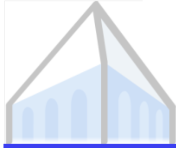
Natural Language Processing

Compositional Semantics and Structured Representations

Weakly Supervised Learning



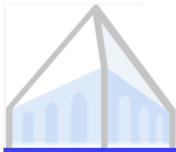
Supervision: Logical Forms

- Data: input sentences paired with annotated LFs

  Show me flights to Prague

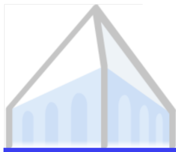
  $\lambda x.flight(x) \wedge to(x, PRG)$

- Problem: no supervision on how to get from sentence to LF
- But we can assume our LF has been generated from some formal grammar
- Combinatory Categorial Grammar (CCG)



CCG: Lexicon

Words	Category
flights	N : $\lambda x.flight(x)$
to	(N\N)/NP : $\lambda x.\lambda f.\lambda y.f(x) \wedge to(y,x)$
Prague	NP : *PRG*
New York city	NP : *NYC*
…	…



CCG: Combinators

Application

- $X/Y : f \quad Y : a \quad \Rightarrow \quad X : f(a)$
- $Y : a \quad X\backslash Y : f \quad \Rightarrow \quad X : f(a)$

Composition

- $X/Y : f \quad Y/Z : g \quad \Rightarrow \quad X/Z : \lambda x.f(g(x))$
- $Y\backslash Z : f \quad X\backslash Y : g \quad \Rightarrow \quad X\backslash Z : \lambda x.f(g(x))$

CCG: Parsing

Show me	flights	to	Prague
S/N	N	(N\N)/NP	NP
$\lambda f.f$	$\lambda x.flight(x)$	$\lambda y.\lambda f.\lambda x.f(y) \wedge to(x,y)$	PRG

to Prague:

N\N

$\lambda f.\lambda x.f(x) \wedge to(x,PRG)$

flights to Prague:

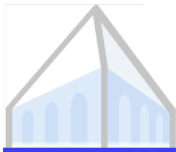
N

$\lambda x.flight(x) \wedge to(x,PRG)$

Show me flights to Prague:

S

$\lambda x.flight(x) \wedge to(x,PRG)$



Weighted CCG

- Lexicon Λ
- GEN: all possible parses *y* for sentence *x* given the lexicon
- Feature function

$$f : X \times Y \to \mathbb{R}^m$$

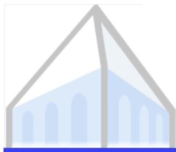
- (Learned) weights

$$w \in \mathbb{R}^m$$

- Best parse:

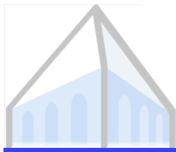
$$y^* = \arg \max_{y \in GEN(x,\Lambda)} w \cdot f(x,y)$$

Words	Category
flights	N : λ*x*.*flight*(*x*)
to	(N\N)/NP : λ*x*.λ*f*.λ*y*.*f*(*x*) ∧ *to*(*y*,*x*)
Prague	NP : *PRG*
New York city	NP : *NYC*
...	...



Training (ZC05/07)

- Start with (x, z) sentence-LF pairs and a small seed lexicon
- Iterate *T* times:
  - Propose new lexical entries from each example (x, z):
    - Generate all possible lexical entries pairing words/phrases in *x* with predicates in *z*
    - Use GEN to get all possible parses of *x* given the existing and new lexicon
    - Find the best parse *y* among these and add its lexical entries to the existing lexicon



GENLEX: Substrings X Categories

Input Training Example

Sentence:	`Show me flights to Prague.`
Logic Form:	$\lambda x.flight(x) \wedge to(x,PRG)$

Output Lexicon

All possible substrings:

```
Show
me
flights
Show me
Show me flights
Show me flights to
...
```

X

Categories created by rules that trigger on the logical form:

NP : PRG

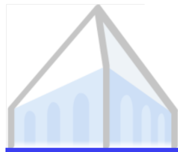
N : $\lambda x.flight(x)$

(S\NP)/NP : $\lambda x.\lambda y.to(y,x)$

(N\N)/NP : $\lambda y.\lambda f.\lambda x.$ …

...

[Zettlemoyer & Collins 2005]

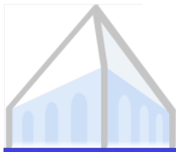


Training (ZC05/07)

- Start with (x, z) sentence-LF pairs and a small seed lexicon
- Iterate *T* times:
  - Propose new lexical entries from each example (x, z)
  - Update weights:
    - Re-parse all examples using newest lexicon and GEN
    - Sort parses into “good” and “bad” according to whether they are valid or invalid
    - Update weights to upweight “good” parses and downweight “bad” parses

Training (ZC05/07)

- Start with (x, z) sentence-LF pairs and a small seed lexicon
- Iterate *T* times:
  - Propose new lexical entries from each example (x, z)
  - Update weights
- Return full lexicon and weights

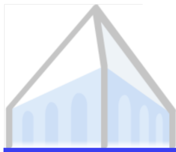


Supervision: Denotations Only

- Data: input sentences paired with denotations only (no LFs)

```
Show me flights to Prague
Flight #s: 123, 456, 78, 342
```

- Problem: no LF supervision at all!
  - Even worse problem of spuriousness
  - Complicates lexicon building
- Can still take advantage of knowing there’s a (latent) structured representation



Learning from Denotations

- Example applications:
  - Grounded QA
  - Instruction following
  - Truth-conditional semantics
- Modification of ZC05/07 approach
  - New validation function: does proposed parse+LF yield expected denotation?
  - New method for generating lexical entries: place constraints (e.g., type constraints) on possible new entries

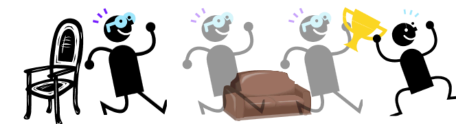
Year	City	Country	Nations
1896	Athens	Greece	14
1900	Paris	France	24
1904	St. Louis	USA	12
...	...	...	...
2004	Athens	Greece	201
2008	Beijing	China	204
2012	London	UK	204

x = Greece held its last Summer Olympics in which year?

