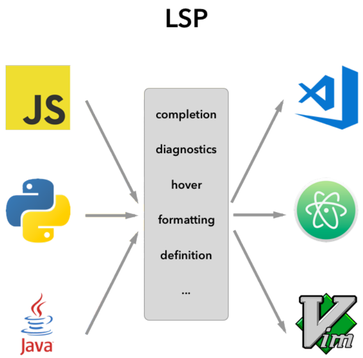
Syntactic and Semantic Control of Large Language Models via Sequential Monte Carlo

João Loula*[1] Benjamin LeBrun*[5] Li Du*[6]
Ben Lipkin[1] Clemente Pasti[2] Gabriel Grand[1] Tianyu Liu[2]
Yahya Emara[2] Marjorie Freedman[8] Jason Eisner[6] Ryan Cotterell[2]
Vikash Mansinghka‡[1] Alexander K. Lew‡[1,7] Tim Vieira‡[2] Timothy J. O'Donnell‡[3,4,5]
[1]MIT [2]ETH Zürich [3]McGill [4]Canada CIFAR AI Chair [5]Mila [6]Johns Hopkins [7]Yale [8]ISI
*co-first authorship, ‡co-senior authorship

Many signals we can use to guide LMs

Many signals we can use to guide LMs

- Code:
  - Static analysis




```
def signal(context):
   server = LSPDiagnosticServer(“python”)
   diagnostics = server.get_diagnostics(code)
   errors = [d for d in diagnostics
             if d["severity"] == 1]
       if errors:
           return 0
       return 1
```

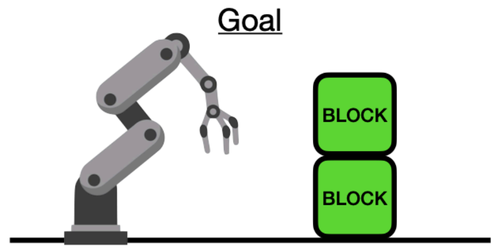
Many signals we can use to guide LMs

- Code:
  - Static analysis
  - Run tests check for errors


```
def signal(context):
   test_code = """import torch
   list_of_tensors = [torch.randn(3),
      torch.randn(3),
      torch.randn(3)]
   """
   try:
      exec(test_code + output)
      return 1
   except Exception as e:
      return 0
```

Many signals we can use to guide LMs

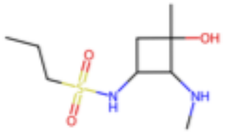
- Code:
  - Static analysis
  - Run tests check for errors
- Robotics: Simulate a plan and check goal

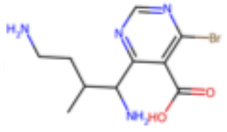



```
def signal(context):
  with open(“task.pddl”, “w”) as f:
    f.write(task.format(goal=output))
  proc = subprocess.run([
    “validate”, “domain.pddl”,
    “task.pddl”, “plan.pddl”])
  if proc.returncode ≠ 0:
    return 0
  return 1
```

Many signals we can use to guide LMs

- Code:
  - Static analysis
  - Run tests check for errors
- Robotics: Simulate a plan and check goal
- Chemistry: Compute molecular properties





```
def signal(context):
  state = SmilesParser.parse(context)
  for atom in state.mol.atoms:
    if not SmilesParser.hasCommonValence(
        atom, state):
      return 0
  return SmilesParser.validateKekulization(
      state.mol, state)
```

Many signals we can use to guide LMs

Signals vary along several dimensions

- Code:
  - Static analysis
  - Run tests check for errors
- Robotics: Simulate a plan and check goal
- Chemistry: Compute molecular properties

Many signals we can use to guide LMs

Signals vary along several dimensions

- Code:
  - Static analysis
  - Run tests check for errors
- Robotics: Simulate a plan and check goal
- Chemistry: Compute molecular properties

continuous/binary

Many signals we can use to guide LMs

Signals vary along several dimensions

- Code:
  - Static analysis
  - Run tests check for errors
- Robotics: Simulate a plan and check goal
- Chemistry: Compute molecular properties

continuous/binary

cheap/expensive

Many signals we can use to guide LMs

Signals vary along several dimensions

- Code:
  - Static analysis
  - Run tests check for errors
- Robotics: Simulate a plan and check goal
- Chemistry: Compute molecular properties

continuous/binary

cheap/expensive

token-by-token/sparse

Signals as *potentials* ϕ

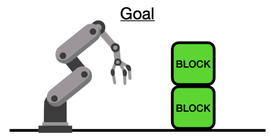
Generalizing verifiers, scorers, reward models…

$$\phi(x) : \text{context} \rightarrow \mathbb{R}_{\geq 0}$$

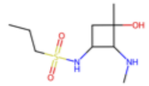

```
def signal(context):
    server = LSPDiagnosticServer(“python”)
    diagnostics = server.get_diagnostics(code)
    errors = [d for d in diagnostics if d["severity"] == 1]
        if errors:
            return 0
        return 0
```

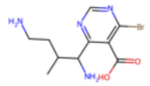
SciPy scikit-learn TensorFlow pandas matplotlib PyTorch

```
def signal(context):
    test_code = """import torch
    list_of_tensors = [torch.randn(3),
       torch.randn(3),
       torch.randn(3)]
    """
    try:
        exec(test_code + output)
        return 1
    except Exception as e:
        return 0
```




```
def signal(context):
    with open(“task.pddl”, “w”) as f:
        f.write(task.format(goal=output))
    proc = subprocess.run([
        “validate”, “domain.pddl”,
        “task.pddl”, “plan.pddl”])
    if proc.returncode ≠ 0:
        return 0
    return 1
```





```
def signal(context):
    state = SmilesParser.parse(context)
    for atom in state.mol.atoms:
        if not SmilesParser.hasCommonValence(
            atom, state):
            return 0
    return SmilesParser.validateKekulization(
        state.mol, state)
```

Key idea: controlled generation as *inference*

Key idea: controlled generation as *inference*

LM takes in a prompt

Prompt: How do I stack a list `"tensors"` of pytorch tensors?

Slide adapted from Alex Lew

Key idea: controlled generation as *inference*

LM takes in a prompt, acts as a *prior* p_{LM}

Prompt: How do I stack a list "tensors" of pytorch tensors?

$p_{LM}(x)$

tensors

torch.tensor(tensors)

for a list of tensors, we have to…

in python, we can stack…

sure, let me see…

…

torch.stack(*tensors)

torch.stack(tensors)

torch.tensor(*tensors, axis=1)

np.stack(tensors, axis=0)

torch.stack(tensors, dim=0)

all strings x

Slide adapted from Alex Lew

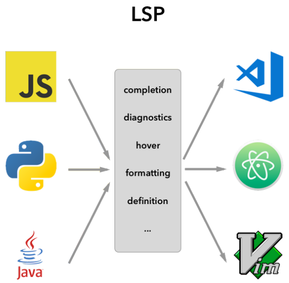
Key idea: controlled generation as *inference*

LM takes in a prompt, acts as a *prior* p_{LM}, and ϕs as likelihoods

Prompt: How do I stack a list "tensors" of pytorch tensors?

$$\propto p_{LM}(x)\phi_{linter}(x)$$

```
def ϕ_linter(context):
  server = LSPDiagnosticServer(“python”)
  diagnostics = server.get_diagnostics(code)
  errors = [d for d in diagnostics if
  d["severity"] == 1]
      if errors:
          return 0
      return 1
```



tensors

torch.tensor(tensors)

$\phi_{linter} = 0$
for a list of tensors, we have to…

$\phi_{linter} = 0$
in python, we can stack…

$\phi_{linter} = 0$
sure, let me see…

…

torch.stack(tensors)

torch.stack(*tensors)

np.stack(tensors, axis=0)

torch.tensor(*tensors, axis=1)

torch.stack(tensors, dim=0)

all strings x

Slide adapted from Alex Lew

Key idea: controlled generation as *inference*

LM takes in a prompt, acts as a *prior* p_{LM}, and ϕs as likelihoods

Prompt: How do I stack a list "tensors" of pytorch tensors?

$$\propto p_{LM}(x)\phi_{linter}(x)\phi_{test}(x)$$

```
def ϕ_test(context):
  test_code = """import torch
  list_of_tensors =
  [torch.randn(3), torch.randn(3),
   torch.randn(3)]"""
  try:
    exec(test_code + output)
    return 1
  except Exception as e:
    return 0
```

SciPy scikit-learn TensorFlow pandas matplotlib PyTorch

$\phi_{test} = 0$
tensors

$\phi_{test} = 0$
torch.tensor(tensors)

$\phi_{linter} = 0$
for a list of tensors, we have to…

$\phi_{linter} = 0$
in python, we can stack…

$\phi_{linter} = 0$
sure, let me see…

…

torch.stack(tensors)

torch.stack(*tensors)

$\phi_{test} = 0$
np.stack(tensors, axis=0)

$\phi_{test} = 0$
torch.tensor(*tensors, axis=1)

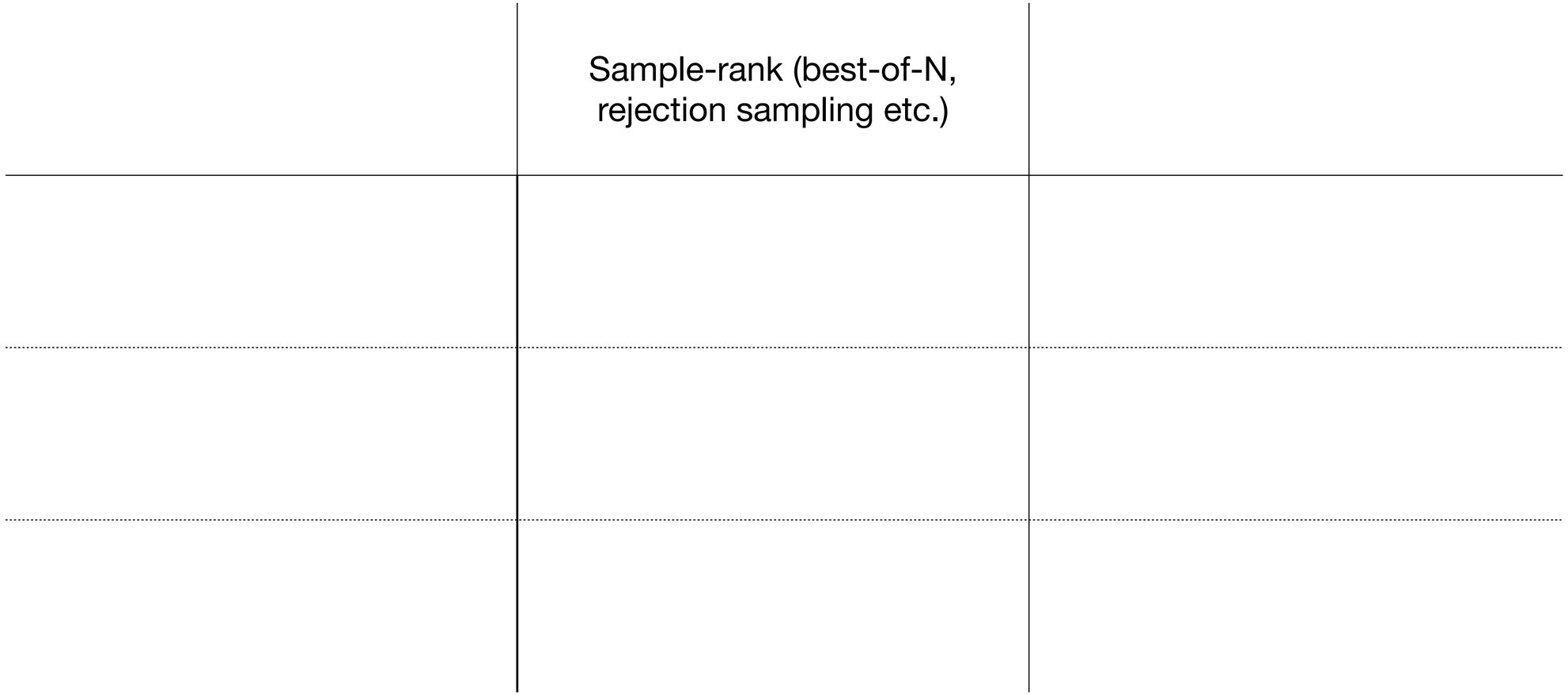
torch.stack(tensors, dim=0)

all strings x

Slide adapted from Alex Lew

Current approaches to controlled generation have major flaws

Current approaches to controlled generation have major flaws



Popular approach 1: Sample-rank

e.g. best-of-n with a reward model (Nakano et al., 2021; Krishna et al., 2022; Zhou et al., 2023; Gui et al., 2024; Mudgal et al., 2024; Ichihara et al., 2025), filtering with a verifier (Olausson et al., 2023; Chen et al. 2024; Lightman et al., 2024; Xin et al., 2024)

Popular approach 1: Sample-rank

Generations

```
a = torch.stack((tensors)
```

```
a = torch.tensor(tensors)
```

```
a = torch.stack(tensors)
```

e.g. best-of-n with a reward model (Nakano et al., 2021; Krishna et al., 2022; Zhou et al., 2023; Gui et al., 2024; Mudgal et al., 2024; Ichihara et al., 2025), filtering with a verifier (Olausson et al., 2023; Chen et al. 2024; Lightman et al., 2024; Xin et al., 2024)

