

Enhancing Controllability and Performance in Stand-Up Comedy Script Generation via Graph-of-Thought Modeling

TEAM: 장진우 , 안진영 , Zahra Gholami, Bahareh Toushkan

LLMs can generate creative text — but linear prompts struggle with:

- Limited narrative control
- Weak logical/emotional buildup
- Inconsistent humor quality

[1] S. Jentzsch and K. Kersting, “ChatGPT is fun, but it is not funny! Humor is still challenging Large Language Models”, 2023.

Prompt engineering

```
Write a stand-up comedy script with the
following structure:
setup: talk about your experience flying for
the first time
incongruity: mention an unexpected aspect
of the airline food
punchline: deliver a funny line about the
surprise
callback: later, reference to airline food
again humorously
```

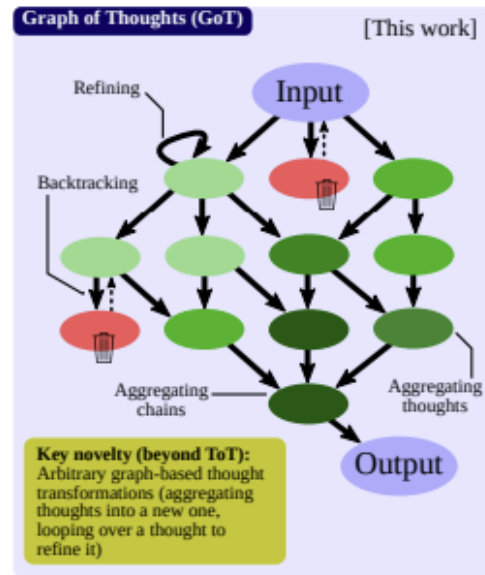
Key Idea: Graph-Of-Thoughts(GoT)

Reasoning over tree rather than sequence.

Each comedic element = a node

Edges = semantic / temporal relations

[2] S. Besta *et al.*, “Graph of Thoughts: Solving Elaborate Problems with Large Language Models”, 2024.



Hypothesis & Objectives

Hypothesis

GoT-based LLM > Linearly prompt-engineered LLM > Unstructured LLM
in humor quality & controllability.

Objectives

- Design GoT representation for comedy.
- Fine-tune on *zachgitt/comedy-transcripts* with LoRA
- Implement graph controller for generation order.
- Compare with linear baseline.

[3] E. J. Hu et. al., “LoRA: Low-Rank Adaptation of Large Language Models”, 2021.

Dataset: *zachgitt/comedy-transcripts*
(w/ short description, sample comedians).

Auto-label with LLM

Label	Example
Setup	"I just moved to New York..."
Incongruity	"...and my roommate's a cat."
Punchline	"He still pays rent though!"
Callback	"Remember the cat?"

Auto-labeling Prompt Example

[Task]

You are labeling each line of a stand-up comedy transcript with its comedic role.

Possible labels:

- Setup: Introduces the topic or premise
- Incongruity: Adds a surprising or conflicting element
- Punchline: Delivers the main joke or humor
- Callback: References an earlier punchline or setup

[Transcript]

1. "I tried to cook healthy last week."
2. "Turns out my idea of healthy is microwaving kale chips."
3. "Now my smoke alarm has abs."
4. "Even my smoke alarm started giving nutrition advice."

[Instruction]

For each line, return:

Line Number | Label |

Training – Baseline 1 (w/o Structure, w/o Graph)

- Fine-tune on pretrained model (e.g. llama-8b)
 - Fine-tune data example:

Generate a stand-up comedy transcript.
Follow a format of {LINE NUMBER: "SENTENCE"}

Output:

- 1: "I tried to cook healthy last week."
- 2: "Turns out my idea of healthy is microwaving kale chips."
- 3: "Now my smoke alarm has abs."
- 4: "Even my smoke alarm started giving nutrition advice."

Training – Baseline 2 (w/o Graph)

- **Fine-tune on pretrained model (e.g. llama-8b)**
 - **Fine-tune data example:**

Generate a stand-up comedy transcript.