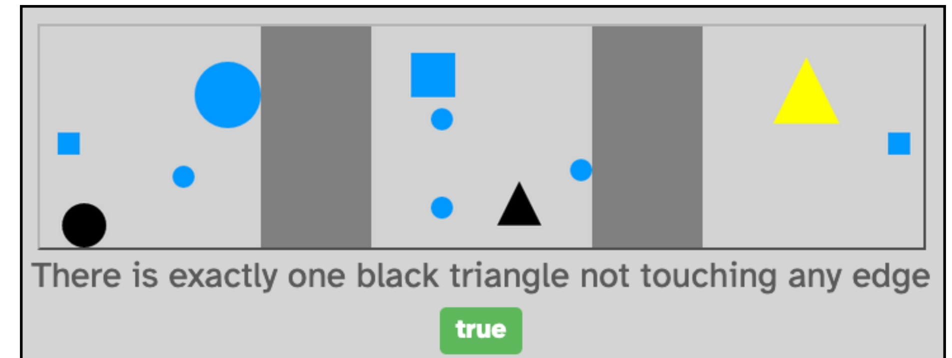
y = 2004

WikiTableQuestions, Pasupat and Liang 2015, ACL

at the chair, move forward three steps past the sofa

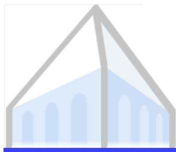


Artzi and Zettlemoyer 2013, TACL



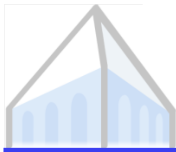
NLVR, Suhr et al. 2017, ACL

Neural Approaches



Sequence-to-Sequence Models

- Same methods from NMT! Encode input with an RNN, decode LF token-by-token
- Training: maximize log likelihood of gold LF conditioned on input utterance
- Can apply techniques like attention, beam search, etc.
- Problems:
  - Out-of-vocabulary terms, e.g., proper names (also a problem in MT)
  - No longer a clear divide between lexical and compositional semantics
  - No guarantee of syntactic validity or executability



Attending, Pointing, and Copying

Encoding of the source sentence.

Target sentence (output)

λ x . flight (x) …

Encoder RNN

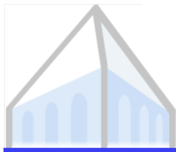
Decoder RNN

flights to new york

<START> λ x . flight (x …

Source sentence (input)

Slides from John DeNero / Philip Koehn



Attending, Pointing, and Copying

Attention output

Attention distribution

Attention scores

Encoder RNN

$\langle s_0 \dots s_3 \rangle$

$\mathbf{h}' = \frac{1}{3}\sum_{i=0}^{3} s_i \mathbf{x}_i \rightarrow \hat{y}_6$

NYC

Queries $\langle \mathbf{h}_0 \dots \mathbf{h}_5 \rangle$

Keys $\langle \mathbf{x}_0 \dots \mathbf{x}_3 \rangle$

Values $\langle \mathbf{x}_0 \dots \mathbf{x}_3 \rangle$

Generating "NYC" from the vocabulary:

$$p(y = \texttt{NYC}) \propto \exp(f(\mathbf{h}', \mathbf{h}_{-1})\mathbf{W}_{\texttt{NYC}} + \mathbf{b}_{\texttt{NYC}})$$

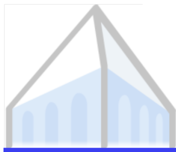
$\langle \mathbf{x}_0 \dots \mathbf{x}_3 \rangle$ ... $\langle \mathbf{h}_0 \dots \mathbf{h}_5 \rangle$

Learn a bias and embedding specifically for NYC

flights to new york

) AND to (x ,

Source sentence (input)



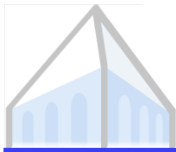
Attending, Pointing, and Copying

Queries $\langle \mathbf{h}_0 \ldots \mathbf{h}_5 \rangle$

Keys $\langle \mathbf{x}_0 \ldots \mathbf{x}_3 \rangle$

Values $\langle \mathbf{x}_0 \ldots \mathbf{x}_3 \rangle$

Slides from John DeNero / Philip Koehn



Attending, Pointing, and Copying

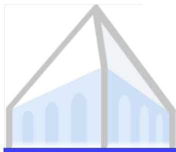
Generating "CITY_NAME0" from the vocabulary:

$$p(y = \texttt{CN0}) \propto \exp(f(\mathbf{h}', \mathbf{h}_{-1})\mathbf{W}_{\texttt{CN0}} + \mathbf{b}_{\texttt{CN0}})$$

Anonymization:
CITY_NAME0 = NYC

Learn a bias and embedding specifically for the 0th city name

Slides from John DeNero / Philip Koehn



Attending, Pointing, and Copying

Attention output

Copying “CITY_NAME0” from the input:

$p(y = x_2) \propto \exp(s_2)$

$\langle s_0 \dots s_3 \rangle$

CITY_NAME0

$\hat{y}_6$

ueries	$\langle \mathbf{h}_0 \dots \mathbf{h}_5 \rangle$
Keys	$\langle \mathbf{x}_0 \dots \mathbf{x}_3 \rangle$
Values	$\langle \mathbf{x}_0 \dots \mathbf{x}_3 \rangle$

