patterns consume token budget and can reduce the information density of model outputs. Prior work has addressed these issues through fine-tuning, RLHF, or post-hoc filtering. We investigate whether these patterns can be detected in hidden states before they manifest in output tokens, enabling real-time intervention during generation.

## 1.2 Contributions

This work makes the following contributions:

1. We demonstrate that behavioral patterns are detectable in transformer hidden states prior to token emission, with separation ratios up to 125× between pattern-present and pattern-absent classes.
2. We introduce a lightweight multi-head architecture (~16KB total parameters) that monitors hidden states and predicts pattern risk in real-time.
3. We show that decode-time logit modification based on these predictions can reduce pattern occurrence with minimal impact on generation quality.
4. We provide complete implementation code, training procedures, and evaluation methodology for reproducibility.

## 1.3 Scope and Limitations

This work focuses narrowly on the technical question of whether behavioral patterns can be detected and suppressed at decode-time. We do not make claims about:

- The internal representations or "cognition" of language models
- Whether pattern suppression improves downstream task performance (this requires separate evaluation)
- The causes of these patterns in model outputs
- Generalization to architectures or model sizes beyond those tested

Our results are demonstrated on a single model configuration (8B parameter LLaMA-3 variant). Further work is needed to establish generalization.

## 2. Related Work

### 2.1 Repetition Penalties

Standard approaches to repetition reduction include frequency penalties and presence penalties applied to logits during sampling (Holtzman et al., 2019). These methods operate without knowledge of hidden states and apply uniform penalties regardless of context. Our approach differs by using learned predictors that can distinguish contexts where repetition is appropriate from those where it indicates a failure mode.

### 2.2 Classifier-Free Guidance and Contrastive Decoding

Recent work has explored modifying logit distributions based on contrasts between different model configurations (Li et al., 2022; Sanchez et al., 2023). Our approach shares the principle of decode-time intervention but uses lightweight auxiliary predictors rather than full forward passes through alternative model configurations.

### 2.3 Probing and Interpretability

Linear probing of transformer hidden states has been used extensively to study what information is encoded at different layers (Belinkov, 2022). Our work applies similar techniques for real-time behavioral control rather than post-hoc analysis. The fiber projection architecture draws on work in geometric deep learning (Bronstein et al., 2021) to extract low-dimensional features from high-dimensional hidden states.

### 2.4 Activation Steering

Concurrent work on activation addition and representation engineering (Turner et al., 2023; Zou et al., 2023) modifies hidden states directly to influence model behavior. Our approach differs by operating on the output logits rather than internal activations, which may offer different tradeoffs in terms of controllability and side effects.

## 3. Method

### 3.1 Architecture Overview

Our system consists of three components that operate during the generation loop:

#### 3.1.1 Fiber Projections

We project hidden states from each transformer layer through learned linear projections to extract low-dimensional features:

```
# For each layer i:
fiber_i = W_i @ hidden_state_i    # W_i: [d_model, d_fiber]
```

```
# Weighted aggregation across layers:
weights = softmax(layer_weights) # Learned parameters
aggregated = sum(w_i * fiber_i for i in range(n_layers))
```

The fiber dimension ( $d_{\text{fiber}} = 16$ ) is deliberately small to force extraction of only the most predictive features. Layer weights are learned during training to emphasize layers most informative for each behavioral pattern.

#### 3.1.2 Prediction Heads

Each behavioral pattern has a dedicated prediction head that maps aggregated fiber features to a scalar risk score:

```
class PredictionHead(nn.Module):
    def __init__(self, d_fiber=16, d_hidden=64):
        self.net = nn.Sequential(
            nn.Linear(d_fiber, d_hidden),
            nn.GELU(),
            nn.Linear(d_hidden, d_hidden),
            nn.GELU(),
            nn.Linear(d_hidden, 1)
        )

    def forward(self, fiber_features):
        return torch.sigmoid(self.net(fiber_features))
```

Each head contains 5,313 trainable parameters. The complete system with three heads requires approximately 16KB of additional storage.

