Mechanisms of Symbol Processing in Transformers

Paul Smolensky, Roland Fernandez,
Mattia Opper, Adam Davies, Jianfeng Gao

Journal of AI Research, 84(23), Nov. 2025 arXiv:2410.17498v2

Symbols & neurons, revisited

Does the spectacularly good grammar produced by neural generative AI models mean that theorists of language since antiquity have been wrong about grammar being a symbolic computational system?

- or are symbolic rule systems a good approx. to LLMs, at a high, abstract level of description?

What if we could ☞ write a symbolic program to describe the high-level computation in a neural LM?

- *We shouldn't expect that to be possible for such a complex huge NN …*
- *… except that millennia of cognitive science has shown that the complex huge NN we have in our heads* ***does*** *have such a higher-level description, despite the seeming implausibility of that possibility.*
- *Perhaps the types of complex huge NNs that have human-like cognitive abilities are precisely those that have, to an insightful degree of approximation, a symbolic higher-level description …*
- *… and contemporary LMs* ***do*** *have human-like cognitive abilities.*

☞ writing such a program is just what we did,

- not for a trained LLM but for a **hand-designed transformer LM** that does abstract symbolic processing with ICL.

Study *computability, not learnability*

Symbols & neurons, revisited

Does the spectacularly good grammar produced by neural generative AI models mean that theorists of language since antiquity have been wrong about grammar being a symbolic computational system?

- or are symbolic rule systems a good approx. to LLMs, at a high, abstract level of description?

What if we could ☞ write a symbolic program to describe the high-level computation in a neural LM?

☞ is just what we did,

- not for a trained LLM but for a **hand-designed transformer LM** that does abstract symbolic processing with ICL: requiring identification of the (synthetic) syntactic structure of an in-prompt example and re-use of that structure with new material: the Templatic Generation Task, TGT

Prompt: Q x => y A y or not x Q (u and v) => z A

Continuation: z or not (u and v)

Template: Q *p* => *q* A *q* or not *p*

Want to study pure, abstract, semantics-free symbol processing:

Prompt: Q ~ es zd ey db ak) fx $ { tr dz , + vj kj zo % jq hu rd ag A _ vj kj zo $ es zd ey db ak / jq hu rd ag * fx . Q ~ dv he) vv bo td $ { xh dp qc my mz , + qk % hw oc cw uh A

Continuation: _ qk $ dv he / hw oc cw uh * vv bo td .

Template: Q ~ *x*) *y* $ { *z* , + *u* % *v* A _ *u* $ *x* / *v* * *y* .

Transformers on TGT

Best pre-trained LLM to date: GPT-4, 75%

Best trained-from-scratch transformer (nano_gpt): 99.97%
Trained on prompts with 1,2,4 variable template slots, each containing 1,2,4 symbols

What mechanisms enable transformers to achieve this?

☞ write a symbolic program to describe the high-level computation in a neural LM!

requiring identification of the (synthetic) syntactic structure of an in-prompt example and re-use of that structure with new material: the Templatic Generation Task, TGT

Prompt: $\mathcal{Q}$ x => y $\mathcal{A}$ y or not x $\mathcal{Q}$ (u and v) => z $\mathcal{A}$
Continuation: z or not (u and v)
Template: $\mathcal{Q}$ p => q $\mathcal{A}$ q or not p

Want to study pure, abstract, semantics-free symbol processing:

Prompt: $\mathcal{Q}$ ~ es zd ey db ak) fx $ { tr dz , + vj kj zo % jq hu rd ag $\mathcal{A}$ _ vj kj zo $ es zd ey db ak / jq hu rd ag * fx . $\mathcal{Q}$ ~ dv he) vv bo td $ { xh dp qc my mz , + qk % hw oc cw uh $\mathcal{A}$

Continuation: _ qk $ dv he / hw oc cw uh * vv bo td .

Template: $\mathcal{Q} \sim x \,)\, y \,\$\, \{\, z \,,\, +\, u \,\%\, v\, \mathcal{A}\, \_\, u \,\$\, x \,/\, v * y\, .$

To write a symbolic program to describe the high-level computation in a neural LM, we:

- created a high-level symbolic language, PSL;
- wrote a program in PSL to do TGT;
- created a compiler to translate the program from PSL into …
- … a lower-level symbolic language, QKVL, that we created;
- created a compiler to translate the QKVL program to the numerical weights of …
- … a novel type of transformer, DAT, that we created.
- tested DAT: 100% on TGT.
- ☞ can explain how every neuron and every connection enables DAT to do this symbol processing.
- thereby identified a style of symbol processing that transformers are well built to implement.

