# Technical Design Document: Entropic-Guided Neuro-Symbolic Diffusion

Target Venue: ICML 2026

Version: 1.0.0

Status: Implementation Ready

## 1. System Architecture Overview

The system is a non-autoregressive language model (NAR-LM) designed for reasoning and code synthesis. Unlike standard LLMs (GPT-4, Llama 3) that generate tokens sequentially ($t \rightarrow t+1$), this model uses **Masked Discrete Diffusion** to generate, refine, and edit tokens iteratively in any order.

### 1.1 High-Level Modules

The architecture consists of three core coupled subsystems:

1. **The Diffusion Backbone (Model):** A bidirectional Transformer trained to predict masked tokens.
2. **The Entropic Planner (Policy):** A dynamic inference-time sampler that decides *which* tokens to unmask based on confidence (entropy).
3. **The Symbolic Verifier (Environment):** An external execution loop (linter/interpreter) that provides hard constraints to the diffusion process.

### 1.2 Tech Stack

* **Framework:** PyTorch (v2.4+ for FlashAttention-3 support).
* **Base Architecture:** Modified Llama-3 or LLaDA structure (Rotary Embeddings, SwiGLU, RMSNorm).
* **Tokenization:** Byte-level BPE or pure Byte-level tokenization to handle multilingual code and avoid OOV issues.
* **Verification:** ast (Python), tree-sitter (Polyglot parsing), and pytest (Execution).

## 2. Model Specification (The Backbone)

We will not use a standard Causal Decoder. We require a **Bidirectional Encoder-Decoder** or a **Non-Causal Decoder** architecture.

### 2.1 Architecture Configuration

* **Parameter Scale:** 7B - 8B (Optimal balance for iteration speed vs. reasoning capability).
* **Context Window:** 8192 tokens (Training), extendable to 32k via RoPE scaling.
* **Attention Mechanism:** Full bidirectional self-attention. No causal masking matrix $M$. Every token attends to every other token.
* **Positional Embeddings:** Rotary Positional Embeddings (RoPE), adapted for non-causal contexts.

### 2.2 Forward Process (Training)

The training objective is **Absorbing State Diffusion** (Masked Language Modeling on steroids).

Mathematical Formulation:

Given a clean sequence $x\_0$, we sample a time step $t \sim U$.

We construct a corrupted sequence $x\_t$ by masking a portion of tokens based on a schedule $\alpha\_t$.

$$q(x\_t | x\_0) = \prod\_{i=1}^L q(x\_t^i | x\_0^i)$$

Where a token becomes `` with probability $1 - \alpha\_t$.

Loss Function:

We minimize the cross-entropy of the predicted tokens at masked locations:

$$L\_{simple} = \mathbb{E}\_{t, x\_0} \left[ \sum\_{i \in Masked} -\log p\_\theta(x\_0^i | x\_t) \right]$$

Implementation Detail:

Use a "Span Masking" strategy rather than uniform random masking to encourage the model to learn local dependencies (like predicting whole function bodies).1

## 3. Inference Engine: The "Diffusion-of-Thought" Loop

This is the core innovation. Instead of model.generate(), we implement a custom sampler class EntropicNeuroSymbolicSampler.

### 3.1 Algorithm: ReMDM with Symbolic Guidance

The generation process iterates from $t=T$ (fully masked) to $t=0$ (fully generated).

**Pseudocode:**

Python

def generate(model, prompt, T\_steps=50, constraints=None):  
 # Initialize: Concatenate prompt (fixed) + mask (to be generated)  
 x\_t = tokenize(prompt) + \* output\_len  
   
 for t in range(T\_steps, 0, -1):  
 # 1. Denoising Step: Predict x\_0 from current noisy state x\_t  
 logits = model(x\_t)  
 x\_pred = sample\_categorical(logits) # Initial guess for x\_0  
   
 # 2. Symbolic Verification (The "System 2" Check)  
 if constraints:  
 # Check x\_pred against linter/compiler  
 valid\_mask, feedback = constraints.verify(x\_pred)  
   
 # If invalid, force re-masking of invalid regions  
 # "Hard" guidance: set probability of invalid tokens to 0  
 x\_pred = apply\_constraints(x\_pred, valid\_mask)  
   
 # 3. Entropic Planning (Adaptive Re-masking)  
 # Calculate entropy (uncertainty) of predictions  
 entropy = calculate\_entropy(logits)  
   
 # Determine which tokens to keep based on schedule and entropy  
 # We want to keep low-entropy (confident) tokens  
 mask\_indices = get\_mask\_indices(entropy, t, T\_steps)  
   
 # 4. Sampling Next Step  
 # Combine kept tokens with new masks  
 x\_t = transition(x\_pred, mask\_indices)  
   
 return x\_t

### 3.2 The "Re-masking" Strategy

We adopt a **"Train for the Worst, Plan for the Best"** strategy.2

* **Step 1:** The model makes a draft prediction.
* **Step 2:** We measure the entropy $H(p\_\theta(\cdot | x\_t))$.
* **Step 3:** Tokens with $H > \tau$ (threshold) are re-masked. They are "sent back" to the diffusion process to be reconsidered in the context of the confident tokens.