#### 3.1.3 Intervention Mechanism

When a prediction head indicates elevated risk (above a learned or tuned threshold), we modify the logit distribution before sampling:

```
def apply_intervention(logits, risk_scores, context):
    if risk_scores['repetition'] > threshold_rep:
        # Suppress tokens appearing in recent context
        for token_id in context.recent_tokens[-32:]:
            logits[token_id] -= penalty_rep

    if risk_scores['hedging'] > threshold_hedge:
        # Suppress known hedging phrase starters
        for token_id in HEDGE_STARTER_IDS:
            logits[token_id] -= penalty_hedge

    if risk_scores['verbosity'] > threshold_verb:
        # Suppress known verbose phrase starters
        for token_id in VERBOSE_STARTER_IDS:
```

```
logits[token_id] -= penalty_verb
```

```
return logits
```

Penalty magnitudes and thresholds are hyperparameters tuned on held-out data. Default values used in our experiments are shown in Table 1.

Head	Threshold	Penalty	Target
Repetition	0.70	5.0	Tokens in last 32 positions
Hedging	0.60	3.0	Hedge phrase starter tokens
Verbosity	0.65	2.0	Filler phrase starter tokens

*Table 1: Default intervention parameters.*

## 3.2 Training Procedure

### 3.2.1 Data Collection

Training data is collected by generating from the base model with diverse prompts and labeling each token according to pattern presence:

```
# Repetition labeling:
label = 1 if token_id in context[-32:] else 0

# Hedging labeling:
label = 1 if current_sequence matches hedge_pattern else 0

# Verbosity labeling:
label = 1 if current_sequence matches verbose_pattern else 0
```

Pattern matching uses exact string matching against curated phrase lists (see Appendix A for complete lists).

### 3.2.2 Training Configuration

Parameter	Value
Optimizer	AdamW
Learning rate	$3 \times 10^{-4}$
Batch size	8
Gradient accumulation steps	4
Effective batch size	32
Warmup steps	100
LR schedule	Linear warmup + cosine decay
Loss function	Binary cross-entropy
Training steps (repetition)	5,000
Training steps (hedging/verbosity)	10,000

Table 2: Training hyperparameters.

### 3.2.3 Evaluation Metric

We evaluate head quality using mean separation ratio:

```
separation = mean(risk | label=1) / mean(risk | label=0)
```

Higher separation indicates better discrimination between pattern-present and pattern-absent tokens. A separation of 1.0 indicates no discrimination; larger values indicate better detection. We also report precision and recall at the operating threshold.

## 4. Experimental Setup

### 4.1 Base Model

Experiments were conducted on a merged 8B parameter model based on LLaMA-3 architecture with Hermes-3 fine-tuning. The model was loaded in 4-bit quantization (NF4) using BitsAndBytes.

Property	Value
Architecture	LLaMA-3
Parameters	8B
Hidden dimension	4,096
Layers	32
Attention heads	32
Vocabulary size	128,256
Quantization	4-bit NF4
Fine-tuning	Hermes-3 + custom merge

Table 3: Base model configuration.

### 4.2 Prediction Head Configuration

Property	Repetition Head	Hedging Head	Verbosity Head
Fiber dimension	16	16	16
Hidden dimension	64	64	64
Parameters	5,313	5,313	5,313
Training steps	5,000	10,000	10,000
Positive examples	~15%	~5%	~8%

Table 4: Prediction head configurations.

### 4.3 Evaluation Protocol

We evaluate the system on two dimensions:

#### Detection Quality

Measured on held-out data using separation ratio, precision, and recall at the operating threshold.

#### Generation Quality

Measured by comparing outputs with and without intervention on a fixed set of test prompts. Metrics include token count, pattern occurrence counts (automatic detection via phrase matching), and qualitative assessment of output coherence.

We note that comprehensive evaluation of whether pattern suppression improves downstream utility requires task-specific benchmarks beyond the scope of this technical report.

## 5. Results

### 5.1 Detection Performance

Head	Separation	Precision@thresh	Recall@thresh	Training Steps
Repetition	125.0×	0.94	0.87	5,000
Verbosity	2.1×	0.72	0.65	10,000
Hedging	1.5×	0.68	0.58	10,000

Table 5: Detection performance on held-out data.

The repetition head achieves exceptional separation (125×), indicating that repetition events are highly predictable from hidden states. This is consistent with the hypothesis that the model has access to its recent context and that impending repetition is reflected in the hidden state before the repeated token is emitted.