Each line must have a type: Setup, Incongruity, Punchline, or Callback.

Choose the most appropriate type for each line based on context.

Follow a format of {LINE NUMBER (TYPE): "SENTENCE"}

Output:

1 (Setup): "I tried to cook healthy last week."

2 (Incongruity): "Turns out my idea of healthy is microwaving kale chips."

3 (Punchline): "Now my smoke alarm has abs."

4 (Callback): "Even my smoke alarm started giving nutrition advice."

Training – Our Proposed Model

- **Firstly, construct graph from the linear data with LLM**

[Task]

Convert a labeled stand-up comedy transcript into a graph.

Each line becomes a node with the following fields:

- Node ID: unique identifier (Node_1, Node_2, etc.)
- Node Type: Setup, Incongruity, Punchline, or Callback
- Content: the line itself
- Connections: list of Node IDs this node depends on (empty if none)

[Input: Labeled Lines]

- 1 | Setup: "I tried to cook healthy last week."
- 2 | Incongruity: "Turns out my idea of healthy is microwaving kale chips."
- 3 | Punchline: "Now my smoke alarm has abs."
- 4 | Callback: "Even my smoke alarm started giving nutrition advice."

[Instruction]

1. Assign Node IDs sequentially (Node_1, Node_2, ...).
2. Create pseudo-connections based on these rules:
 - Incongruity → nearest previous Setup
 - Punchline → nearest previous Incongruity
 - Callback → nearest previous Punchline
3. Return the graph in JSON format with keys: node_id, type, content, connections.

```
[
  {
    "node_id": "Node_1",
    "type": "Setup",
    "content": "I tried to cook healthy last week.",
    "connections": []
  },
  {
    "node_id": "Node_2",
    "type": "Incongruity",
    "content": "Turns out my idea of healthy is microwaving kale chips.",
    "connections": ["Node_1"]
  },
  {
    "node_id": "Node_3",
    "type": "Punchline",
    "content": "Now my smoke alarm has abs.",
    "connections": ["Node_2"]
  },
  {
    "node_id": "Node_4",
    "type": "Callback",
    "content": "Even my smoke alarm started giving nutrition advice.",
    "connections": ["Node_3"]
  }
]
```

Training – Our Proposed Model

- **Fine-tune on pretrained model (e.g. llama-8b)**
 - **Fine-tune data example:**

Generate the next node in the stand-up comedy graph.

Each node must have a type: Setup, Incongruity, Punchline, or Callback.

Specify which existing node(s) this new node connects to.

Follow a format of {Node_<ID> (<Type>) connected to [<Node_IDs>]: "<Content>"}

Output:

Node_1 (Setup) connected to []: "I tried to cook healthy last week."

Node_2 (Incongruity) connected to [Node_1]: "Turns out my idea of healthy is microwaving kale chips."

Node_3 (Punchline) connected to [Node_2]: "Now my smoke alarm has abs."

Node_4 (Callback) connected to [Node_3]: "Even my smoke alarm started giving nutrition advice."

Evaluation Plan

Category	Metric	Description	Evaluation Method
Automatic	Coherence (BERTScore)	Measures semantic consistency between sentences and overall flow	Compute similarity between adjacent segments
	LLM-based AI-likeness Judgment	Measures how “human-like” or natural the generated comedy feels compared to real data	Given a real and a generated transcript, an LLM is asked “ <i>Which one seems more AI-written?</i> ” to assess stylistic naturalness
Human (1–5 Likert)	Humor Quality	How funny the generated script feels overall	Human raters score funniness (1 = not funny, 5 = very funny)
	Story Flow	Smoothness of transitions and buildup	Raters judge logical and emotional continuity
	Controllability	Ease of steering topics or style	Raters assess how well model follows instructions

Expected Results & Contributions

- Improved humor coherence & pacing
- Graph-controlled generation toolkit (open source)
- Benchmark for humor generation research

Thank You!

A good joke is logic with surprise — our goal is to teach AI that surprise

QnA