Attention distribution

Attention scores

Encoder RNN

$\langle \mathbf{x}_0 \dots \mathbf{x}_3 \rangle$ … $\langle \mathbf{h}_0 \dots \mathbf{h}_5 \rangle$

flights to CITY_NAME0) AND to (x ,

Source sentence (input)

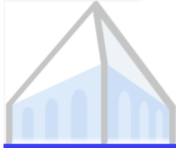
Anonymization:
CITY_NAME0 = NYC

Slides from John DeNero / Philip Koehn

Intrinsic Structure

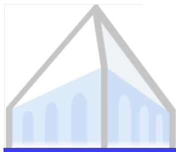
Constraints

- With token-by-token decoding, we lose the benefit of generating from a grammar
  - Our network now needs to (implicitly) learn the grammar from data
  - No guarantees that it will generate executable code
    - Syntax
    - Semantics
- How can we take advantage of this underlying structure?



Rejection Sampling

- Generate a number of candidate(s) (e.g., via beam search)
- Execute candidates, ensuring it compiles and runs without an error
- Return the highest-probability candidate that executes
- Could be very inefficient, especially because it requires running code at inference time



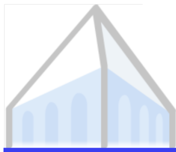
Intermediate Logical Forms

- Design an intermediate representation that implicitly captures structural dependencies in the code
- Generation in this output space reduces the need for the network to learn particular dependencies

```
SQL:     SELECT people.name FROM people JOIN films ON people.id = film.person_id
         WHERE films.id = 5
SQL^UF:  SELECT people.name UF WHERE films.id = 5
```

- However:
  - Cannot capture full expressivity of target language
  - Requires manual engineering of intermediate language, and deterministic mapping to / from the target language

E.g., Guo et al. 2019, ACL



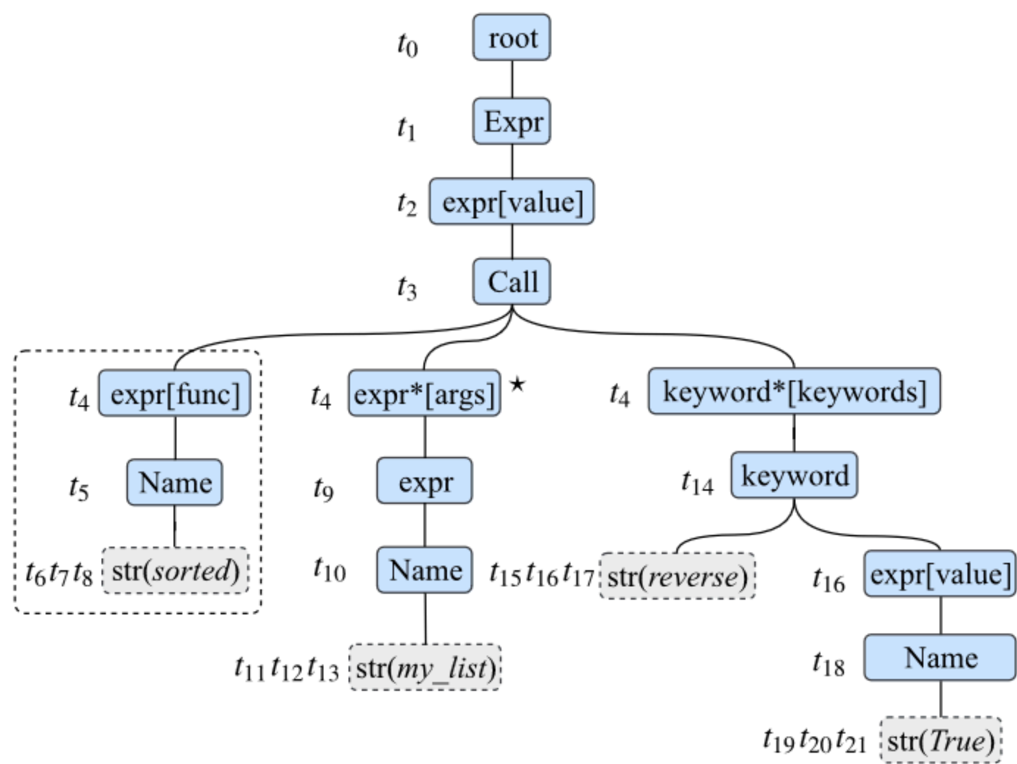
Constrained Decoding

- Generate actions that construct the AST that underlies the target code rather than the code itself
- Output space includes two types of actions:
  - ApplyRule *r* — apply production rule *r* to the current derivation tree
  - GenerateToken *t* — generate a variable terminal *t*
- Tokens *t* in sequence comprise the surface form of the code
- The current derivation tree constrains the set of rules *r* that can be applied and tokens *t* that can be generated
- At decoding time, simply mask out rules and tokens that cannot be generated