Popular approach 1: Sample-rank

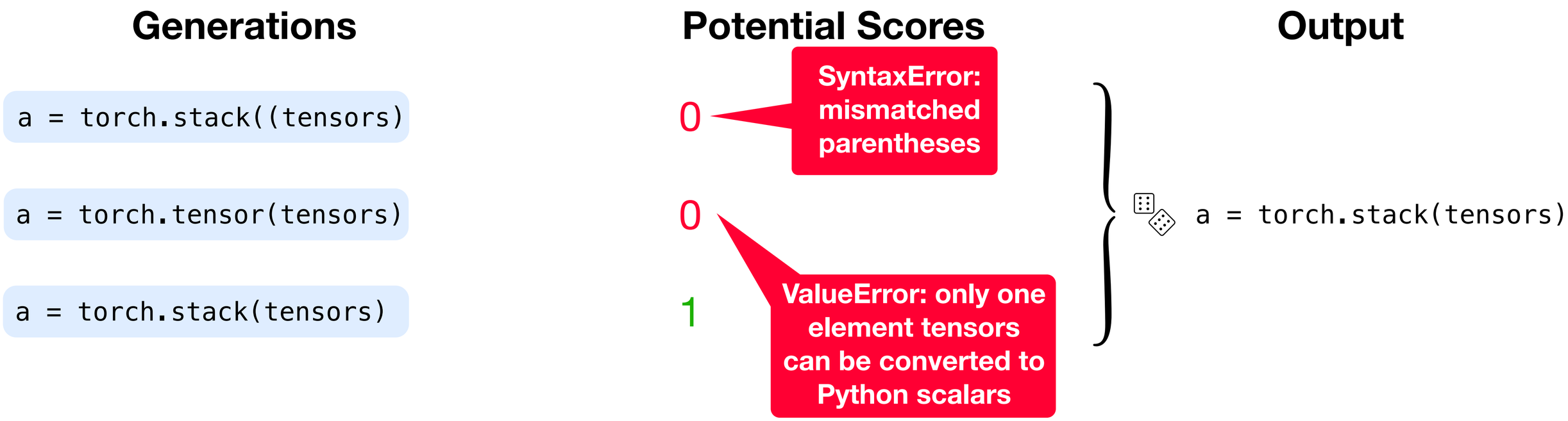
Generations	Potential Scores
`a = torch.stack((tensors)`	0
`a = torch.tensor(tensors)`	0
`a = torch.stack(tensors)`	1

SyntaxError: mismatched parentheses

ValueError: only one element tensors can be converted to Python scalars

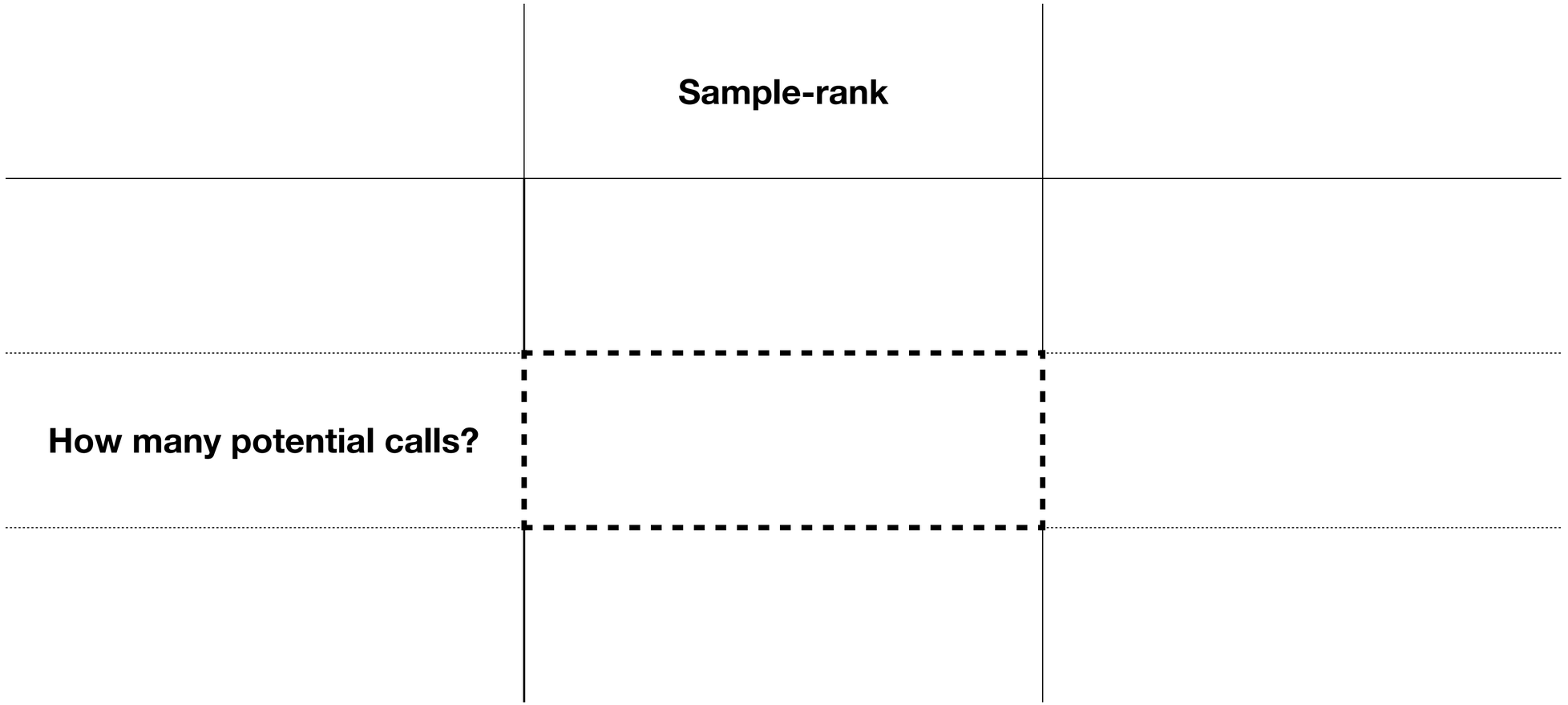
e.g. best-of-n with a reward model (Nakano et al., 2021; Krishna et al., 2022; Zhou et al., 2023; Gui et al., 2024; Mudgal et al., 2024; Ichihara et al., 2025), filtering with a verifier (Olausson et al., 2023; Chen et al. 2024; Lightman et al., 2024; Xin et al., 2024)

Popular approach 1: Sample-rank

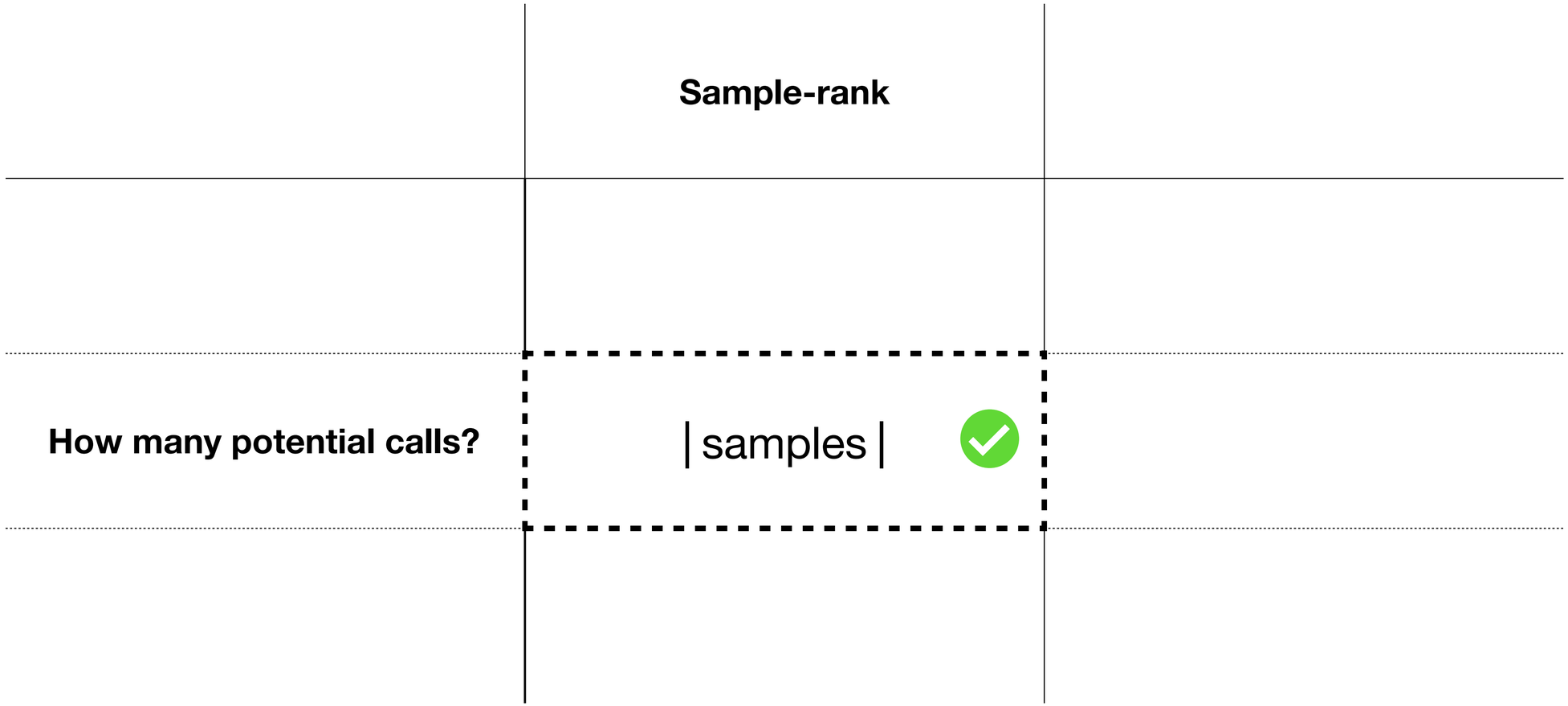


e.g. best-of-n with a reward model (Nakano et al., 2021; Krishna et al., 2022; Zhou et al., 2023; Gui et al., 2024; Mudgal et al., 2024; Ichihara et al., 2025), filtering with a verifier (Olausson et al., 2023; Chen et al. 2024; Lightman et al., 2024; Xin et al., 2024)

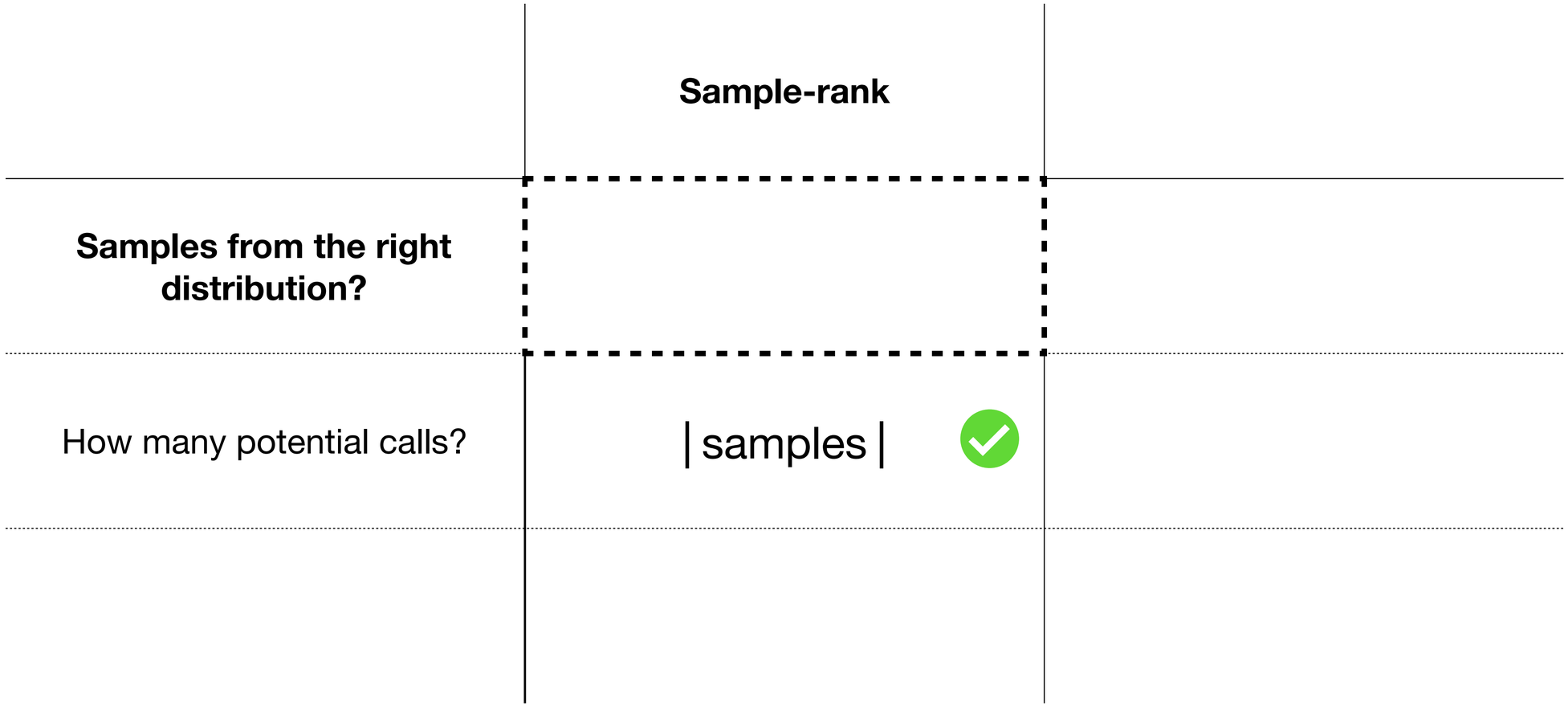
Current approaches to controlled generation have major flaws



Current approaches to controlled generation have major flaws



Current approaches to controlled generation have major flaws



Sample-rank samples from the correct distribution

Importance sampling tells us that…

Sample-rank samples from the correct distribution

Importance sampling tells us that…

$$p_{LM}(x)$$

Distribution we sample from

Sample-rank samples from the correct distribution

Importance sampling tells us that…

$$p_{LM}(x)\phi(x)$$

Distribution we sample from

Function we score by

Sample-rank samples from the correct distribution

Importance sampling tells us that…

Distribution we converge to → $g(x) \propto p_{LM}(x)\phi(x)$

Distribution we sample from

Function we score by

Current approaches to controlled generation have major flaws

	Sample-rank
Samples from the right distribution?	$g(x) \propto p_{LM}(x)\phi(x)$ ✓
How many potential calls?	$\lvert \text{samples} \rvert$ ✓

Current approaches to controlled generation have major flaws

	Sample-rank	
Samples from the right distribution?	$g(x) \propto p_{LM}(x)\phi(x)$ ✓	
How many potential calls?	$\lvert \text{samples} \rvert$ ✓	
How many LM calls?		