## 4. Neuro-Symbolic Integration (Constraints)

This module acts as the "Critic" in the Actor-Critic loop, but operates *during* generation, not after.

### 4.1 Constraint Types

1. **Syntactic Constraints (Hard):**
   * *Tool:* Tree-sitter.
   * *Action:* If generated code cannot be parsed into a valid AST, mask the tokens corresponding to the syntax error node.
2. **Logical Constraints (Soft/Hard):**
   * *Tool:* Z3 Solver or Unit Tests.
   * *Action:* If assert x > 0 fails, mask the definition of x.
3. **Process Reward Guidance (GenPRM):**
   * *Tool:* A lightweight Reward Model (fine-tuned classifier).
   * *Action:* Compute gradient $\nabla \text{Reward}(x\_t)$ and bias the logits $p\_\theta(x\_t)$ toward high-reward tokens.4

### 4.2 Implementation: The Symbolic Manifold Projection

We define a function Projection(x):

1. Decode tokens $x$ to string $S$.
2. Attempt parse(S).
3. If fail: identifying the byte-span of the error.
4. Map byte-span back to token indices.
5. Return mask vector $M\_{error}$.

## 5. Data Pipeline & Training Strategy

### 5.1 Datasets

* **Primary Source:** CommitPackFT (High-quality commit messages + code changes).
* **Reasoning Traces:** OpenMathInstruct or synthetic "Chain-of-Thought" traces.
* **Diff-Based Formatting:**
  + Input: Original Code + Issue Description
  + Target: Refined Code
  + Training masking: Mask the Refined Code part heavily (80-100%), leave Original Code mostly unmasked.

### 5.2 Training Phases

1. **Phase 1: Absorbing Diffusion Pre-training (2 weeks)**
   * Train on pure code (The Stack v2) using the objective in Section 2.2.
   * Goal: Learn the joint probability distribution of code tokens.
2. **Phase 2: Instruction Tuning (1 week)**
   * Dataset: CommitPackFT.
   * Format: <Instruction> <Code>
   * Masking: Mask the <Code> part. Train model to reconstruct code given instruction.
3. **Phase 3: Reasoning Alignment (3 days)**
   * Dataset: Hard reasoning problems (AIME/SWE-bench).
   * Objective: Train with "Reasoning Trace" included.
   * <Problem> <Solution>.

## 6. Evaluation Framework

To prove the efficacy of this architecture for ICML 2026, we must benchmark against AR models (e.g., DeepSeek-Coder, Llama-3-Code).

### 6.1 Metrics

1. **Pass@1 (Code):** Standard HumanEval/MBPP.
2. **Repair Rate (SWE-bench):** Can the model fix a bug given a repo context?
   * *Hypothesis:* Our model should win here because it can "edit" (inpaint) without regenerating the whole file.5
3. **Inference Cost:** Total FLOPs per correct solution.
   * *Target:* Show that 50 diffusion steps < 1000 AR tokens in wall-clock time due to parallelism.

## 7. Development Roadmap

1. **Week 1-2:** Setup LLaDA-style backbone. Implement basic bidirectional training loop on a small dataset (e.g., TinyStories or localized code subset).
2. **Week 3-4:** Implement the EntropicNeuroSymbolicSampler. Get basic "Inpainting" working (mask random code, have model fix it).
3. **Week 5-6:** Integrate tree-sitter for syntactic guidance. Run ablation: Diffusion vs. Diffusion + Syntax.
4. **Week 7-10:** Large-scale training on 8xH100s.
5. **Week 11:** Eval on SWE-bench Lite.

#### Works cited

1. Compositional Diffusion-Based Continuous Constraint Solvers - arXiv, accessed November 24, 2025, <https://arxiv.org/pdf/2309.00966>
2. [2502.06768] Train for the Worst, Plan for the Best: Understanding Token Ordering in Masked Diffusions - arXiv, accessed November 24, 2025, <https://arxiv.org/abs/2502.06768>
3. Train for the Worst, Plan for the Best: Understanding Token Ordering in Masked Diffusions - ChatPaper, accessed November 24, 2025, <https://chatpaper.com/paper/106691>
4. GenPRM: Scaling Test-Time Compute of Process Reward Models via Generative Reasoning | alphaXiv, accessed November 24, 2025, <https://www.alphaxiv.org/overview/2504.00891v1>
5. Improve Code Editor Efficiency: Avoid Full Code Regeneration on Edits - Feature requests, accessed November 24, 2025, <https://community.openai.com/t/improve-code-editor-efficiency-avoid-full-code-regeneration-on-edits/1268599>