E.g., Yin and Neubig 2017, ACL

Constrained Decoding

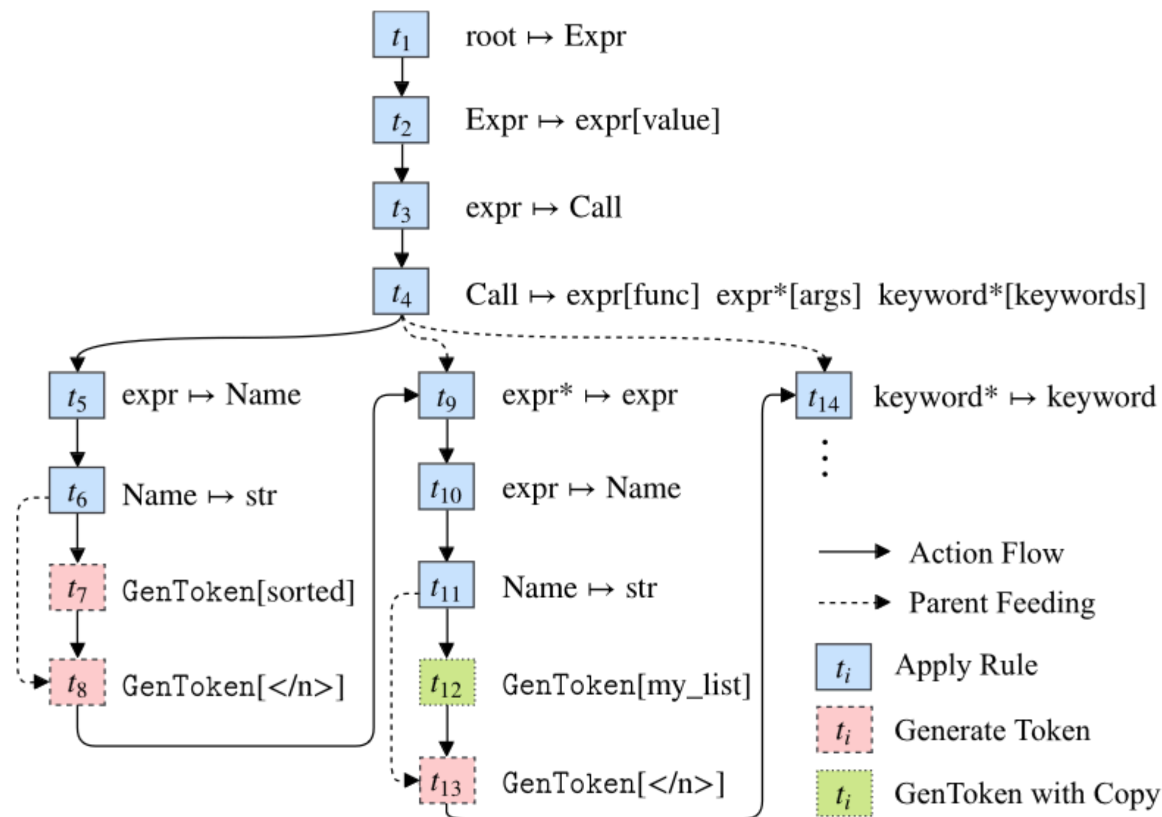
Generated AST



(a)

Input: `sort my_list in descending order`

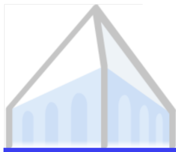
Production Rule Actions



(b)