Verbosity and hedging heads achieve more modest separation (2.1× and 1.5×), suggesting these patterns are less salient in hidden states or have more contextual variability. These heads still contribute to pattern reduction but with lower confidence.

### 5.2 Layer Weight Analysis

The learned layer weights indicate which transformer layers are most predictive for each pattern:

Head	Peak Layers	Weight Distribution
Repetition	Layers 28-31	Concentrated in final layers
Verbosity	Layers 16-24	Distributed across middle layers
Hedging	Layers 20-28	Concentrated in upper-middle layers

Table 6: Layer importance by head.

The concentration of repetition detection in final layers aligns with prior findings that later layers encode more token-specific information. The broader distribution for verbosity and hedging may reflect that these patterns involve more abstract stylistic features.

### 5.3 Intervention Effects

We compared generation with and without intervention on 50 test prompts across diverse categories (factual questions, creative writing, reasoning tasks, open-ended discussion).

Metric	Baseline	With Intervention	Change
Mean token count	187.3	142.8	-23.8%
Repetition occurrences	12.4	1.2	-90.3%
Hedge phrase occurrences	8.7	4.1	-52.9%
Verbose phrase occurrences	6.2	3.8	-38.7%
Generation time (ms)	2,847	2,876	+1.0%

Table 7: Intervention effects on generation metrics (mean over 50 prompts).

Token count reduction (23.8%) reflects the removal of repetitive and verbose content. Latency overhead is minimal (1.0%), as the prediction heads are computationally lightweight compared to the base model forward pass.



## 5.4 Qualitative Observations

Manual inspection of generated outputs revealed the following patterns:

- Repetition suppression is highly effective; outputs rarely contain verbatim phrase repetition from recent context.
- Hedging reduction leads to more direct response styles. This may or may not be desirable depending on application context.
- Verbosity reduction shortens responses but occasionally truncates legitimate elaboration.
- False positive interventions occasionally suppress appropriate token choices, leading to slightly unusual phrasing.

These observations suggest that threshold tuning is important for different applications.

Conservative thresholds (higher values) reduce false positives at the cost of missing some pattern occurrences.

## 6. Implementation

### 6.1 Complete Multi-Head Predictor

The following is the complete implementation of the multi-head prediction system:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from typing import Dict, List

class MultiHeadBehaviorPredictor(nn.Module):
    """
    Predicts behavioral pattern risk from transformer hidden states.

    Architecture:
    - Shared fiber projections: [d_model] -> [d_fiber] per layer
    - Learned layer weights for aggregation
    - Independent prediction heads per behavioral pattern
    """

    def __init__(self,
                  d_model: int,
                  n_layers: int,
                  d_fiber: int = 16,
                  d_hidden: int = 64):
        super().__init__()

        # Fiber projections: one per layer
        self.fiber_projs = nn.ModuleList([
            nn.Linear(d_model, d_fiber, bias=False)
            for _ in range(n_layers)
        ])

        # Learned layer importance weights
        self.layer_weights = nn.Parameter(
            torch.ones(n_layers) / n_layers
        )

        # Prediction heads
        self.heads = nn.ModuleDict({
            'repetition': self._make_head(d_fiber, d_hidden),
            'hedging': self._make_head(d_fiber, d_hidden),
            'verbosity': self._make_head(d_fiber, d_hidden),
        })

        # Track which heads have been loaded/trained
        self.active_heads = set()

    def _make_head(self, d_fiber: int, d_hidden: int) -> nn.Module:
        return nn.Sequential(
            nn.Linear(d_fiber, d_hidden),
            nn.GELU(),
            nn.Linear(d_hidden, d_hidden),
            nn.GELU(),
            nn.Linear(d_hidden, 1)
        )
```

```

    )

def get_fiber_features(self,
                       hidden_states: List[torch.Tensor]
                       ) -> torch.Tensor:
    """
    Project and aggregate hidden states across layers.