Sample-rank needs a lot of LM calls

$$g(x) \propto p_{LM}(x)\phi(x)$$

Sample-rank needs a lot of LM calls

$$g(x) \propto p_{LM}(x)\phi(x)$$

Required # LM samples: $e^{D_{KL}(g||p_{LM})}$

(Chatterjee and Diaconis, 2018)

Current approaches to controlled generation have major flaws

	Sample-rank
Samples from the right distribution?	$g(x) \propto p_{LM}(x)\phi(x)$ ✓
How many potential calls?	$\lvert \text{samples} \rvert$ ✓
How many LM calls?	$\approx \exp D_{KL}(g \lvert\lvert p_{LM})$ ✗

Current approaches to controlled generation have major flaws

	Sample-rank	**Locally Constrained Decoding (token masking, logit biasing etc.)**
Samples from the right distribution?	$g(x) \propto p_{LM}(x)\phi(x)$ ✓	
How many potential calls?	$\lvert \text{samples} \rvert$ ✓	
How many LM calls?	$\approx \exp D_{KL}(g \lvert\rvert p_{LM})$ ✗	

Popular approach 2: Locally Constrained Decoding

(e.g., OpenAI 2024; Shin et al., 2021; Scholak et al., 2021; Poesia et al., 2022; Willard & Louf, 2023; Moskal et al., 2024; Ugare et al., 2024)

Popular approach 2: Locally Constrained Decoding

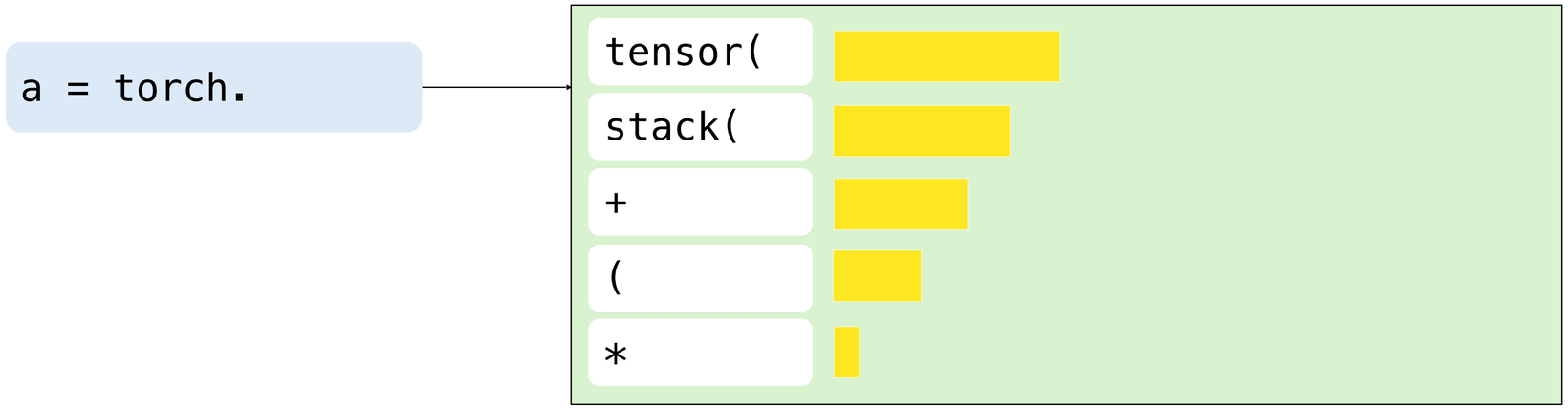
At each step:

```
a = torch.
```

(e.g., OpenAI 2024; Shin et al., 2021; Scholak et al., 2021; Poesia et al., 2022; Willard & Louf, 2023; Moskal et al., 2024; Ugare et al., 2024)

Popular approach 2: Locally Constrained Decoding

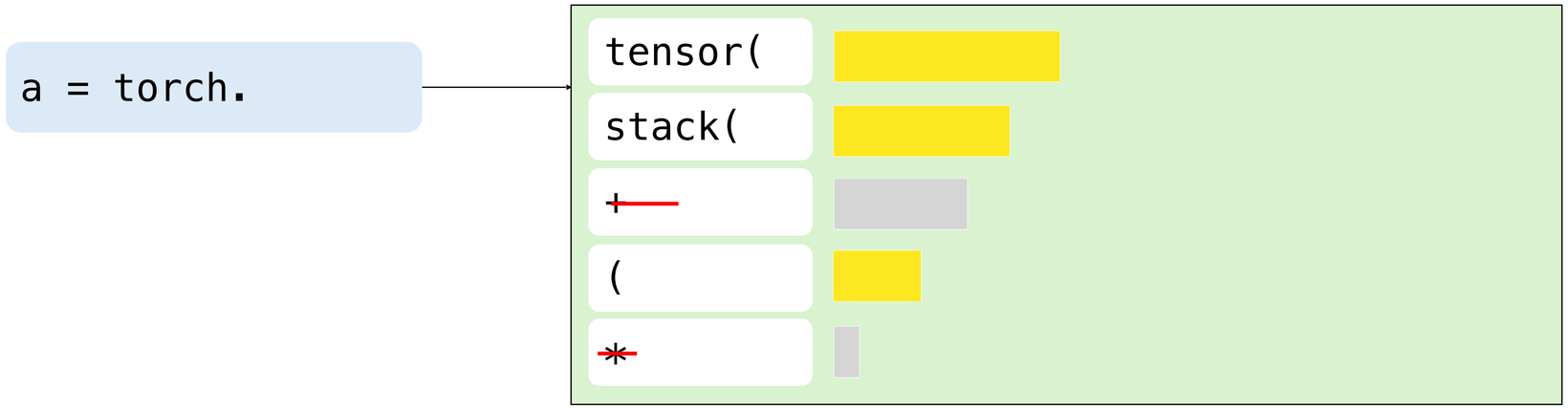
At each step: get next token probs



(e.g., OpenAI 2024; Shin et al., 2021; Scholak et al., 2021; Poesia et al., 2022; Willard & Louf, 2023; Moskal et al., 2024; Ugare et al., 2024)

Popular approach 2: Locally Constrained Decoding

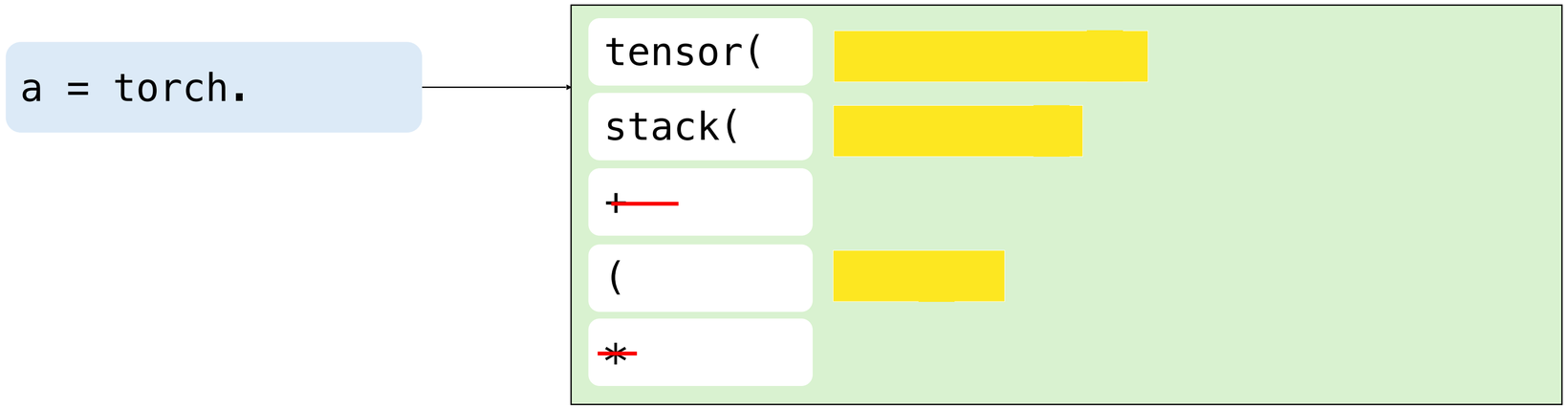
At each step: get next token probs, multiply potentials



(e.g., OpenAI 2024; Shin et al., 2021; Scholak et al., 2021; Poesia et al., 2022; Willard & Louf, 2023; Moskal et al., 2024; Ugare et al., 2024)

Popular approach 2: Locally Constrained Decoding

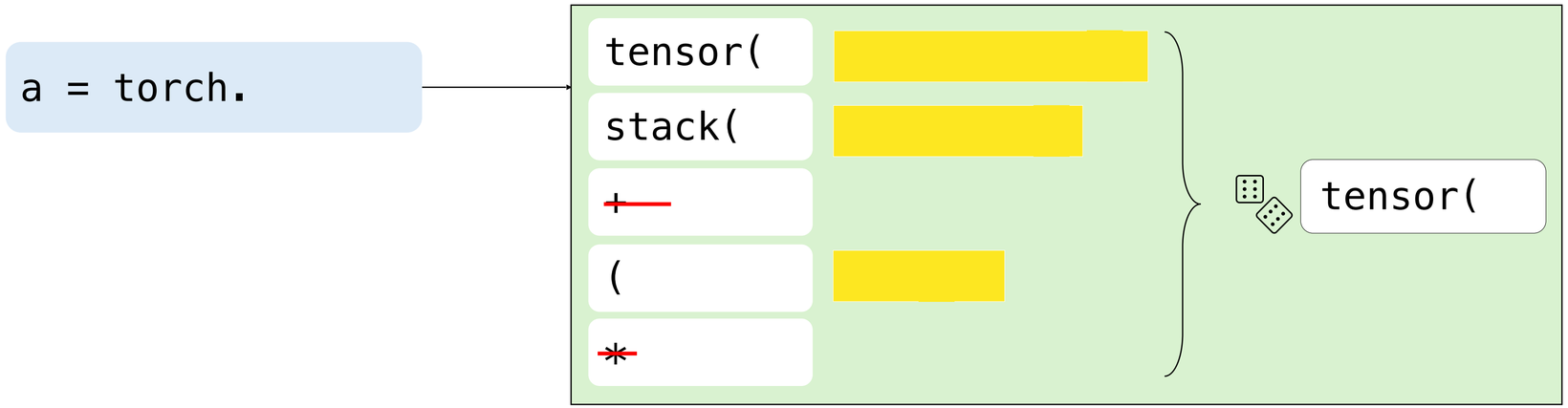
At each step: get next token probs, multiply potentials, normalize



(e.g., OpenAI 2024; Shin et al., 2021; Scholak et al., 2021; Poesia et al., 2022; Willard & Louf, 2023; Moskal et al., 2024; Ugare et al., 2024)

Popular approach 2: Locally Constrained Decoding

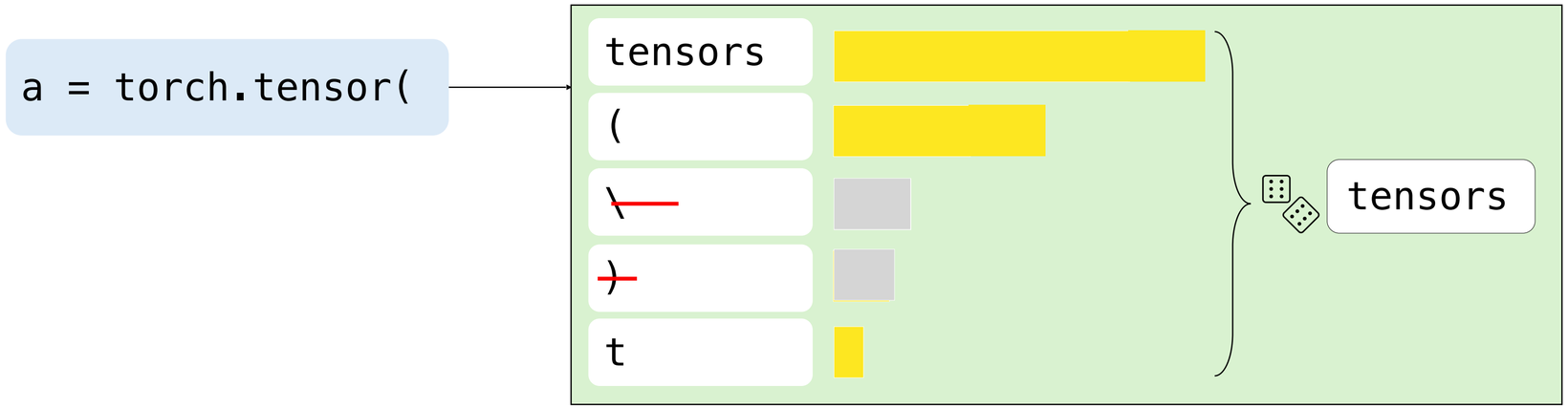
At each step: get next token probs, multiply potentials, normalize, sample



(e.g., OpenAI 2024; Shin et al., 2021; Scholak et al., 2021; Poesia et al., 2022; Willard & Louf, 2023; Moskal et al., 2024; Ugare et al., 2024)

Popular approach 2: Locally Constrained Decoding

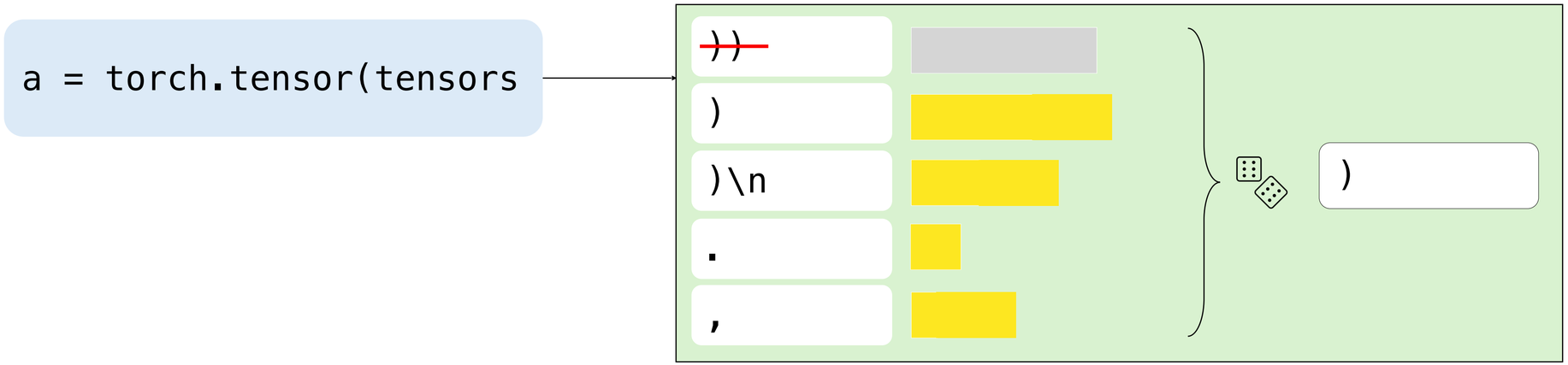
At each step: get next token probs, multiply potentials, normalize, sample