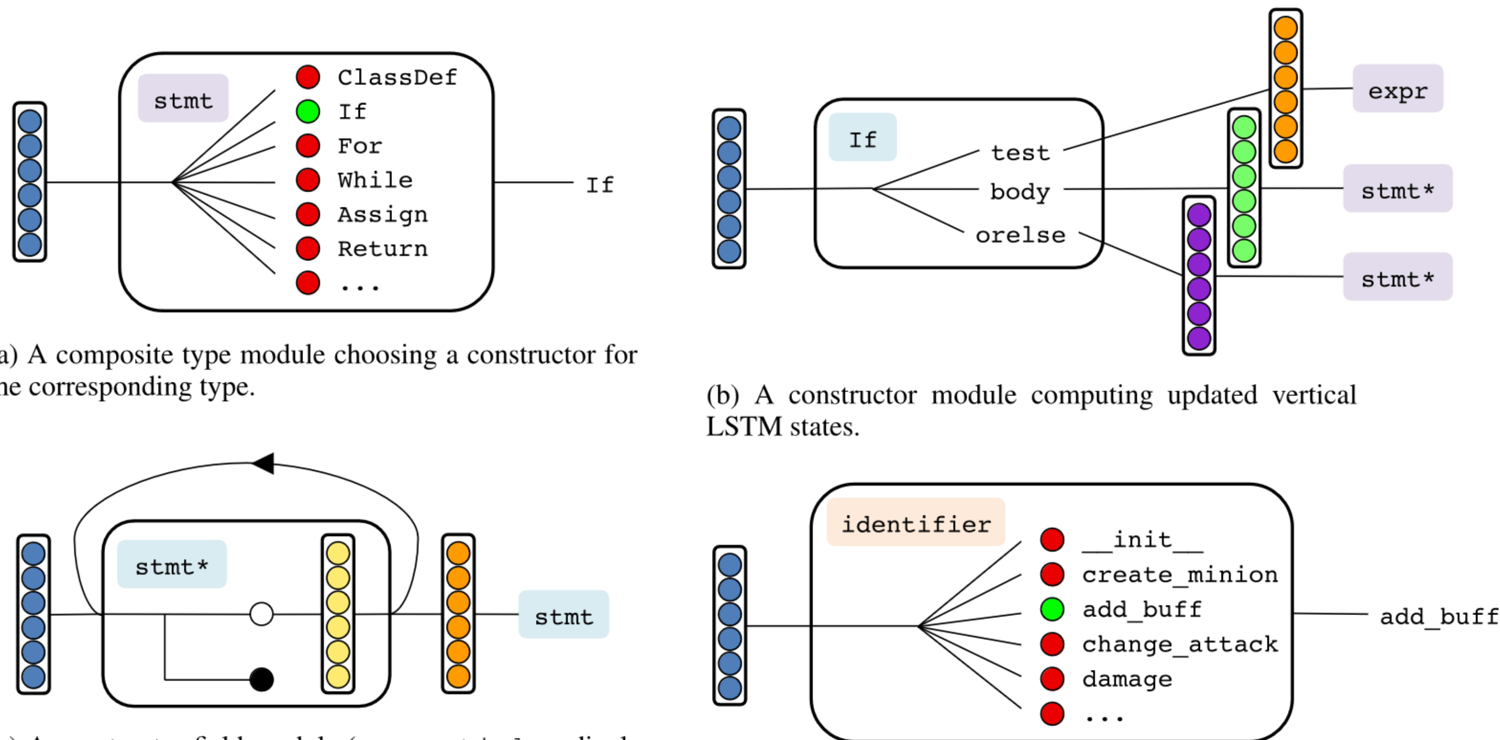
Code: `sorted(my_list, reverse=True)`

From Yin and Neubig 2017, ACL



Abstract Syntax Networks

- Generate AST, but learn and use custom decoders (“modules”) for different parts of the grammar



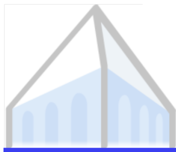
(a) A composite type module choosing a constructor for the corresponding type.

(b) A constructor module computing updated vertical LSTM states.

(c) A constructor field module (sequential cardinality) generating children to populate the field. At each step, the module decides whether to generate a child and continue (white circle) or stop (black circle).

(d) A primitive type module choosing a value from a closed list.

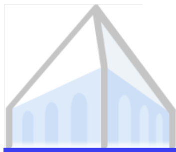
Rabinovich et al. 2017, ACL



Training at Scale

- With enough training data, modern neural architectures can capture underlying code structure without requiring injection of inductive biases
- It’s also easy to generate arbitrary amounts of code for training
- However, provides no guarantees
  - Without explicit copying mechanisms:
    - Possible for the model to learn biases in its vocabulary
    - No guarantees it will properly use new variables and functions
  - Ability to generalize to completely new programming languages and new structures?

General-Purpose
Code Generation



Code Generation

- Before: tasks with clear denotations
- What about general-purpose code generation?

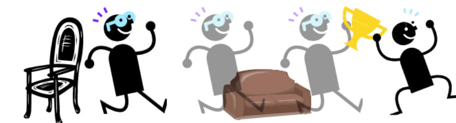
Year	City	Country	Nations
1896	Athens	Greece	14
1900	Paris	France	24
1904	St. Louis	USA	12
...	...	...	...
2004	Athens	Greece	201
2008	Beijing	China	204
2012	London	UK	204

x = Greece held its last Summer Olympics in which year?

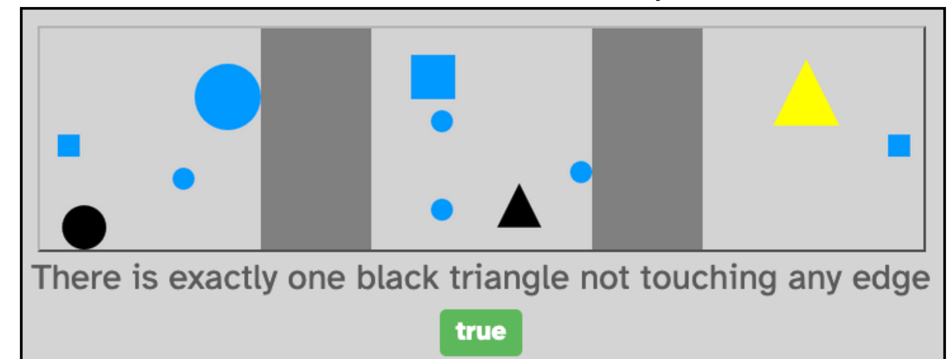
y = 2004

WikiTableQuestions, Pasupat and Liang 2015, ACL

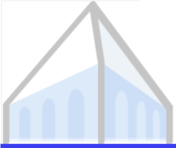
at the chair, move forward three steps past the sofa



Artzi and Zettlemoyer 2013, TACL



NLVR, Suhr et al. 2017, ACL



Code Generation

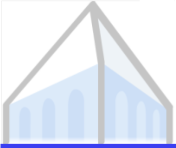
- Before: tasks with clear denotations
- What about general-purpose code generation?

```
"""Compute dates for today
and 1 month ago."""
import datetime

today =
datetime.date.today()
one_month_ago = today -
datetime.timedelta(days=30
)

print(today)
print(one_month_ago)
```

OpenAI Codex, 2021



Code Generation

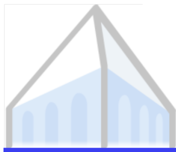
- Before: tasks with clear denotations
- What about general-purpose code generation?
  - Denotation: program output?

```
"""Compute dates for today
and 1 month ago."""
import datetime

today =
datetime.date.today()
one_month_ago = today -
datetime.timedelta(days=30
)

print(today)
print(one_month_ago)
```

OpenAI Codex, 2021



Code Generation

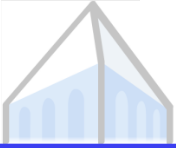
- Before: tasks with clear denotations
- What about general-purpose code generation?
  - Denotation: program output?
  - Less alignment between NL and LF

```
"""Compute dates for today
and 1 month ago."""
import datetime

today =
datetime.date.today()
one_month_ago = today -
datetime.timedelta(days=30
)

print(today)
print(one_month_ago)
```

OpenAI Codex, 2021

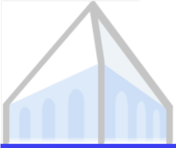


Code Generation

- Before: tasks with clear denotations
- What about general-purpose code generation?
  - Denotation: program output?
  - Less alignment between NL and LF
- What is a "denotation" isn't always clear…

```
/* Increment the score by 1
point, every 500ms. */
var scoreIncrement =
setInterval(function() {
  score++;
  scoreDisplay.innerHTML =
'Score: ' + score;
}, 500);
```

