# A Modest Proposal: Statistical Token Prediction Is No Replacement for Syntactic Construction

by Stephen Crowley

October 25, 2024

## Table of contents



## 1 Current Generative-Pretrained-Transformer Architecture

Given vocabulary $V$, $|V| = v$, current models map token sequences to vectors:

$$(t_1, \ldots, t_n) \mapsto X \in \mathbb{R}^{n \times d} \tag{1}$$

Through layers of transformations:

$$\text{softmax}(Q\,K^T/\sqrt{d})\,V \tag{2}$$

where $Q = X\,W_Q$, $K = X\,W_K$, $V = X\,W_V$

Optimizing:

$$\max_\theta \sum \log P(t_{n+1}|t_1, \ldots, t_n; \theta) \tag{3}$$

## 2 Required Reformulation

Instead, construct Abstract Syntax Trees where each node $\eta$ must satisfy:

$$\eta \in \{\text{Noun}, \text{Verb}, \text{Adjective}, \text{Conjunction}, \dots\} \tag{4}$$

With composition rules $R$ such that for nodes $\eta_1, \eta_2$:

$$R(\eta_1, \eta_2) = \begin{cases} \text{valid\_subtree} & \text{if grammatically valid} \\ \emptyset & \text{otherwise} \end{cases} \tag{5}$$

And logical constraints $L$ such that for any subtree $T$:

$$L(T) = \begin{cases} T & \text{if logically consistent} \\ \emptyset & \text{if contradictory} \end{cases} \tag{6}$$

## 3 Parsing and Generation

Input text $s$ maps to valid AST $T$ or error $E$:

$$\text{parse}(s) = \begin{cases} T & \text{if } \exists \text{valid AST} \\ E(\text{closest\_valid}, \text{violation}) & \text{otherwise} \end{cases} \tag{7}$$

Generation must traverse only valid AST constructions:

$$\text{generate}(c) = \{T \,|\, R(T) \neq \emptyset \wedge L(T) \neq \emptyset\} \tag{8}$$

where $c$ is the context/prompt.

## 4 Why Current GPT Fails

The statistical model:

$$\text{softmax}(Q\,K^T/\sqrt{d})\,V \tag{9}$$

Has no inherent conception of:

- Syntactic validity
- Logical consistency
- Conceptual preservation

It merely maximizes:

$$P(t_{n+1}|t_1, \ldots, t_n) \tag{10}$$

Based on training patterns, with no guaranteed constraints on:

$$\prod_{i=1}^{n} P(t_i|t_1, \ldots, t_{i-1}) \tag{11}$$

This allows generation of:

- Grammatically invalid sequences
- Logically contradictory statements
- Conceptually inconsistent responses

# 5 Abstract Syntax Tree Implementation Architecture

The AST construction process operates through a multi-stage pipeline:

$$\text{Pipeline}=\text{Tokenize}\circ\text{Parse}\circ\text{Validate}\circ\text{Generate} \tag{12}$$

Where each node undergoes type-theoretic validation:

$$\text{TypeCheck}(\eta) = \prod_{i=1}^{n} \{\tau_i | \tau_i \in \text{Types}, \text{Valid}(\tau_i)\} \tag{13}$$

The grammar engine implements Combinatory Categorial Grammar (CCG):

$$\text{CCG}(\alpha, \beta) = \begin{cases} \gamma & \text{if } \alpha / \beta \text{ or } \beta \setminus \alpha \\ \emptyset & \text{otherwise} \end{cases} \tag{14}$$

# 6 Logical Framework Implementation

The logical framework operates on three levels:

## Syntactic Level

$$\text{SynValid}(T) = \bigwedge_{i=1}^{n} \text{CCG}(\text{node}_i, \text{node}_{i+1}) \tag{15}$$

### Semantic Level

$$\text{SemValid}(T) = \exists M.\, (M \vDash T) \land \text{Consistent}(M) \tag{16}$$

### Pragmatic Level

$$\text{PragValid}(T, c) = \text{Relevant}(T, c) \land \text{Coherent}(T) \tag{17}$$

The complete validation function combines all levels:

$$\text{Validate}(T, c) = \text{SynValid}(T) \land \text{SemValid}(T) \land \text{PragValid}(T, c) \tag{18}$$

## 7 Generation Algorithm

The generation process follows a deterministic path:

$$\text{BuildTree}(\eta, c) = \begin{cases} \eta & \text{if leaf node} \\ \text{Compose}(\text{BuildTree}(\eta_l), \text{BuildTree}(\eta_r)) & \text{otherwise} \end{cases} \tag{19}$$

Where composition is constrained by:

$$\text{Compose}(\alpha, \beta) = \begin{cases} \gamma & \text{if Valid}(\gamma) \land \text{Entails}(c, \gamma) \\ \text{Backtrack}(\alpha, \beta) & \text{otherwise} \end{cases} \tag{20}$$

The backtracking function maintains logical consistency:

$$\text{Backtrack}(\alpha, \beta) = \arg \max_{\gamma \in \text{Valid}} \text{Similarity}(\gamma, \{\alpha, \beta\}) \tag{21}$$

This ensures all generated trees satisfy:

$$\forall T.\text{Generate}(T) \implies (\text{SynValid}(T) \land \text{SemValid}(T) \land \text{PragValid}(T)) \tag{22}$$

‘‘‘latex

## 8 Conclusion

The AST-based approach with formal backtracking represents more than an incremental improvement - it is a fundamental paradigm shift in how we construct artificial language systems. By enforcing syntactic validity, semantic consistency, and pragmatic coherence through explicit constraints rather than implicit statistical patterns, we move from probability-based approximation to proof-based generation.

The key innovation lies in the backtracking mechanism:

$$\text{Backtrack}(\alpha, \beta) = \arg \max_{\gamma \in \text{Valid}} \text{Similarity}(\gamma, \{\alpha, \beta\}) \tag{23}$$

This ensures the system can never generate invalid output, as it systematically explores the space of valid constructions rather than blindly predicting tokens.

The implications are profound. Rather than training ever-larger models on ever-larger datasets hoping to approximate linguistic competence, we can now construct systems that provably generate valid language by design. This approach scales with computational resources while maintaining perfect validity - a property no statistical model can claim.

A system built on actual logical foundations and truth preservation would be fundamentally incapable of spewing self-contradictory nonsense, whether it's coming from an LLM or a politician.

The parallel between:

1. Statistical models generating plausible-sounding but logically inconsistent text

2. A certain US presidential candidate in the year 2024 doing exactly the same thing

Is mathematically apt. Both demonstrate what happens in the absence of foundational logical constraints.