(e.g., OpenAI 2024; Shin et al., 2021; Scholak et al., 2021; Poesia et al., 2022; Willard & Louf, 2023; Moskal et al., 2024; Ugare et al., 2024)

Popular approach 2: Locally Constrained Decoding

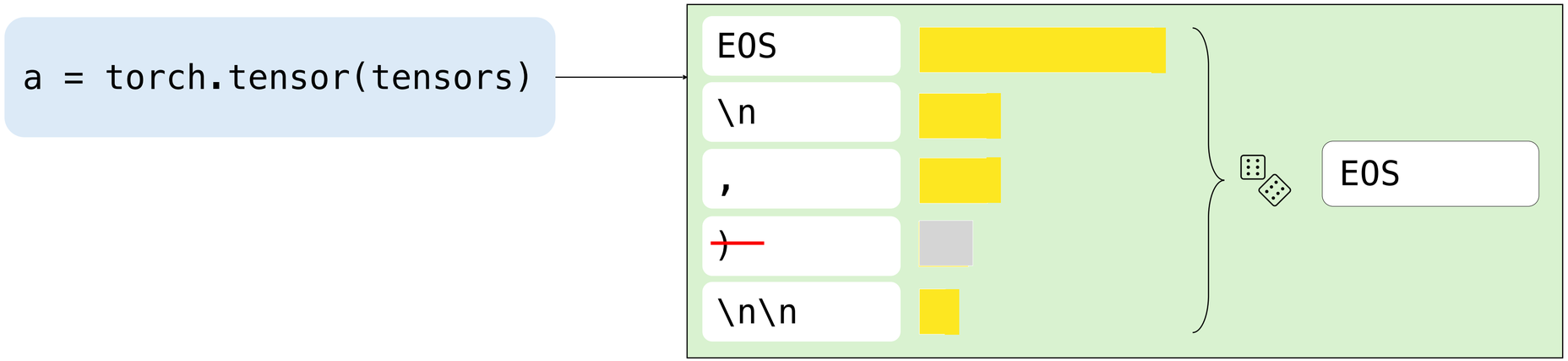
At each step: get next token probs, multiply potentials, normalize, sample



(e.g., OpenAI 2024; Shin et al., 2021; Scholak et al., 2021; Poesia et al., 2022; Willard & Louf, 2023; Moskal et al., 2024; Ugare et al., 2024)

Popular approach 2: Locally Constrained Decoding

At each step: get next token probs, multiply potentials, normalize, sample



(e.g., OpenAI 2024; Shin et al., 2021; Scholak et al., 2021; Poesia et al., 2022; Willard & Louf, 2023; Moskal et al., 2024; Ugare et al., 2024)

Current approaches to controlled generation have major flaws

	Sample-rank	**Locally-Constrained Decoding**
Samples from the right distribution?	$g(x) \propto p_{LM}(x)\phi(x)$ ✓	
How many potential calls?	$\lvert \text{samples} \rvert$ ✓	
How many LM calls?	$\approx \exp D_{KL}(g \lvert\lvert p_{LM})$ ✗	

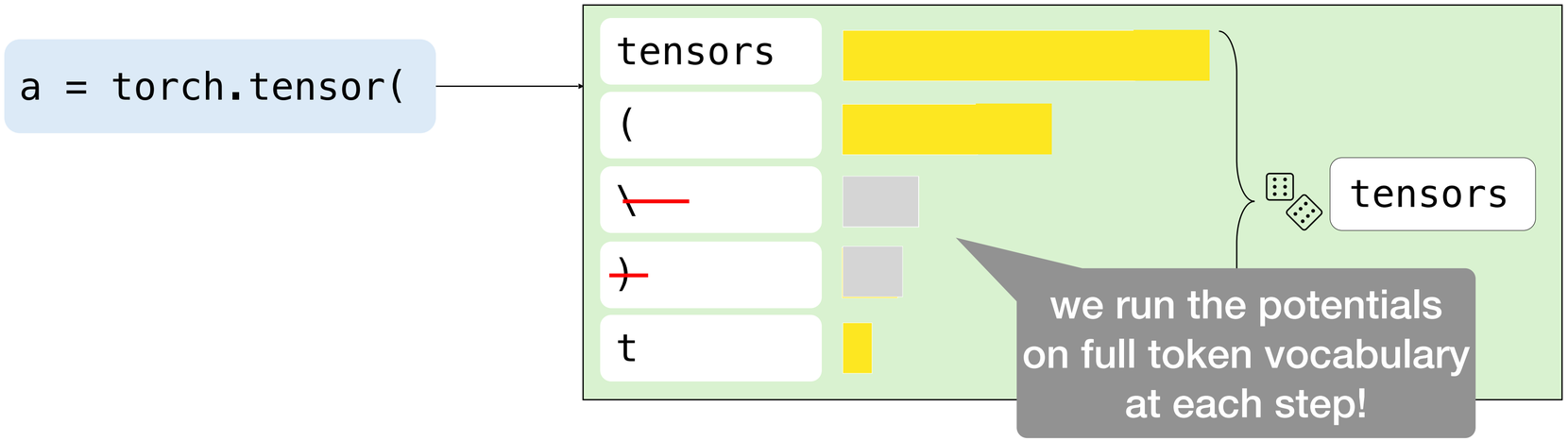
Current approaches to controlled generation have major flaws

	Sample-rank	**Locally-Constrained Decoding**
Samples from the right distribution?	$g(x) \propto p_{LM}(x)\phi(x)$ ✓	
How many potential calls?	$\lvert \text{samples} \rvert$ ✓	
How many LM calls?	$\approx \exp D_{KL}(g \lvert\lvert p_{LM})$ ✗	1 ✓

Current approaches to controlled generation have major flaws

	Sample-rank	**Locally-Constrained Decoding**
Samples from the right distribution?	$g(x) \propto p_{LM}(x)\phi(x)$ ✓	
How many potential calls?	$\lvert \text{samples} \rvert$ ✓	
How many LM calls?	$\approx \exp D_{KL}(g \lvert\rvert p_{LM})$ ✗	1 ✓

Locally Constrained Decoding calls potentials ~100k times per step



(e.g., Shin et al., 2021; Scholak et al., 2021; Poesia et al., 2022; Willard & Louf, 2023; Moskal et al., 2024; Ugare et al., 2024)

Current approaches to controlled generation have major flaws

	Sample-rank	**Locally-Constrained Decoding**
Samples from the right distribution?	$g(x) \propto p_{LM}(x)\phi(x)$ ✓	
How many potential calls?	$\lvert \text{samples} \rvert$ ✓	$\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$ ✗
How many LM calls?	$\approx \exp D_{KL}(g \lvert\rvert p_{LM})$ ✗	1 ✓

Current approaches to controlled generation have major flaws

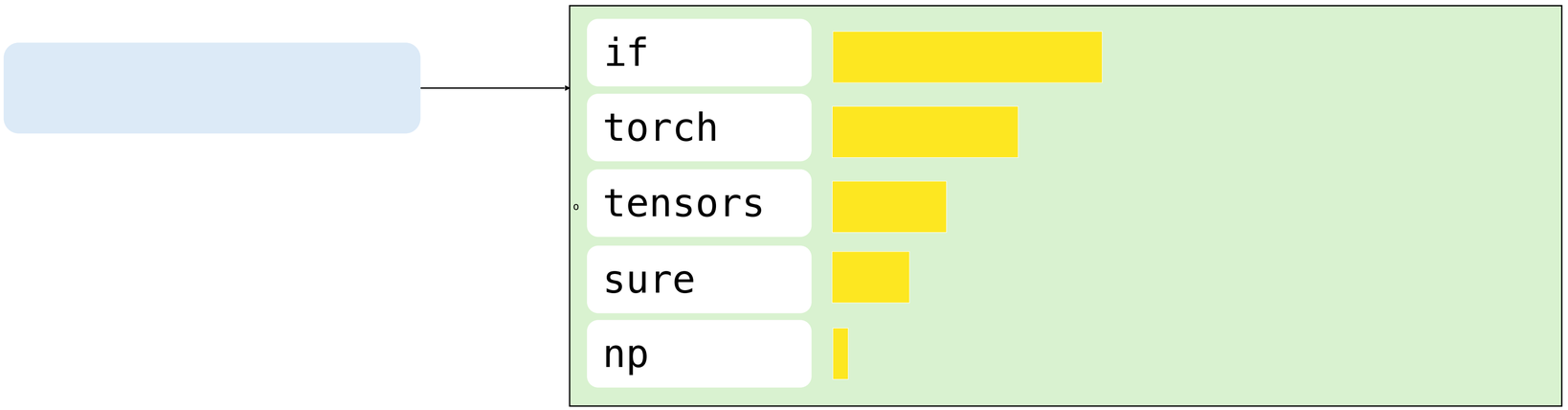
	Sample-rank	**Locally-Constrained Decoding**
Samples from the right distribution?	$g(x) \propto p_{LM}(x)\phi(x)$ ✓	
How many potential calls?	$\lvert \text{samples} \rvert$ ✓	$\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$ ✗
How many LM calls?	$\approx \exp D_{KL}(g \lvert\rvert p_{LM})$ ✗	1 ✓

Locally Constrained Decoding samples from the wrong distribution

Example: LM wants to write English instead of Python: “if you have a list of tensors in torch, you can stack it by…”

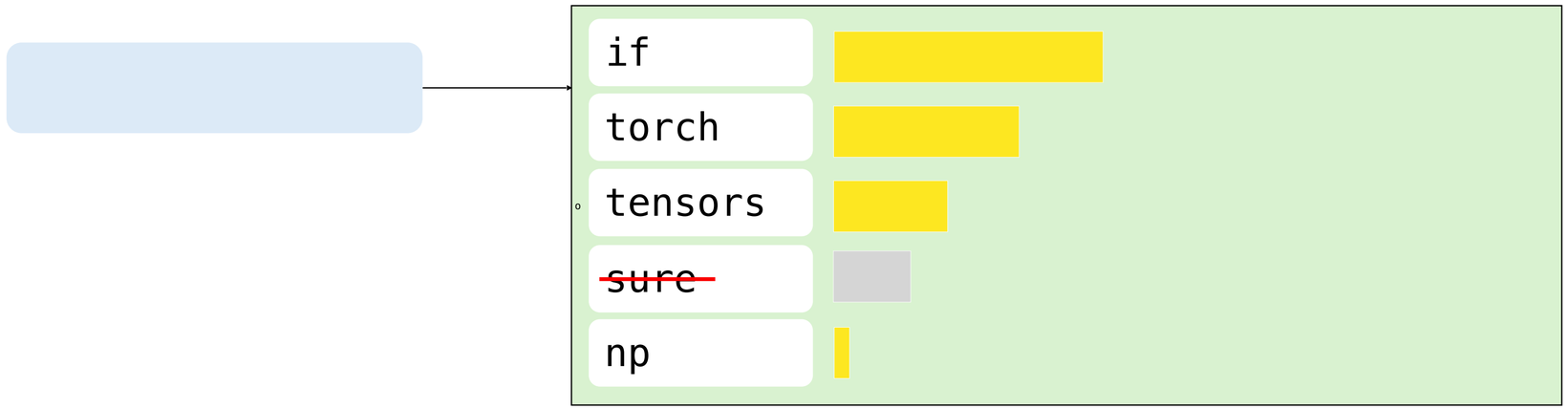
Locally Constrained Decoding samples from the wrong distribution

Example: LM wants to write English instead of Python: “if you have a list of tensors in torch, you can stack it by…”



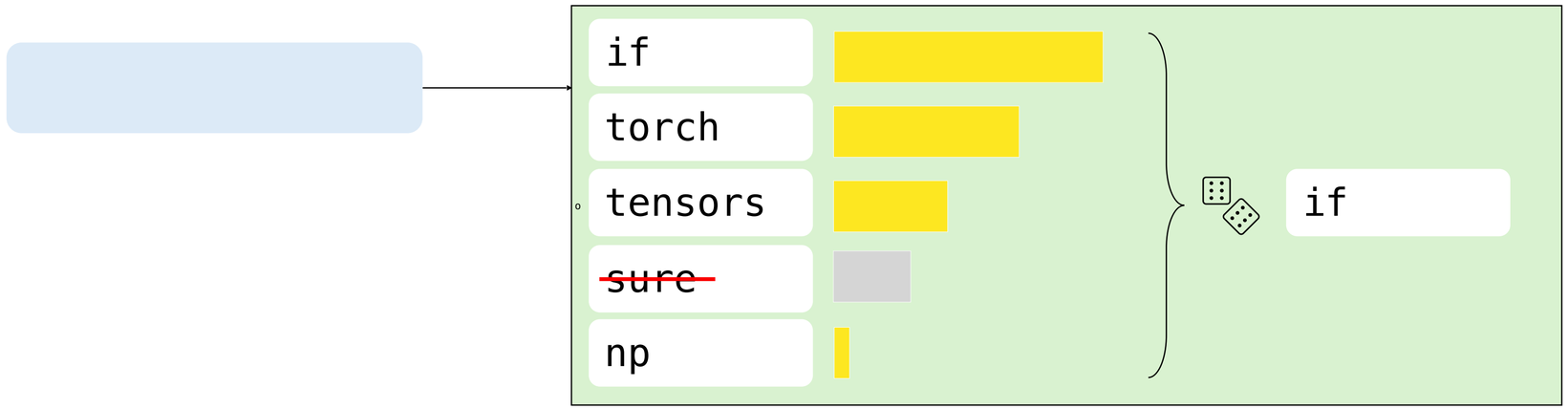
Locally Constrained Decoding samples from the wrong distribution