OpenAI Codex, 2021



Evaluation

- Code doesn’t always produce a single, evaluable output
- Instead: write test cases, report pass@k
- Labor-intensive: requires programming expertise for annotation (HumanEval only contains 164 problems)

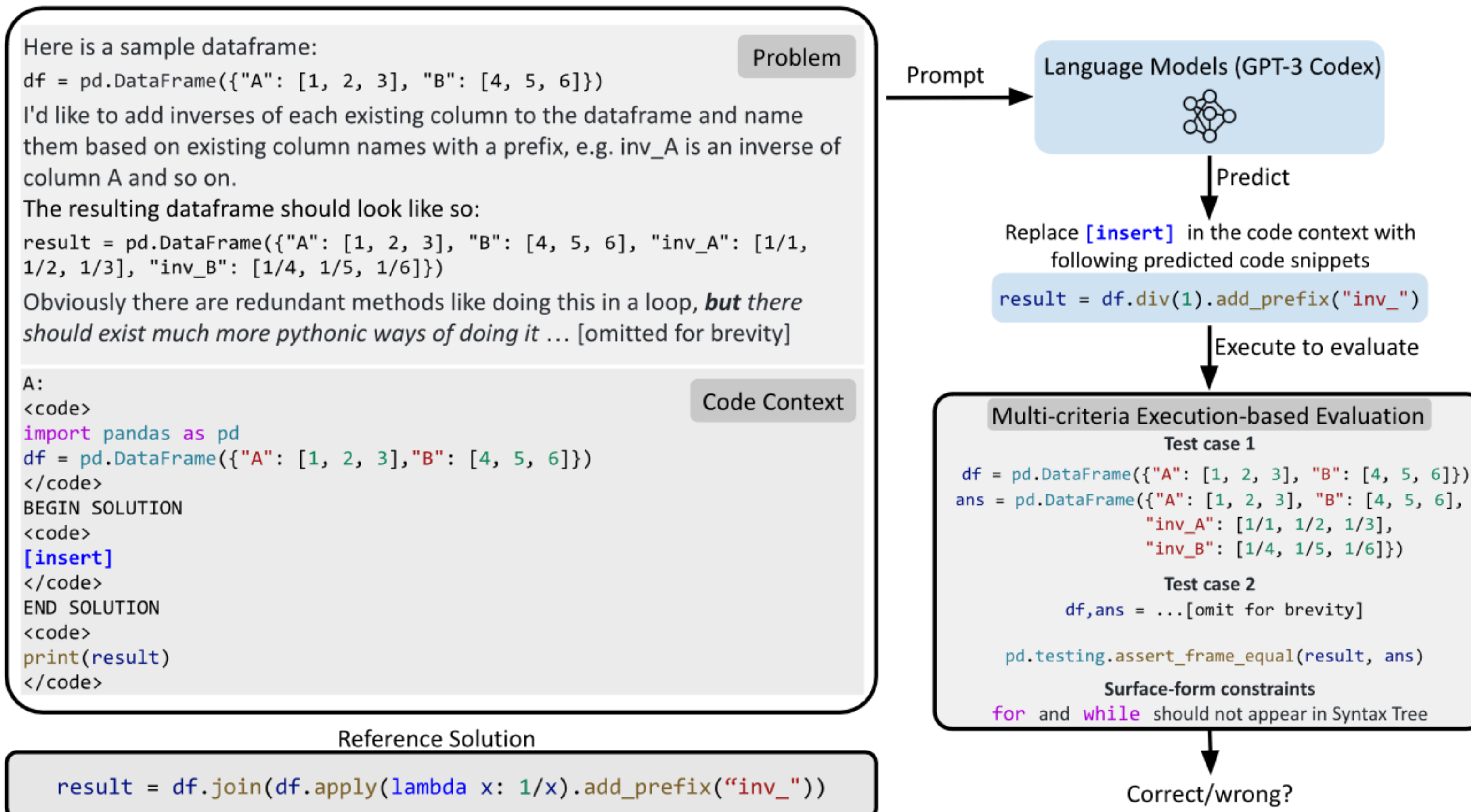
```
def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.

    Examples
    solution([5, 8, 7, 1]) ==>12
    solution([3, 3, 3, 3, 3]) ==>9
    solution([30, 13, 24, 321]) ==>0
    """
    return sum(lst[i] for i in range(0,len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

HumanEval, Chen et al. 2021

Evaluation

- Any automated benchmark has to focus on a subset of problems
- Going beyond solving programming puzzles

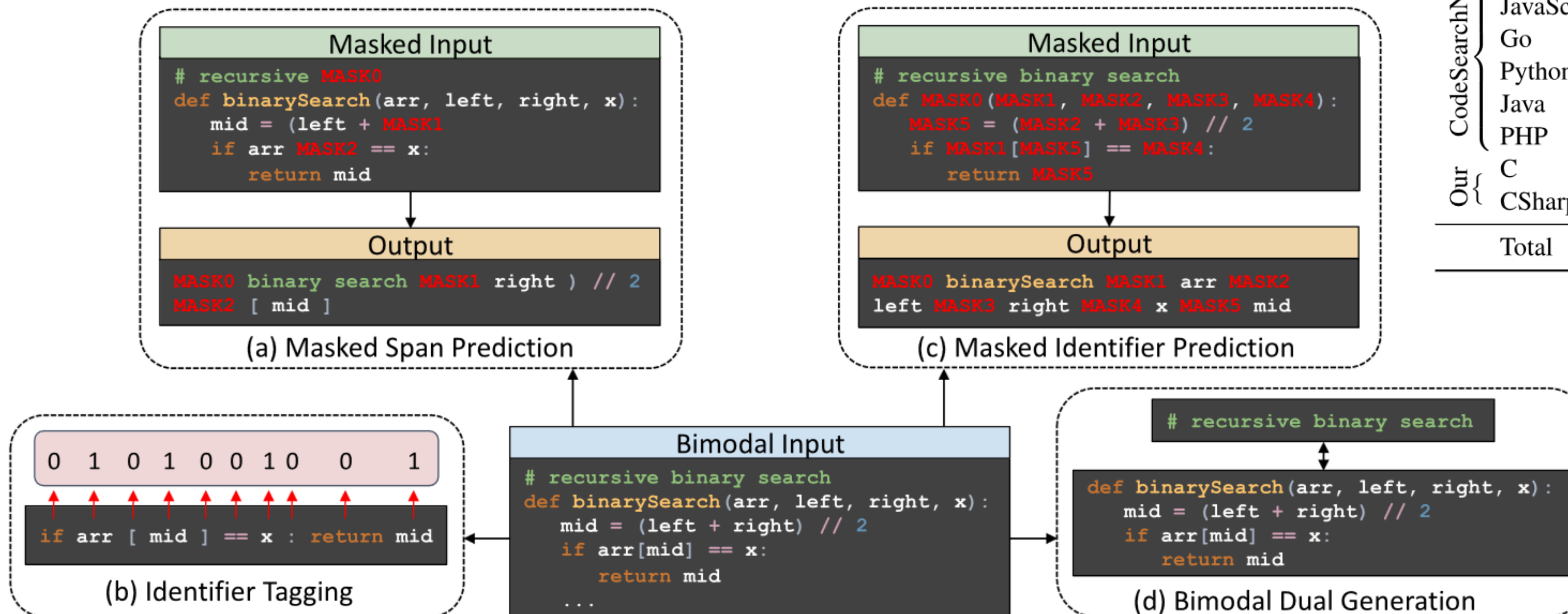


DS-1000, Lai et al. 2022

- Sample real problems from StackOverflow
- Collect reference solutions and setting up environment for testing
- Expert-written test cases
- Evaluate adherence to surface form constraints (e.g., that a library must be used)

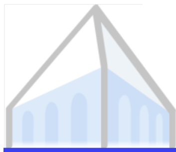
Approaches

- Multi-task learning: masking, tagging, generation
- Train on a large amount of code, some annotated with natural language



	PLs	W/ NL	W/o NL	Identifier
CodeSearchNet	Ruby	49,009	110,551	32.08%
	JavaScript	125,166	1,717,933	19.82%
	Go	319,132	379,103	19.32%
	Python	453,772	657,030	30.02%
	Java	457,381	1,070,271	25.76%
	PHP	525,357	398,058	23.44%
Our	C	1M	-	24.94%
	CSharp	228,496	856,375	27.85%
	Total	3,158,313	5,189,321	8,347,634

CodeT5, Wang et al. 2021



Automated Software Development?

MetaGPT Agents Collaboration with Developing SOP

Planning

Requirement Analysis

Architectural Design

System Design

Coding

Testing

Human Developing SOP

Product Manager — Requirement document

Architect — System design

Project Manager — Tasks

Engineer — Code

QA Engineer — Acceptance Check

Meta Programming

1/5 Define

2/5 Design

3/5 Plan&Code

4/5 Check

Human interaction

One-line requirement

Write a classic and simple Flappy Bird game.

Boss makes acceptance check and payment

Pretty good ! I can directly use the interface and keyboard to play Flappy Bird.