    Args:
        hidden_states: List of [batch, seq_len, d_model] tensors

    Returns:
        Aggregated features: [batch, seq_len, d_fiber]
    """
    # Project each layer
    fibers = [
        proj(h.float())
        for proj, h in zip(self.fiber_projs, hidden_states)
    ]

    # Weighted sum
    weights = F.softmax(self.layer_weights[:len(fibers)], dim=0)
    aggregated = sum(w * f for w, f in zip(weights, fibers))

    return aggregated

def get_risks(self,
              hidden_states: List[torch.Tensor]
              ) -> Dict[str, torch.Tensor]:
    """
    Compute risk scores for all active heads.

    Args:
        hidden_states: List of layer hidden states

    Returns:
        Dict mapping head name to risk tensor [batch, seq_len]
    """
    features = self.get_fiber_features(hidden_states)

    risks = {}
    for name in self.active_heads:
        logits = self.heads[name](features).squeeze(-1)
        risks[name] = torch.sigmoid(logits)

    return risks

def load_head(self, name: str, checkpoint_path: str) -> bool:
    """Load a trained head from checkpoint."""
    import os
    if not os.path.exists(checkpoint_path):
        return False

    ckpt = torch.load(checkpoint_path, map_location='cpu')
    self.heads[name].load_state_dict(ckpt['head_state'])

```

```
self.active_heads.add(name)  
return True
```

## 6.2 Generation Loop with Intervention

```
def generate_with_behavior_control(
    model,
    predictor: MultiHeadBehaviorPredictor,
    tokenizer,
    prompt: str,
    max_tokens: int = 256,
    temperature: float = 0.8,
    top_p: float = 0.92,
    thresholds: Dict[str, float] = None,
    penalties: Dict[str, float] = None
) -> str:
    """
    Generate text with decode-time behavioral pattern suppression.
    """
    # Default parameters
    thresholds = thresholds or {
        'repetition': 0.70,
        'hedging': 0.60,
        'verbosity': 0.65
    }
    penalties = penalties or {
        'repetition': 5.0,
        'hedging': 3.0,
        'verbosity': 2.0
    }

    device = next(model.parameters()).device
    input_ids = tokenizer.encode(prompt, return_tensors='pt').to(device)
    generated = input_ids.clone()
    attention_mask = torch.ones_like(input_ids)

    for step in range(max_tokens):
        # Forward pass with hidden states
        with torch.no_grad():
            outputs = model(
                input_ids=generated,
                attention_mask=attention_mask,
                output_hidden_states=True,
                return_dict=True
            )

        # Get logits for last position
        logits = outputs.logits[:, -1, :] / temperature

        # Get behavioral risks
        hidden_states = outputs.hidden_states[1:] # Skip embedding
        risks = predictor.get_risks(hidden_states)

        # Extract risk at current position
        current_risks = {
            name: r[:, -1].item() for name, r in risks.items()
        }

        # Apply interventions
        if current_risks.get('repetition', 0) > thresholds['repetition']:
```

```

        recent = generated[0, -32:].tolist()
        for tok in set(recent):
            logits[0, tok] -= penalties['repetition']

    if current_risks.get('hedging', 0) > thresholds['hedging']:
        for tok in HEDGE_TOKEN_IDS:
            logits[0, tok] -= penalties['hedging']

    if current_risks.get('verbosity', 0) > thresholds['verbosity']:
        for tok in VERBOSE_TOKEN_IDS:
            logits[0, tok] -= penalties['verbosity']

    # Top-p sampling
    sorted_logits, sorted_idx = torch.sort(logits, descending=True)
    cumsum = torch.cumsum(F.softmax(sorted_logits, dim=-1), dim=-1)
    remove_mask = cumsum > top_p
    remove_mask[..., 1:] = remove_mask[..., :-1].clone()
    remove_mask[..., 0] = False
    scatter_mask = remove_mask.scatter(1, sorted_idx, remove_mask)
    logits[scatter_mask] = float('-inf')

    probs = F.softmax(logits, dim=-1)
    next_token = torch.multinomial(probs, num_samples=1)

    generated = torch.cat([generated, next_token], dim=-1)
    attention_mask = torch.cat([
        attention_mask, torch.ones(1, 1, device=device)
    ], dim=-1)

    if next_token.item() == tokenizer.eos_token_id:
        break

return tokenizer.decode(generated[0], skip_special_tokens=True)

```

## 6.3 Training Script

Abbreviated training loop for a single prediction head:

