The Judge & The Curve: Video Generation Report

Project Overview

This educational video explains the mathematical foundations behind AI reward models and regression, demonstrating how Large Language Models learn to evaluate response quality through regression-based scoring systems.

Duration: 2 minutes (120 seconds) **Resolution:** 1920x1080 @ 60fps **Format:** Animated mathematical visualization with synchronized voiceover

---

Tools & Technologies

Tool	Purpose
Manim	Mathematical animation engine for creating precise visualizations
manim-voiceover	Synchronized text-to-speech narration integration
gTTS	Google Text-to-Speech service for voiceover generation
Python 3.14	Core programming language with strict type checking
NumPy	Data generation for regression visualizations
Nix + direnv	Reproducible development environment
Pyright	Static type checking (strict mode)
Ruff	Code linting and formatting

Teaching Strategy

Our pedagogical approach follows a **Problem → Naive Solution → Theory** structure:

1. Problem: The Hallucination (Section 1)

We open with AI generating incoherent output—"The moon is made of blue algorithm potato…"—establishing that **intelligence without measurement is chaos**. This creates cognitive dissonance and motivates the need for evaluation.

2. Naive Solution: Binary Judgment (Section 2)

We introduce the concept of scoring but emphasize that **simple yes/no classification is insufficient**. A sliding gauge demonstrates continuous scoring (0.99

for correct answers, -0.50 for incorrect), leading to the definition: *Regression predicts a quantity.*

3. Theory: Linear & Non-Linear Regression (Sections 3-4)

- **Linear Regression**: Study hours vs. test scores—a perfect world where `y = mx + b` minimizes error (residuals)
- **Non-Linear Regression**: Drug dosage vs. patient health—demonstrating why **reality curves** and linear models fail on biological data

4. Synthesis (Section 5)

The curved regression line transforms into a single neuron, then reveals a neural network—showing that **AI judges are millions of non-linear regressions stacked together**.

Project Architecture

Directory Structure

```
Manim-ML/
  src/
     config.py              # Central configuration (colors, timing, constants)
     utils/
        color_utils.py     # Color manipulation helpers
        data_generator.py  # Reproducible regression data (seed=42)
  videos/
     scenes/                # 5 scene modules + complete composition
        section1_hallucination.py
        section2_scale.py
        section3_linear.py
        section4_nonlinear.py
        section5_synthesis.py
        judge_curve_complete.py  # Main composition
     templates/
        animations.py      # Reusable animation functions
        effects.py         # Visual effects (glow, chromatic aberration)
        custom_shapes.py   # Custom Manim mobjects
  media/                     # Generated output (git-ignored)
```

Manim Architecture Patterns

1. VoiceoverScene Base Class All scenes extend `VoiceoverScene` from manim-voiceover, enabling synchronized narration:

2

```python
class LinearRegressionScene(VoiceoverScene):
    def construct(self) -> None:
        self.set_speech_service(GTTSService())
        with self.voiceover(text=self.VOICEOVER_TEXT) as tracker:
            self.play(Create(graph), run_time=tracker.duration)
```

2. Centralized Configuration Single source of truth in `src/config.py` using frozen dataclasses: - `ColorPalette`: Consistent hex colors across all scenes - `TimingConfig`: Animation duration categories (FAST/MEDIUM/SLOW) - `SceneConfig`: Precise timing boundaries (start/end timestamps)

3. Composable Scene Rendering The `JudgeCurveComplete` class orchestrates all 5 sections by calling each scene's `render_animations()` method, passing the parent scene context for seamless transitions.

4. Reproducible Data Generation NumPy-based data generators use a fixed seed (`DATA_SEED = 42`) ensuring identical scatter plots across renders: - `generate_linear_data()`: Upward-trending points for Section 3 - `generate_nonlinear_data()`: Dose-response curves for Section 4

5. Reusable Effects Library `videos/templates/effects.py` provides visual effects: - `apply_chromatic_aberration()`: RGB-split distortion effect - `apply_glow_effect()`: Neon glow for emphasis - `create_static_noise_overlay()`: TV static for hallucination scene

---

Team Members

Member	GitHub	Contributions
Enrique	@Enriquefft	Script writing, voiceover production, codebase architecture & setup
Joaquin Arriaga	@joaquinarriaga-ui	Section 1 (Hallucination) & Section 2 (Continuous Scale)
Renato	@jfloredev	Section 3 (Linear Regression) & Section 4 (Non-Linear Regression)

Final Video

Watch the video

Generated with Manim-ML framework | MIT License