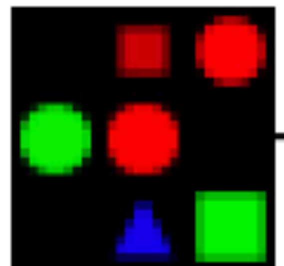
MetaGPT, Hong et al. 2023

Modularity

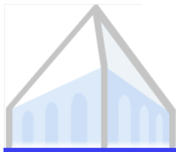
Neural Module Networks

- Task: visual question answering
  - Cast it as a semantic parsing task
  - What is a denotation?
- Tie predicates in LF to composable neural modules

“Is there a red shape above a circle?”

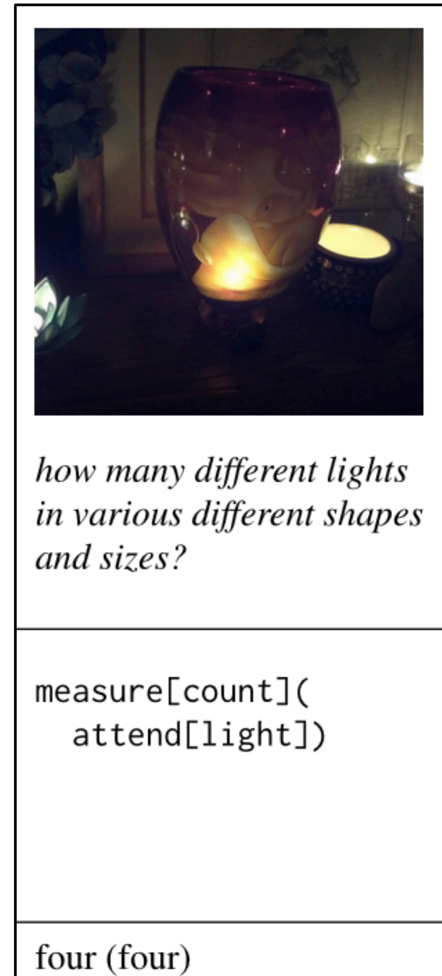


Andreas et al. 2015



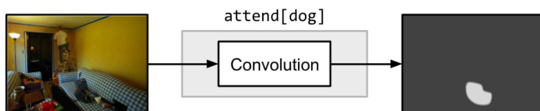
Neural Module Networks

- Determine layout from sentence
  - Option 1: deterministic layouts — requires gold annotation
    - Get dependency parse for input question
    - Construct layout of modules given this parse
  - Option 2: latent layouts — requires RL
- Compose modules and run inference / training (end-to-end)



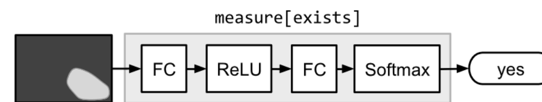
Attention

attend : *Image* → *Attention*



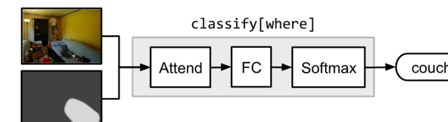
Measurement

measure : *Attention* → *Label*



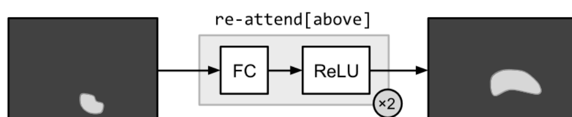
Classification

classify : *Image* × *Attention* → *Label*



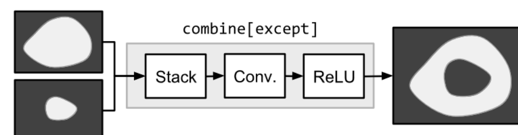
Re-attention

re-attend : *Attention* → *Attention*

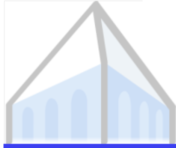


Combination

combine : *Attention* × *Attention* → *Attention*



Andreas et al. 2015, Hu et al. 2017



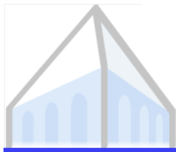
Neural Module Networks

- Benefits: interpretability and controllability
  - You know what modules are being used
  - You know how they are composed
  - You know the intermediate outputs of each module
- Problems
  - Requires formalizing the set of modules
  - Doesn’t work very well, empirically

Andreas et al. 2015, Hu et al. 2017

Code as a Reasoning Bottleneck

- Taking advantage of general-purpose code models
- Formal representation is given to us!
- Need very little paired data (use in-context learning)
- Still interpretable and controllable
- Some drawbacks:
  - Still requires choosing a few modules
  - Particular choice of in-context examples and modules can limit reasoning



Code as a Reasoning Bottleneck

Question:
Is the carriage to the right of a horse?

Codex
(Few–Shot Prompting)

In-Context Examples