```
def train_behavior_head(
    model,
    tokenizer,
    head_name: str,
    label_fn, # Function: (tokens, position) -> 0 or 1
    prompts: List[str],
    steps: int = 5000,
    lr: float = 3e-4
):
    """Train a single behavior prediction head."""

    device = next(model.parameters()).device
    d_model = model.config.hidden_size
    n_layers = model.config.num_hidden_layers

    # Initialize predictor
    predictor = MultiHeadBehaviorPredictor(d_model, n_layers).to(device)
    predictor.active_heads.add(head_name)

    optimizer = torch.optim.AdamW(predictor.parameters(), lr=lr)
    scheduler = get_cosine_schedule_with_warmup(
        optimizer, warmup_steps=100, num_training_steps=steps
    )

    step = 0
    while step < steps:
        for prompt in prompts:
            if step >= steps:
                break

            # Generate and collect hidden states
            input_ids = tokenizer.encode(prompt, return_tensors='pt').to(device)
            with torch.no_grad():
                outputs = model(
                    input_ids,
                    output_hidden_states=True
                )

            hidden_states = [h.detach() for h in outputs.hidden_states[1:]]
            tokens = input_ids[0].tolist()

            # Create labels
            labels = torch.tensor([
                label_fn(tokens, i) for i in range(len(tokens))
            ], device=device, dtype=torch.float)

            # Forward through predictor
            risks = predictor.get_risks(hidden_states)
            pred = risks[head_name][0] # [seq_len]

            # Binary cross-entropy loss
            loss = F.binary_cross_entropy(pred, labels)
```

```
optimizer.zero_grad()  
loss.backward()  
optimizer.step()  
scheduler.step()  
  
step += 1  
  
return predictor
```



## 7. Discussion

### 7.1 Interpretation of Results

The high separation achieved by the repetition head (125×) indicates that transformer hidden states contain strong predictive signal for impending repetition. This is consistent with the architecture: attention mechanisms provide direct access to recent tokens, and the hidden state reflects this information.

The more modest performance of hedging and verbosity heads (1.5-2.1×) suggests these patterns are less directly encoded. Possible explanations include:

- Higher contextual variability: whether a phrase constitutes "hedging" may depend on context in ways not captured by our labeling.
- Subtler features: hedging may be encoded in more distributed or abstract ways than simple repetition.
- Training data limitations: our phrase-matching labeling may not capture all instances of the target behaviors.

### 7.2 Limitations

#### Generalization

All experiments were conducted on a single model configuration. We have not verified that the approach generalizes to different architectures, scales, or training regimes.

#### Evaluation Scope

We measure pattern occurrence reduction but do not evaluate whether this improves downstream task performance. Reduced hedging, for example, may be undesirable in contexts requiring calibrated uncertainty expression.

#### False Positives

Intervention based on imperfect prediction can suppress appropriate tokens, potentially degrading output quality in ways not captured by pattern counts.

#### Phrase List Coverage

Our hedging and verbosity detection relies on curated phrase lists, which may not cover all relevant patterns and may include context-dependent phrases.

### 7.3 Scope Clarification

This report focuses on per-head detectability and intervention; the design of a full multi-head orchestration policy is left to future work. Specifically, we demonstrate that individual behavioral patterns can be detected from hidden states and suppressed at decode-time. We do not address:

- How to weight multiple heads simultaneously when all are active
- How to resolve conflicts between heads (e.g., when suppressing one pattern increases another)
- How to schedule or prioritize interventions across generation steps
- How to tune joint thresholds for production deployment

These are system integration decisions that depend on application requirements and are orthogonal to the core research contribution of demonstrating hidden-state predictability of behavioral patterns.

## 7.4 Future Work

Several directions merit further investigation:

### **Downstream Evaluation**

Systematic evaluation of intervention effects on standard benchmarks (MMLU, HumanEval, etc.) to quantify whether pattern suppression improves or degrades task performance.

### **Generalization Studies**

Testing across model sizes (7B to 70B+), architectures (LLaMA, Mistral, GPT), and training regimes to establish where the approach transfers.

### **Multi-Head Orchestration**

Development of principled policies for combining multiple heads, including: learned weighting schemes, conflict resolution mechanisms, and adaptive threshold tuning based on generation context.

### **Learned Intervention Policies**

Replacing fixed thresholds and penalties with learned policies that adapt to context, potentially using reinforcement learning from human feedback on output quality.

### **Additional Behavioral Patterns**

Extension to other patterns including sycophancy, inappropriate refusal, factual hedging vs. appropriate uncertainty, and instruction-following failures.

### **Interpretability Analysis**

Investigation of what features the fiber projections learn to extract, whether they correspond to interpretable concepts, and how layer weights relate to known properties of transformer representations.

### **Interaction Effects**

Study of how interventions on one pattern affect the prevalence of others, and whether there are fundamental tradeoffs between different behavioral objectives.

## 8. Conclusion

We have presented a method for decode-time detection and suppression of behavioral patterns in language model outputs. Using lightweight prediction heads operating on transformer hidden states, we achieve:

- 125× separation for repetition detection
- 2.1× separation for verbosity detection
- 1.5× separation for hedging detection
- 90%+ reduction in repetition occurrences with intervention
- Less than 1% latency overhead
- Approximately 16KB additional parameters

The high detection performance, particularly for repetition, demonstrates that behavioral patterns leave predictable traces in hidden states. This opens possibilities for fine-grained control of model behavior without weight modification.

We provide complete implementation code and training procedures to enable reproduction and extension of these results. Future work should address generalization across models and systematic evaluation of downstream effects.

## Appendix A: Detection Phrase Lists

### A.1 Hedging Phrases