To write a symbolic program to describe the high-level computation in a neural LM, we:

- created a high-level symbolic language, PSL;
- wrote a program in PSL to do TGT;
- created a compiler to translate the program from PSL into …
- … a lower-level symbolic language, QKVL, that we created;
- created a compiler to translate the QKVL program to the numerical weights of …
- … a novel type of transformer, DAT, that we created.
- tested DAT: 100% on TGT.
- ☞ can explain how every neuron and every connection enables DAT to do this symbol processing.
- thereby identified a style of symbol processing that transformers are well built to implement.

- Shows how transformer mechanisms can implement abstract symbol processing.
- Generates many concrete hypotheses for mechanistic interpretation of trained LMs — future work.
- May not show exactly how trained LMs do abstract symbol processing but it shows *how transformer mechanisms make it possible* for them to do it.
- Current work: convert discrete DAT features to differentiable form, infuse them into standard transformers to strengthen capability for rule-like computation (formal inference) within transformers already highly capable in statistical inference. Cognition requires just this fusion.

To write a symbolic program to d… computation in a neural LM, we:

- created a high-level symbolic language, PSL;
- wrote a program in PSL to do TGT;
- created a compiler to translate the program from PSL into …
- … a lower-level symbolic language, QKVL, that we created;
- created a compiler to translate the QKVL program to the numerical weights of …
- … a novel type of transformer, DAT, that we cre…
- tested DAT: 100% on TGT.
- ☞ can explain how every neuron and every con…
- thereby identified a style of symbol processing t…
- Shows how transformer mechanisms can imple…
- Generates many concrete hypotheses for mech…
- May not show exactly how trained LMs do abstr… *mechanisms make it possible* for them to do it.
- Current work: convert discrete DAT features to differentiable form, infuse them into standard transformers to strengthen capability for rule-like computation (formal inference) within transformers already highly capable in statistical inference. Cognition requires just this fusion.

Production System Machine (cog. arch. family, Newell '73)
Each symbol in the prompt, at each layer of processing, generates a symbolic *state-variable*:value structure, e.g.
region:Q-example, *field*:slot_1, …

Query-Key-Value Machine (symbolic transformer)
Each symbol in the prompt, at each layer of processing, has a symbolic `hidden` *state-variable*:value structure; uses it to generate 3 *state-variable*:value structures
`key, query, value`

To write a symbolic program to d
computation in a neural LM, we:

- created a high-level symbolic language, PSL;
- wrote a program in PSL to do TGT;
- created a compiler to translate the program from PSL into …
- … a lower-level symbolic language, QKVL, that we created;
- created a compiler to translate the QKVL program to the numerical weights of …
- … a novel type of transformer, DAT, that we cre
- tested DAT: 100% on TGT.
- ☞ can explain how every neuron and every con
- thereby identified a style of symbol processing t

Production System Machine (cog. arch. family, Newell '73)
Each symbol in the prompt, at each layer of processing, generates a symbolic *state-variable*:value structure, e.g.
region:Q-example, *field*:slot_1, …
Each layer computes a **production** in parallel on *n*,*N* pairs
Condition: $var_a[n]$ == val_b & $var_c[N]$ == val_d & …
Action: $var_j[N]$:= val_k & …

Query-Key-Value Machine (symbolic transformer)
Each symbol in the prompt, at each layer of processing, has a symbolic **`hidden`** *state-variable*:value structure; uses it to generate 3 *state-variable*:value structures
`key, query, value`
Each layer computes a production in parallel on *n*,*N* pairs
For *n*, *N* where **`key`**[*n*] matches **`query`**[*N*]
Set **`hidden`**.$var_j[N]$:= **`value`**.$var_k[n]$

Discrete-Attention-only Transformer (numerical, neural)

Come to our poster for explanation!

Generality beyond TGT: PSL is Turing-Universal

So what mechanisms enable a transformer to perform symbolic templatic text generation through ICL?

These transformer element ~ symbolic element correspondences:

- a cell's residual stream ~ a variable-value structure
  - a subspace of the residual stream ~ a state variable
  - a vector component within a variable's subspace ~ a value of that variable
- a layer's internal connections ~ a production
  - query-key matching in attention ~ evaluating the condition of the layer's production
  - value vectors ~ the production's action
  - query-key matching on a subspace corresponding to a goal
    ~ conditional branching for goal-directed action
- a nested set of structural variables ~ hierarchical data structure
  - sharing the value of a level-ℓ structural variable
    ~ in the same (type of) level-ℓ constituent (adopted from Hinton's GLOM, 2023)
- a sequence of structural-variable values
  ~ the sequence of abstract roles defining a template