Example: LM wants to write English instead of Python: “if you have a list of tensors in torch, you can stack it by…”



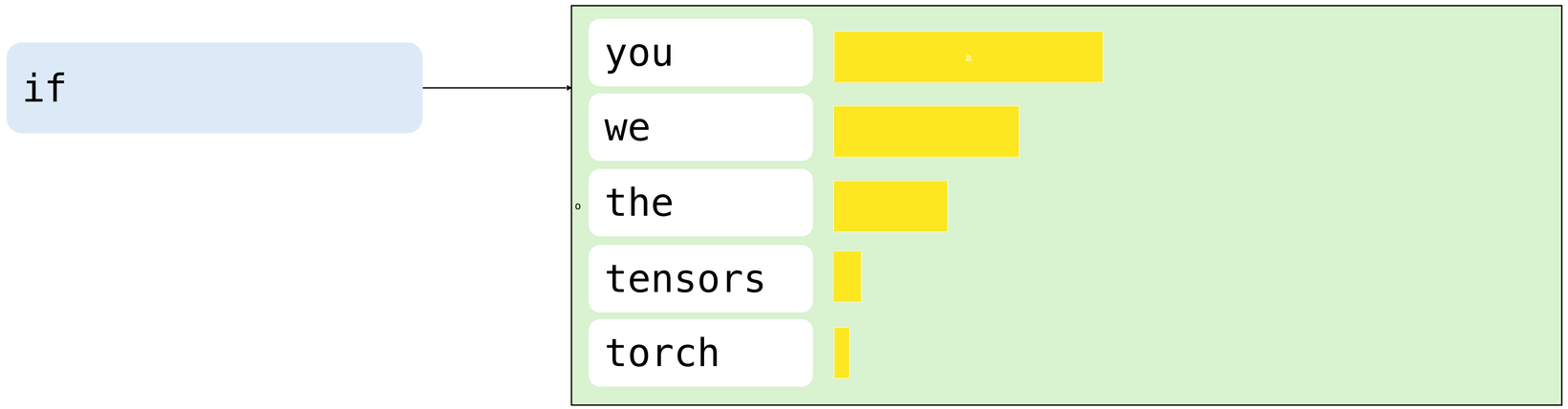
Locally Constrained Decoding samples from the wrong distribution

Example: LM wants to write English instead of Python: “if you have a list of tensors in torch, you can stack it by…”



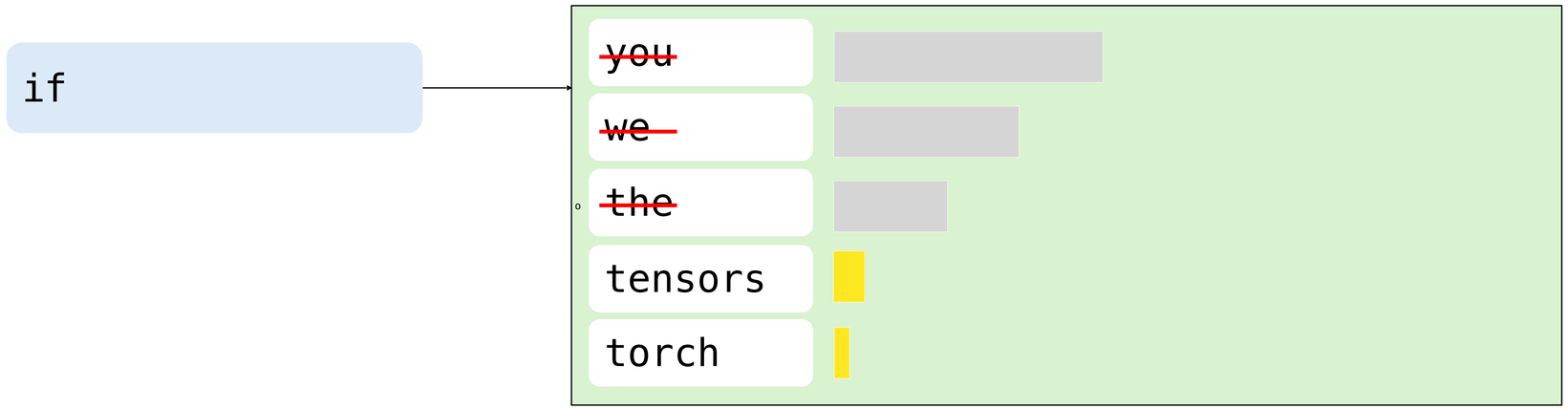
Locally Constrained Decoding samples from the wrong distribution

Example: LM wants to write English instead of Python: “if you have a list of tensors in torch, you can stack it by…”



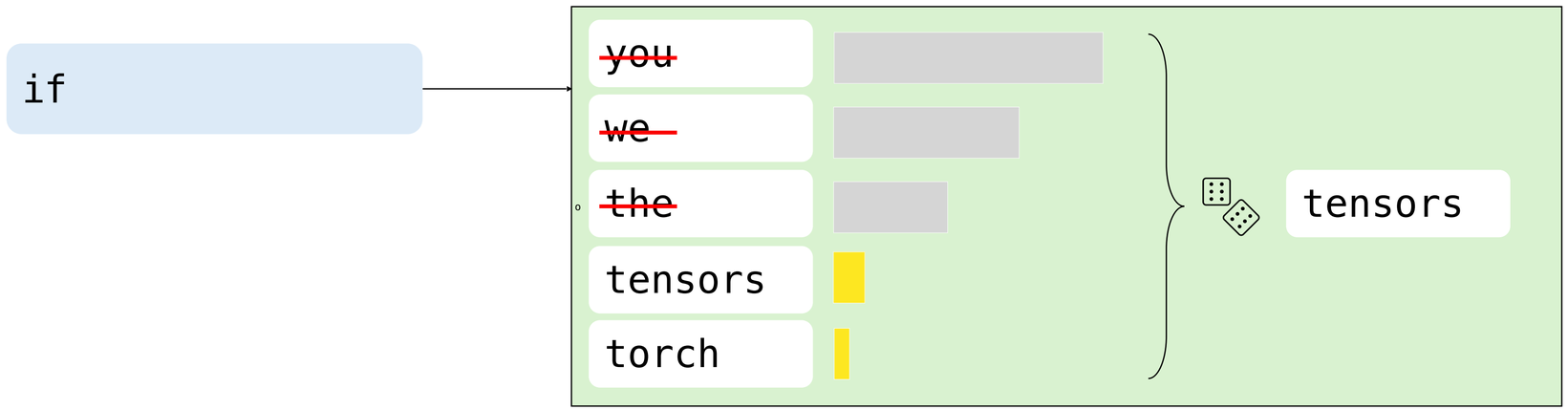
Locally Constrained Decoding samples from the wrong distribution

Example: LM wants to write English instead of Python: “if you have a list of tensors in torch, you can stack it by…”



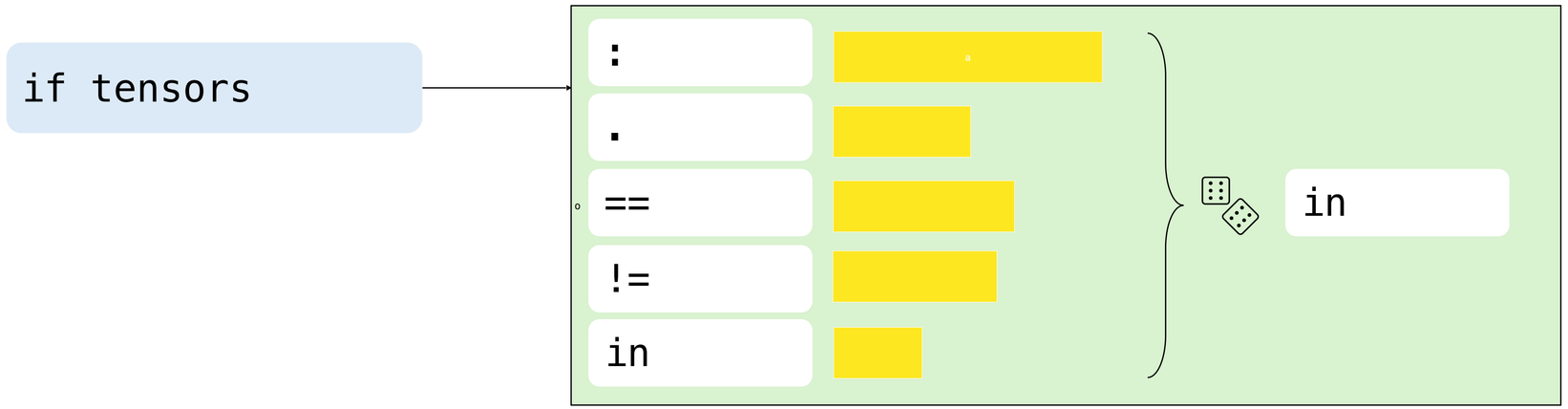
Locally Constrained Decoding samples from the wrong distribution

Example: LM wants to write English instead of Python: “if you have a list of tensors in torch, you can stack it by…”



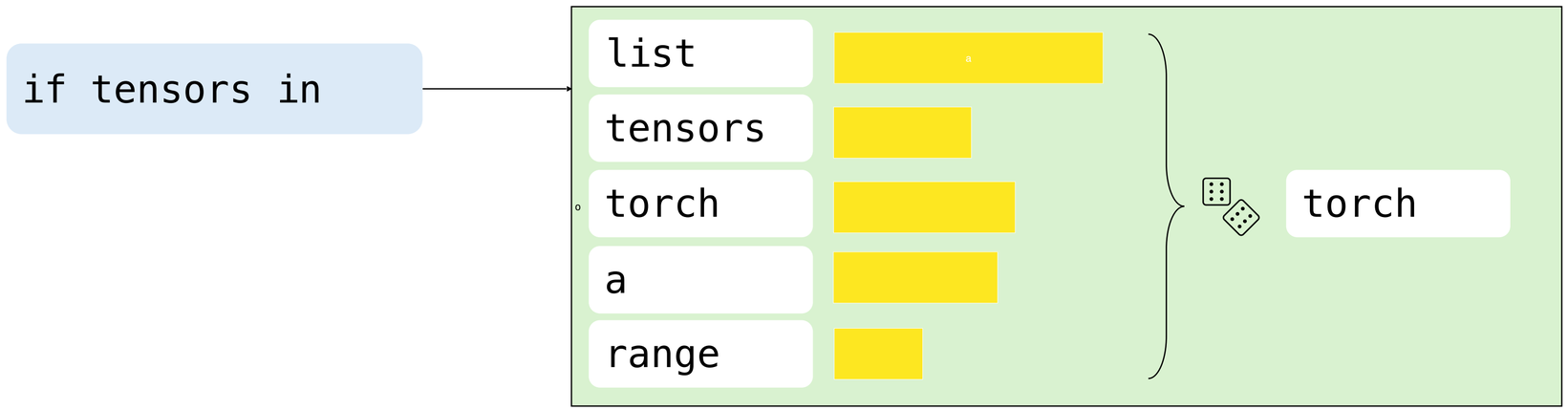
Locally Constrained Decoding samples from the wrong distribution

Example: LM wants to write English instead of Python: “if you have a list of tensors in torch, you can stack it by…”



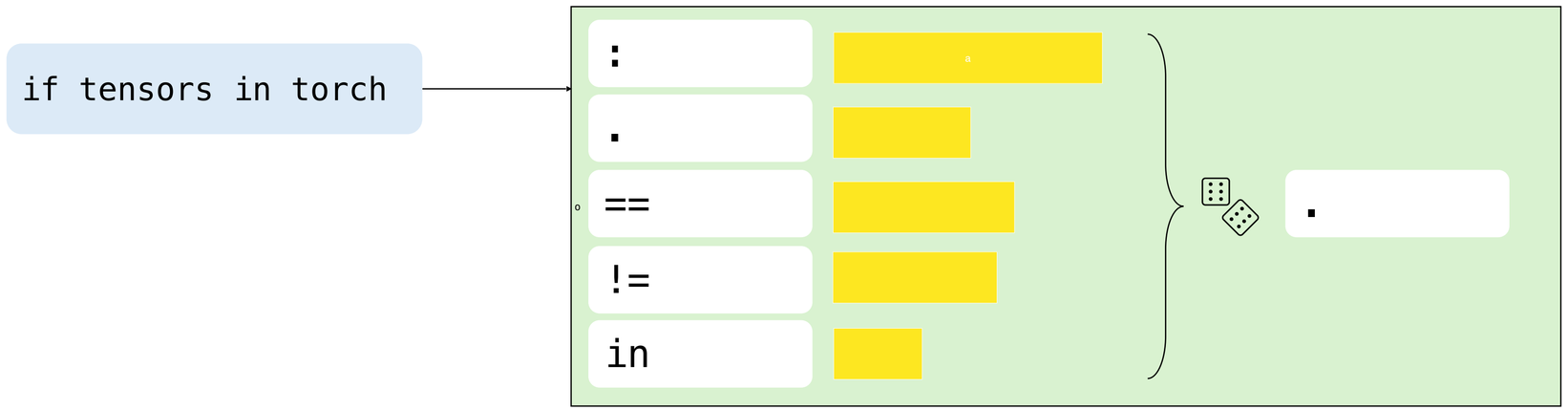
Locally Constrained Decoding samples from the wrong distribution

Example: LM wants to write English instead of Python: “if you have a list of tensors in torch, you can stack it by…”