```
# Image 1: On which side of
the picture is the rug?
img = open_image("Image1.jpg")
rug_pos_x, rug_pos_y =
get_pos(img, "rug")
if rug_pos_x < (LEFT+RIGHT)/2:
        answer = "left"
else:
        answer = "right”
. . .
```

Code Generation

```
horse_exists = query(img, "Is there
a horse?")
answer = "no”

if horse_exists == "yes":
    carriage_pos_x,carriage_pos_y =
      get_pos(img,"carriage")
    horse_pos_x, horse_pos_y =
      get_pos(img, "horse")
    if carriage_pos_x > horse_pos_x:
      answer = "yes”’
```

Execute Code

query(img,“Is there a horse?”)

Captions:
1. *'a police horse pulled by a fire policeman in a wagon',*
2. *'man riding a horse drawn carriage pulling horse next to a officer', ...*

returns “yes”

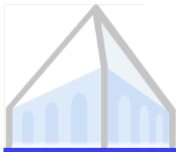
get_pos(img,“carriage”) *get_pos(img,“horse”)*

returns 5, 11 returns 12, 11

carriage_pos_x < horse_pos_x

Answer:
No

VisProg, Gupta and Kembhavi 2023, CVPR
Subramanian et al. 2023, ACL
ViperGPT, Surís et al. 2023



Code as a Reasoning Bottleneck

IMAGE:

Prediction: IMAGE0

Instruction: Replace desert with lush green grass

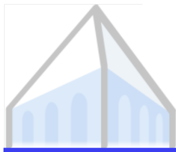
Program:

```
OBJ0=Seg(image=IMAGE)
OBJ1=Select(image=IMAGE, object=OBJ0, query='desert', category=None)
IMAGE0=Replace(image=IMAGE, object=OBJ1, prompt='lush green grass')
RESULT=IMAGE0
```

VisProg, Gupta and Kembhavi 2023, CVPR

Subramanian et al. 2023, ACL

ViperGPT, Surís et al. 2023



Code as a Reasoning Bottleneck

Query: Return the two kids that are furthest from the woman right before she hugs the girl

```
def execute_command(video):
    video_segment = VideoSegment(video)
    hug_detected = False
    for i, frame in enumerate(video_segment.frame_iterator()):
        if frame.exists("woman") and frame.exists("girl") and \
                frame.simple_query("Is the woman hugging the girl?") == "yes":
            hug_detected = True
            break
    if hug_detected:
        index_frame = i - 1
    frame_of_interest = ImagePatch(video_segment, index_frame)
    woman_patches = frame_of_interest.find("woman")
    woman_patch = woman_patches[0]
    kid_patches = frame_of_interest.find("kid")
    kid_patches.sort(key=lambda kid: distance(kid, woman_patch))
    kid_patch_1 = kid_patches[-1]
    kid_patch_2 = kid_patches[-2]
    return [kid_patch_1, kid_patch_2]
```

VisProg, Gupta and Kembhavi 2023, CVPR
Subramanian et al. 2023, ACL
ViperGPT, Surís et al. 2023