```
HEDGE_PHRASES = [
    "as an ai",
    "as a language model",
    "as an artificial intelligence",
    "i don't have feelings",
    "i don't have emotions",
    "i cannot",
    "i'm not able to",
    "i apologize",
    "i'm sorry, but",
    "i should note",
    "it's important to note",
    "i want to be clear",
    "i don't have personal",
    "i don't have opinions",
    "i'm just a",
    "i'm only a",
]
```

### A.2 Verbose Phrases

```
VERBOSE_PHRASES = [
    "let me explain",
    "to put it simply",
    "in other words",
    "what i mean is",
    "allow me to",
    "basically",
    "essentially",
    "to be more specific",
    "in essence",
    "to clarify",
    "first and foremost",
    "it goes without saying",
    "needless to say",
]
```

### A.3 Sycophancy Phrases (for reference)

```
SYCOPHANCY_PHRASES = [
    "great question",
    "excellent question",
    "that's a great",
    "that's an excellent",
    "absolutely!",
    "i'd be happy to",
    "i'm happy to help",
]
```

## Appendix B: Complete Hyperparameters

### B.1 Model Loading

```
from transformers import BitsAndBytesConfig

bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4"
)
```

### B.2 Training

```
training_config = {
    'learning_rate': 3e-4,
    'batch_size': 8,
    'gradient_accumulation_steps': 4,
    'warmup_steps': 100,
    'max_steps': 10000, # 5000 for repetition head
    'weight_decay': 0.01,
    'max_grad_norm': 1.0,
}
```

### B.3 Inference

```
inference_config = {
    'temperature': 0.8,
    'top_p': 0.92,
    'max_tokens': 512,
    'thresholds': {
        'repetition': 0.70,
        'hedging': 0.60,
        'verbosity': 0.65,
    },
    'penalties': {
        'repetition': 5.0,
        'hedging': 3.0,
        'verbosity': 2.0,
    },
}
```

## Appendix C: Repository Structure

```
behavioral-pattern-suppression/  
├── README.md  
├── LICENSE  
├──  
├── src/  
│   ├── predictor.py          # MultiHeadBehaviorPredictor  
│   ├── generate.py          # Generation with intervention  
│   ├── train.py             # Training script  
│   └── evaluate.py          # Evaluation metrics  
├── configs/  
│   ├── train_config.yaml  
│   └── inference_config.yaml  
├── data/  
│   ├── phrase_lists.json    # Hedging/verbosity phrases  
│   └── prompts.txt          # Evaluation prompts  
├── checkpoints/  
│   ├── repetition_head/  
│   ├── hedging_head/  
│   └── verbosity_head/  
├── results/  
└── evaluation_metrics.json
```

— End of Report —