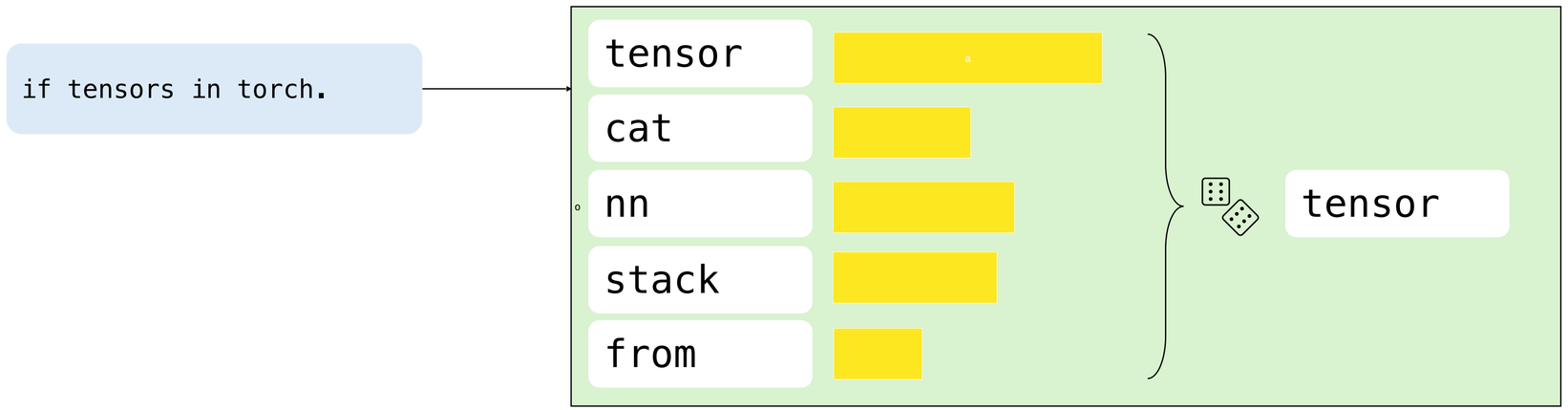
Locally Constrained Decoding samples from the wrong distribution

Example: LM wants to write English instead of Python: “if you have a list of tensors in torch, you can stack it by…”



Locally Constrained Decoding samples from the wrong distribution

Example: LM wants to write English instead of Python: “if you have a list of tensors in torch, you can stack it by…”



Locally Constrained Decoding samples from the wrong distribution

Distribution has the right *support*, but overrepresents greedy samples

```
if tensors in torch.tensor…
```

```
try:
  torch.stack(tensors)
except…
```

```
for i in torch.stack(tensors):…
```

```
with torch.cuda.device(‘0’):…
```

Locally Constrained Decoding samples from the wrong distribution

Distribution has the right *support*, but overrepresents greedy samples

```
if tensors in torch.tensor…
```

```
try:
  torch.stack(tensors)
except…
```

```
for i in torch.stack(tensors):…
```

```
with torch.cuda.device(‘0’):…
```

$$l_\phi(x) = \prod_i^{|x|} \frac{p_{LM}(x_t \mid x_{<t})\phi(x_t \mid x_{<t})}{\sum_{x'} p_{LM}(x' \mid x_{<t})\phi(x' \mid x_{<t})}$$

Current approaches to controlled generation have major flaws

	Sample-rank	Locally Constrained Decoding
Samples from the right distribution?	$g(x) \propto p_{LM}(x)\phi(x)$ ✓	$l_\Phi(x) = \prod_{t=1}^{\lvert x \rvert} \frac{p(x_t \mid \mathbf{x}_{<t})\phi(x_t \mid \mathbf{x}_{<t})}{\sum_{x'} p(x' \mid \mathbf{x}_{<t})\phi(x' \mid \mathbf{x}_{<t})}$ ✗
How many potential calls?	$\lvert \text{samples} \rvert$ ✓	$\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$ ✗
How many LM calls?	$\approx \exp D_{KL}(g \mid\mid p_{LM})$ ✗	1 ✓

Better approaches to controlled generation

	Sample-rank	Locally-Constrained Decoding	?
Output distribution	$g(x) \propto p_{LM}(x)\phi(x)$ ✓	$l_\Phi(x) = \prod_{t=1}^{\lvert x \rvert} \frac{p(x_t \mid \mathbf{x}_{<t})\phi(x_t \mid \mathbf{x}_{<t})}{\sum_{x'} p(x' \mid \mathbf{x}_{<t})\phi(x' \mid \mathbf{x}_{<t})}$ ✗	✓
# potential calls	# LM samples ✓	$\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$ ✗	✓
# LM samples	$\approx \exp D_{KL}(g \mid\mid p_{LM})$ ✗	1 ✓	✓

Better approaches to controlled generation

	Sample-rank	Locally-Constrained Decoding	?
Output distribution	$g(x) \propto p_{LM}(x)\phi(x)$ ✓	$l_\Phi(x) = \prod_{t=1}^{\lvert x \rvert} \frac{p(x_t \mid \mathbf{x}_{<t})\phi(x_t \mid \mathbf{x}_{<t})}{\sum_{x'} p(x' \mid \mathbf{x}_{<t})\phi(x' \mid \mathbf{x}_{<t})}$ ✗	✓
# potential calls	# LM samples ✓	$\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$ ✗	✓
# LM samples	$\approx \exp D_{KL}(g \mid\mid p_{LM})$ ✗	1 ✓	✓

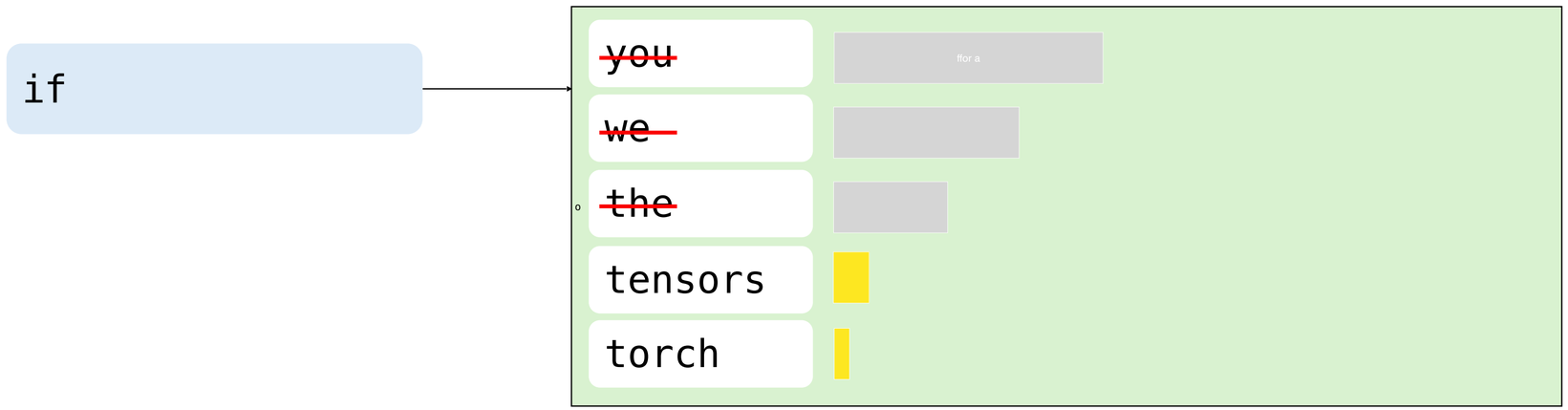
Correcting the output distribution

Correcting the output distribution

Use weights to discount bad samples and emphasize good ones

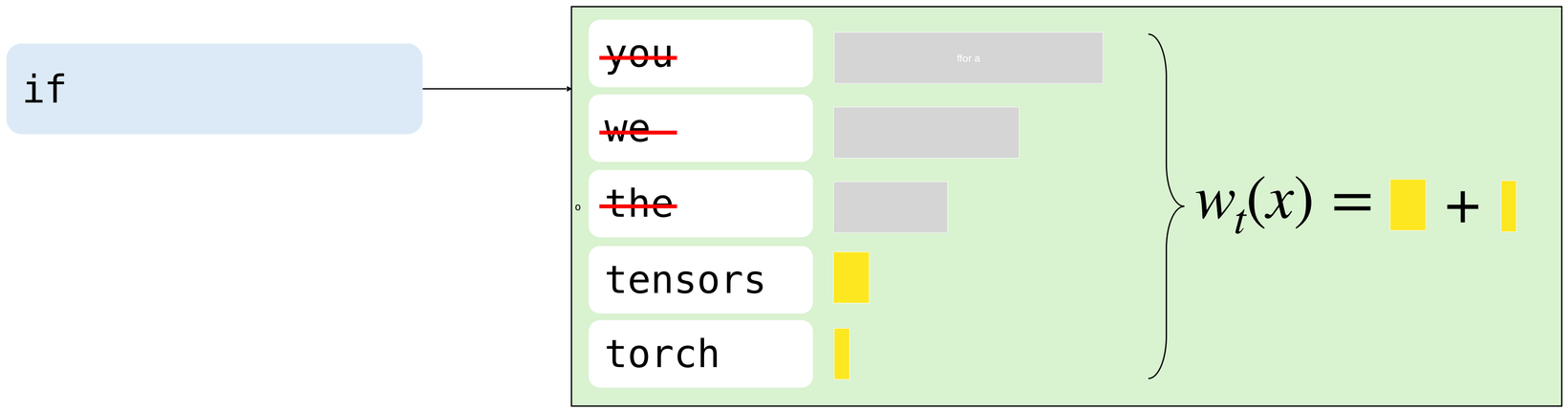
Correcting the output distribution

Use weights to discount bad samples and emphasize good ones



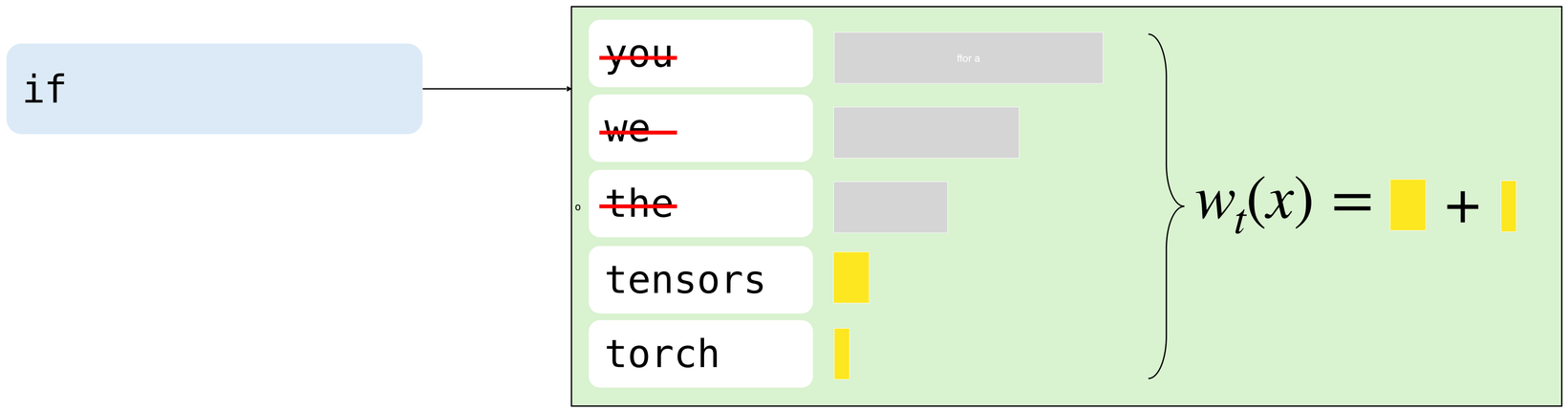
Correcting the output distribution

Use weights to discount bad samples and emphasize good ones



Correcting the output distribution

Use weights to discount bad samples and emphasize good ones



$$l_\phi(x) \prod_{i}^{|x|} w_t(x) = p_{LM}(x)\phi(x)$$

Better approaches to controlled generation

	Sample-rank	Locally-Constrained Decoding	?
Output distribution	$g(x) \propto p_{LM}(x)\phi(x)$ ✓	$l_\Phi(x) = \prod_{t=1}^{\lvert x \rvert} \frac{p(x_t \mid \mathbf{x}_{<t})\phi(x_t \mid \mathbf{x}_{<t})}{\sum_{x'} p(x' \mid \mathbf{x}_{<t})\phi(x' \mid \mathbf{x}_{<t})}$ ✗	$g(x) \propto p_{LM}(x)\phi(x)$ ✓
# potential calls	# LM samples ✓	$\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$ ✗	✓
# LM samples	$\approx \exp D_{KL}(g \mid\mid p_{LM})$ ✗	1 ✓	✓

Better approaches to controlled generation

	Sample-rank	Locally-Constrained Decoding	?
Output distribution	$g(x) \propto p_{LM}(x)\phi(x)$ ✓	$l_\Phi(x) = \prod_{t=1}^{\lvert x \rvert} \frac{p(x_t \mid \mathbf{x}_{<t})\phi(x_t \mid \mathbf{x}_{<t})}{\sum_{x'} p(x' \mid \mathbf{x}_{<t})\phi(x' \mid \mathbf{x}_{<t})}$ ✗	$g(x) \propto p_{LM}(x)\phi(x)$ ✓
# potential calls	# LM samples ✓	$\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$ ✗	✓
# LM samples	$\approx \exp D_{KL}(g \mid\mid p_{LM})$ ✗	1 ✓	✓

Reducing calls to the potentials

Split potentials into Φ_{eff} (efficient) and Φ_{exp}(expensive)

Reducing calls to the potentials

Split potentials into Φ_{eff} (efficient) and Φ_{exp}(expensive)

Φ_{eff}: grammars, finite-state machines, regular expressions, reward models…

Reducing calls to the potentials

Split potentials into Φ_{eff} (efficient) and Φ_{exp}(expensive)

Φ_{eff}: grammars, finite-state machines, regular expressions, reward models…

Φ_{exp}: executing test cases, running simulations, LM as judge…

Reducing calls to the potentials

Split potentials into Φ_{eff} (efficient) and Φ_{exp}(expensive)

Φ_{eff}: grammars, finite-state machines, regular expressions, reward models…

Φ_{exp}: executing test cases, running simulations, LM as judge…

Sample **x** from Locally Constrained Decoding using only Φ_{eff}

Reducing calls to the potentials

Split potentials into Φ_{eff} (efficient) and Φ_{exp}(expensive)

Φ_{eff}: grammars, finite-state machines, regular expressions, reward models…

Φ_{exp}: executing test cases, running simulations, LM as judge…

Sample **x** from Locally Constrained Decoding using only Φ_{eff}

Weights correct for greediness at every step and for Φ_{exp} at the last step

Better approaches to controlled generation

	Sample-rank	Locally-Constrained Decoding	?
Output distribution	$g(x) \propto p_{LM}(x)\phi(x)$ ✓	$l_\Phi(x) = \prod_{t=1}^{\lvert x \rvert} \frac{p(x_t \mid \mathbf{x}_{<t})\phi(x_t \mid \mathbf{x}_{<t})}{\sum_{x'} p(x' \mid \mathbf{x}_{<t})\phi(x' \mid \mathbf{x}_{<t})}$ ✗	$g(x) \propto p_{LM}(x)\phi(x)$ ✓
# potential calls	# LM samples ✓	$\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$ ✗	Φ_{exp}: # LM samples Φ_{eff}: # LM samples × $\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$ ✓
# LM samples	$\approx \exp D_{KL}(g \mid\mid p_{LM})$ ✗	1 ✓	✓

Better approaches to controlled generation

	Sample-rank	Locally-Constrained Decoding	**Syntactic and Semantic Importance Sampling (Ours)**
Output distribution	$g(x) \propto p_{LM}(x)\phi(x)$ ✓	$l_\Phi(x) = \prod_{t=1}^{\lvert x \rvert} \frac{p(x_t \mid \mathbf{x}_{<t})\phi(x_t \mid \mathbf{x}_{<t})}{\sum_{x'} p(x' \mid \mathbf{x}_{<t})\phi(x' \mid \mathbf{x}_{<t})}$ ✗	$g(x) \propto p_{LM}(x)\phi(x)$ ✓
# potential calls	# LM samples ✓	$\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$ ✗	Φ_{exp}: # LM samples Φ_{eff}: # LM samples × $\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$ ✓
# LM samples	$\approx \exp D_{KL}(g \mid\mid p_{LM})$ ✗	1 ✓	✓

Better approaches to controlled generation

	Sample-rank	Locally-Constrained Decoding	**Syntactic and Semantic Importance Sampling (Ours)**
Output distribution	$g(x) \propto p_{LM}(x)\phi(x)$ ✓	$l_\Phi(x) = \prod_{t=1}^{\lvert x \rvert} \frac{p(x_t \mid \mathbf{x}_{<t})\phi(x_t \mid \mathbf{x}_{<t})}{\sum_{x'} p(x' \mid \mathbf{x}_{<t})\phi(x' \mid \mathbf{x}_{<t})}$ ✗	$g(x) \propto p_{LM}(x)\phi(x)$ ✓
# potential calls	# LM samples ✓	$\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$ ✗	Φ_{exp}: # LM samples Φ_{eff}: # LM samples × $\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$ ✓
# LM samples	$\approx \exp D_{KL}(g \mid\mid p_{LM})$ ✗	1 ✓	✓

Better approaches to controlled generation

	Sample-rank	Locally-Constrained Decoding	**Syntactic and Semantic Importance Sampling (Ours)**
Output distribution	$g(x) \propto p_{LM}(x)\phi(x)$ ✓	$l_\Phi(x) = \prod_{t=1}^{\lvert x \rvert} \frac{p(x_t \mid \mathbf{x}_{<t})\phi(x_t \mid \mathbf{x}_{<t})}{\sum_{x'} p(x' \mid \mathbf{x}_{<t})\phi(x' \mid \mathbf{x}_{<t})}$ ✗	$g(x) \propto p_{LM}(x)\phi(x)$ ✓
# potential calls	# LM samples ✓	$\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$ ✗	Φ_{exp}: # LM samples Φ_{eff}: # LM samples × $\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$ ✓
# LM samples	$\approx \exp D_{KL}(g \mid\mid p_{LM})$ ✗	1 ✓	$\approx \exp D_{KL}(g \mid\mid l_{\Phi_{\text{eff}}})$ ✓

Reducing LM samples: Sequential Monte Carlo

Use incremental weights to focus computation on promising particles

Reducing LM samples: Sequential Monte Carlo

Use incremental weights to focus computation on promising particles

Current generations

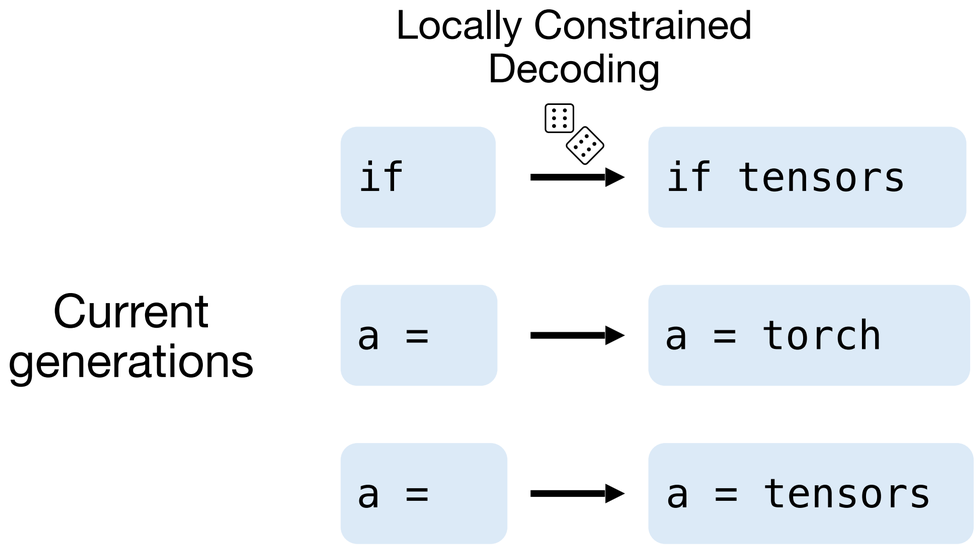
```
if
```

```
a =
```

```
a =
```

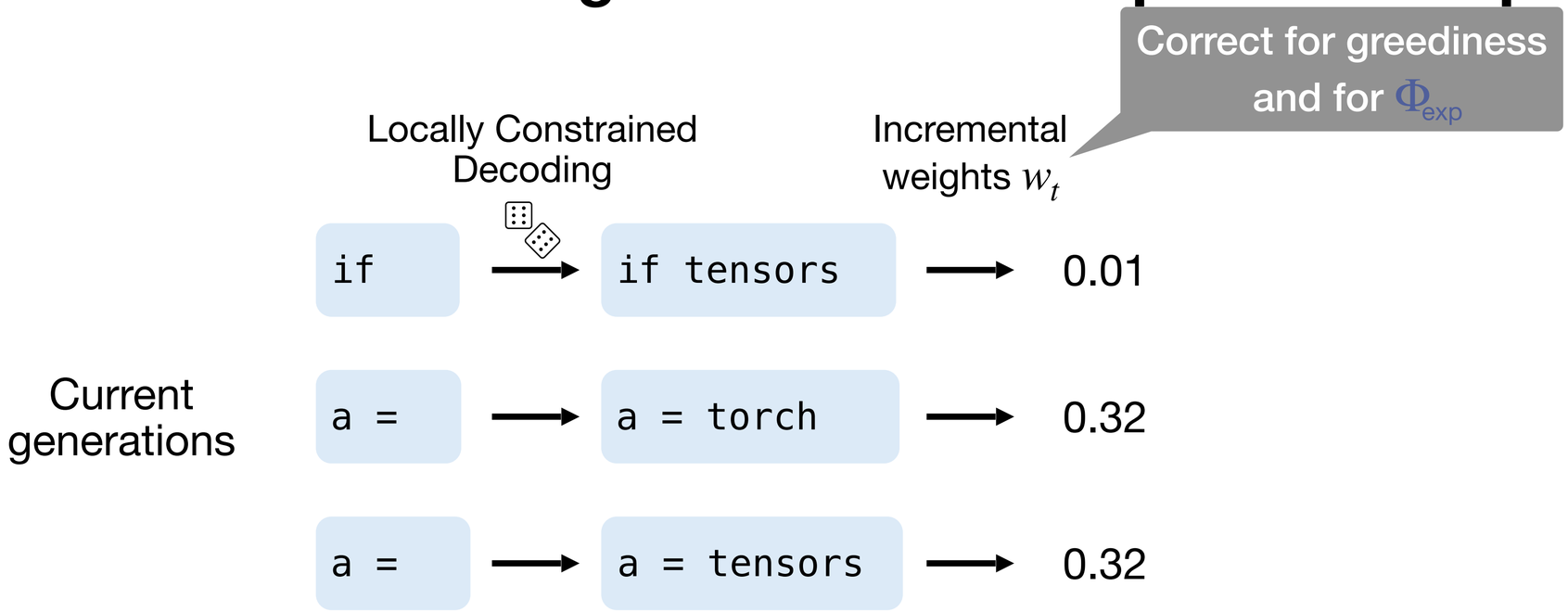
Reducing LM samples: Sequential Monte Carlo

Use incremental weights to focus computation on promising particles



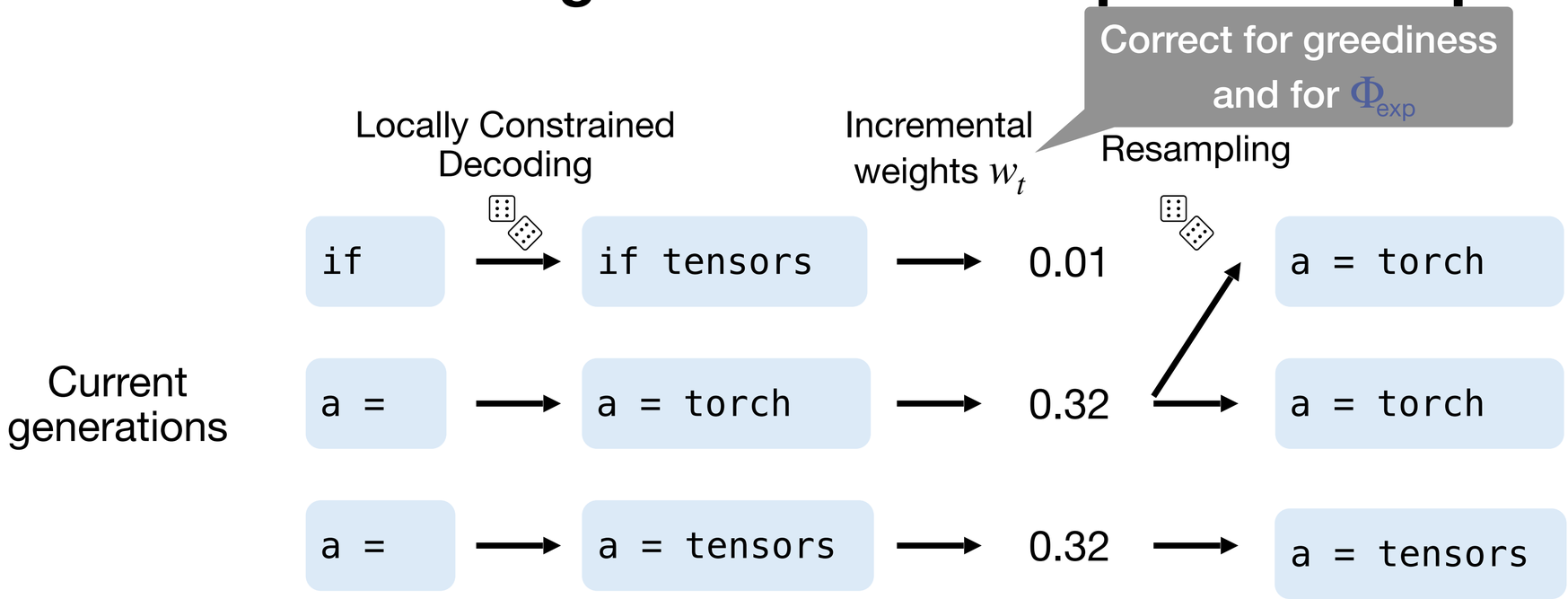
Reducing LM samples: Sequential Monte Carlo

Use incremental weights to focus computation on promising particles



Reducing LM samples: Sequential Monte Carlo

Use incremental weights to focus computation on promising particles



More efficient approaches to controlled generation

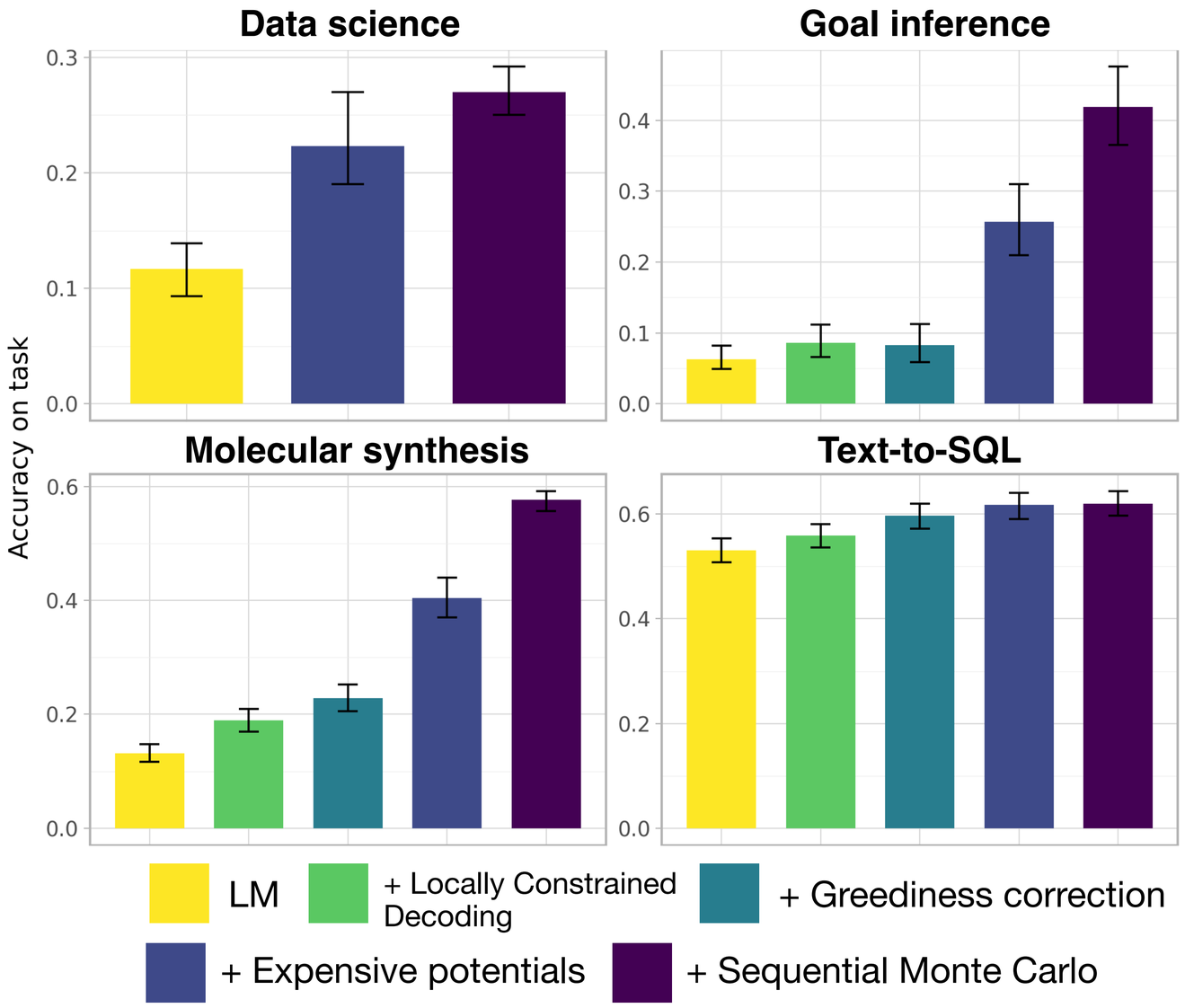
	Sample-rank	Locally-Constrained Decoding	**Syntactic and Semantic Importance Sampling (Ours)**	**Syntactic and Semantic Sequential Monte Carlo (Ours)**
Output distribution	$g(x) \propto p_{LM}(x)\phi(x)$	$l_\Phi(x) = \prod_{t=1}^{\lvert x \rvert} \frac{p(x_t \mid \mathbf{x}_{<t})\phi(x_t \mid \mathbf{x}_{<t})}{\sum_{x'} p(x' \mid \mathbf{x}_{<t})\phi(x' \mid \mathbf{x}_{<t})}$	$g(x) \propto p_{LM}(x)\phi(x)$	$g(x) \propto p_{LM}(x)\phi(x)$
# verifier calls	# LM samples	$\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$	Φ_{exp}: # LM samples Φ_{eff}: # LM samples × $\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$	Φ_{exp}: # LM samples × $\lvert \mathbf{x} \rvert$ Φ_{eff}: # LM samples × $\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$
# LM samples	$\approx \exp D_{KL}(g \mid\mid p_{LM})$	1	$\approx \exp D_{KL}(g \mid\mid l_\Phi)$	~10x fewer than Syntactic and Semantic Importance Sampling

More efficient approaches to controlled generation

How do these methods perform?

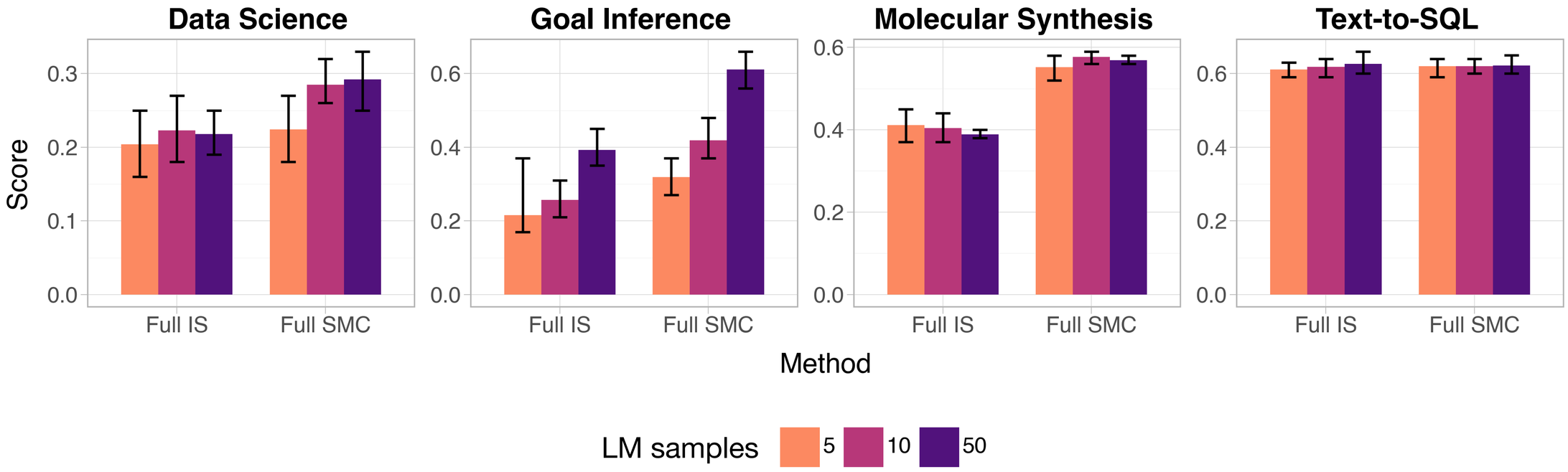
	Sample-rank	Locally-Constrained Decoding	**Syntactic and Semantic Importance Sampling (Ours)**	**Syntactic and Semantic Sequential Monte Carlo (Ours)**
Output distribution	$g(x) \propto p_{LM}(x)\phi(x)$ ✓	$l_\Phi(x) = \prod_{t=1}^{\lvert x \rvert} \frac{p(x_t \mid \mathbf{x}_{<t})\phi(x_t \mid \mathbf{x}_{<t})}{\sum_{x'} p(x' \mid \mathbf{x}_{<t})\phi(x' \mid \mathbf{x}_{<t})}$ ✗	$g(x) \propto p_{LM}(x)\phi(x)$ ✓	$g(x) \propto p_{LM}(x)\phi(x)$ ✓
# verifier calls	# LM samples ✓	$\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$ ✗	Φ_{exp}: # LM samples Φ_{eff}: # LM samples × $\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$ ✓	Φ_{exp}: # LM samples × $\lvert \mathbf{x} \rvert$ Φ_{eff}: # LM samples × $\lvert \text{vocabulary} \rvert \times \lvert \mathbf{x} \rvert$ ✓
# LM samples	$\approx \exp D_{KL}(g \mid\mid p_{LM})$ ✗	1 ✓	$\approx \exp D_{KL}(g \mid\mid l_\Phi)$ ✓	~10x fewer than Syntactic and Semantic Importance Sampling ✓

Sequential Monte Carlo boosts performance across challenging domains

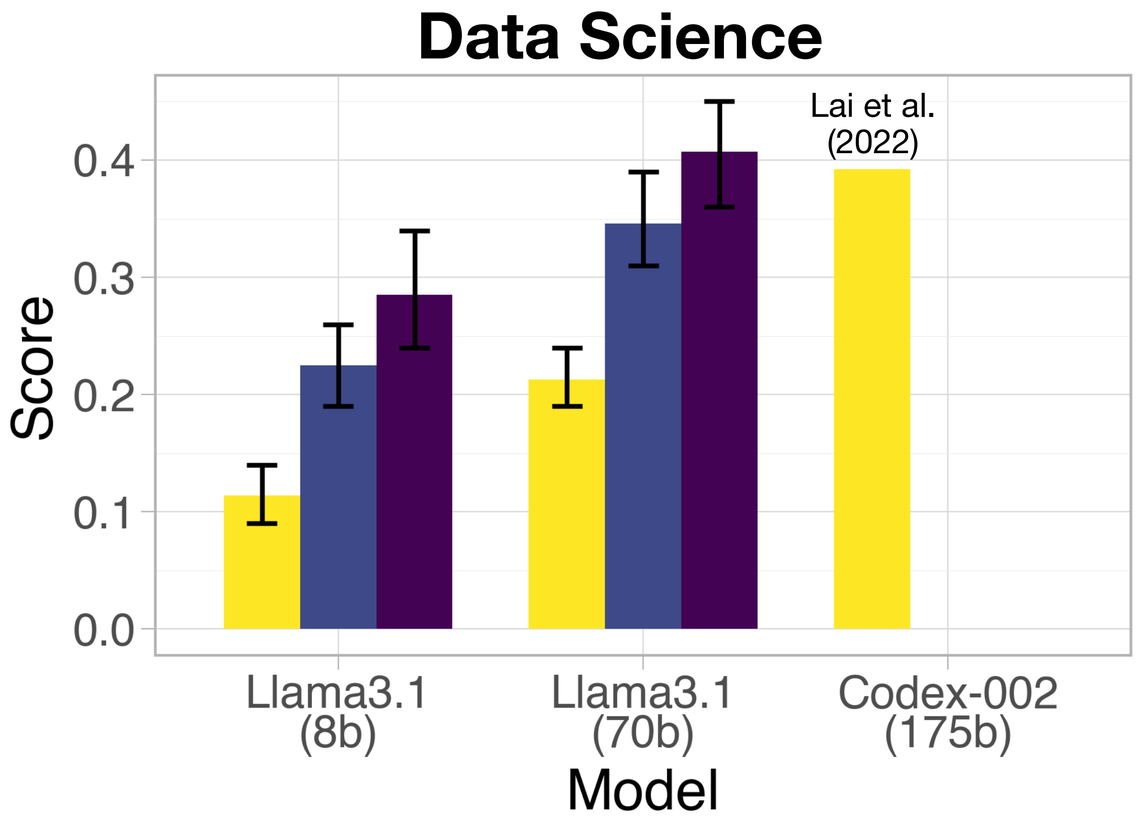


Sequential Monte Carlo is more frugal than Importance Sampling

~10x fewer LM calls in domains with constraining potentials



Sequential Monte Carlo allows small LMs to punch above their weights



Base LM + greediness correction, expensive potentials + SMC

tl;dr

Frame controlled generation as *probabilistic inference*

Use *Sequential Monte Carlo* to sample from the posterior

Library: github.com/genlm/genlm-control

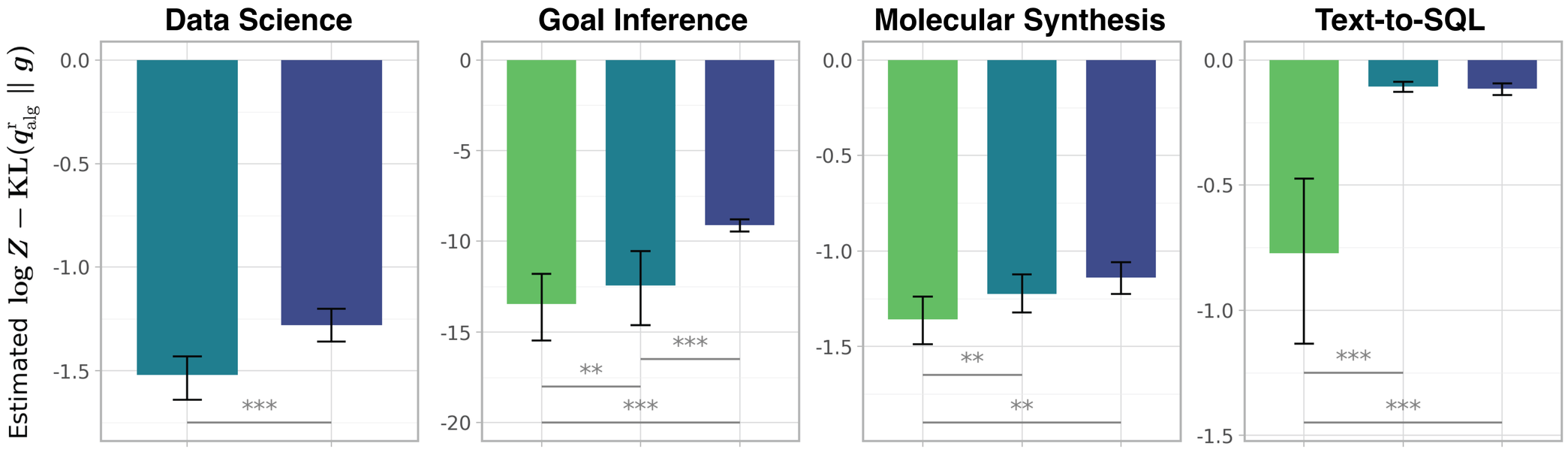
```python
from genlm.control import PromptedLLM, BoolFSA, AWRS

# Create a language model potential.
llm = PromptedLLM.from_name("gpt2")
llm.set_prompt_from_str("Here is my honest opinion:")

# Create a finite-state automaton potential using a regular expression.
fsa = BoolFSA.from_regex(r" SMC is (🔥🔥|😍😍|🤌🤌) with LMs")

# Coerce the FSA so that it operates on the token type of the language model.
coerced_fsa = fsa.coerce(llm, f=b"".join)

# Create a token sampler that combines the language model and FSA.
token_sampler = AWRS(llm, coerced_fsa)

# Generate text using SMC.
# Generation is asynchronous; use `await` if calling in an async context (like in an async
# function or in a Jupyter notebook) and `asyncio.run(token_sampler.smc(...))` otherwise.
sequences = await token_sampler.smc(
    n_particles=10, # Number of candidate sequences to maintain
    ess_threshold=0.5, # Threshold for resampling
    max_tokens=30, # Maximum sequence length
    verbosity=1 # Print particles at each step
)

sequences.decoded_posterior
# Example output:
# {
#   ' SMC is 🔥🔥 with LMs': 1.0,
# }
```

Our org GenLM is recruiting! genlm.org

Appendix

Does it matter that our output follows the right distribution?

Yes! Methods that are closer in KL to true posterior perform better



Locally Constrained Decoding + expensive potentials

+ greediness correction

+ Sequential Monte Carlo

Does it matter that our output follows the right distribution?

Yes! Better methods' probabilities more correlated with downstream performance