Compiling Stan to Generative Probabilistic Languages and Extension to Deep Probabilistic Programming

Guillaume Baudart (Inria), Louis Mandel (IBM)
Javier Burroni (UMass Amherst), Martin Hirzel (IBM), Avraham Shinnar (IBM)
Stan: A Probabilistic Programming Language

“A Stan program imperatively defines a log probability function over parameters conditioned on specified data and constants”

In a nutshell:
- Small imperative language to describe probabilistic models
- Bayesian inference on continuous latent variables
- No U-Turn Sampler (NUTS): an optimized Hamiltonian Monte Carlo (HMC) inference
- Interface with popular programming language: R, Python, etc…

Impressive documentation
- User Manual: 393 pages
- Reference Manual: 188 pages
- Functions Reference: 202 pages
- + tutorials, case studies, notebooks...

Academic research papers and text books
- all fields of science:
  - epidemiology, chemistry, economics, cognitive science, zoology, ...

Carpenter et al. 2017
Stan: Example

data {
  int<lower=0> N;          // players
  int<lower=0> K[N];       // initial trials
  int<lower=0> y[N];       // initial successes
}

parameters {
  vector<lower=0, upper=1>[N] theta; // chance of success
}

model {
  theta ~ uniform(0, 1);                // prior
  y ~ binomial(K, theta);               // likelihood
}
Stan: Example

data {
  int<lower=0> N; // players
  int<lower=0> K[N]; // initial trials
  int<lower=0> y[N]; // initial successes
}

parameters {
  real<lower=0, upper=1> phi; // population chance of success
  real<lower=1> kappa; // population concentration
  vector<lower=0, upper=1>[N] theta; // chance of success
}

model {
  phi ~ uniform(0, 1); // hyperprior
  kappa ~ pareto(1, 1.5); // hyperprior
  theta ~ beta(phi * kappa, (1 - phi) * kappa); // prior
  y ~ binomial(K, theta); // likelihood
}
Generative Probabilistic Languages

General purpose programming languages extended with probabilistic constructs
- \( x = \text{sample}(D) \): draw a sample from a distribution
- \( \text{observe}(D, \text{obs}) \): penalize execution path assuming \( \text{obs} \) was sampled from \( D \)
- \( \text{infer}(m, \text{obs}) \): compute the posterior distribution of a model \( m \) given observations \( \text{obs} \)

Multiple examples:
- Church, Anglican (lisp, clojure), 2008
- WebPPL (javascript), 2014
- Pyro/NumPyro (python), 2017/2019
- Gen (julia), 2018
- ProbZelus (Zelus), 2019
- ...

More and more, incorporating new ideas:
- New inference techniques, e.g., stochastic variational inference (SVI)
- Interaction with neural nets (deep probabilistic programming)
Why Compiling Stan to Generative Languages?

For Stan users
- New inference techniques
- New features
- Speed: optimized compilation, hardware accelerators, ...
- Interaction with different host languages

For generative language developers
- Access to a new community in academia and industry
- Hundreds of existing models
- Baseline for research comparison
def model(N, K, y):
    phi = sample('phi', uniform(0, 1))
    kappa = sample('kappa', pareto(1, 1.5))
    theta = sample('theta', beta(phi * kappa, (1-phi) * kappa, shape=N))
    observe('y', binomial(K, theta), y)
    return (phi, kappa, theta)
def model(N, K, y):
    phi = sample('phi', uniform(0, 1))
    kappa = sample('kappa', pareto(1, 1.5))
    theta = sample('theta', beta(phi * kappa, (1 - phi) * kappa), shape=N)
    observe('y', binomial(K, theta), y)
    return (phi, kappa, theta)
Generative Compilation?

...
Generative Compilation?

parameters {
    real mu; // population mean of success log-odds
    real<lower=0> sigma; // population std of success log-odds
    vector[N] alpha_std; // success log-odds
}
transformed parameters {
    vector[N] phi; 
    phi = mu + sigma * alpha_std;
}
model {
    mu ~ normal(-1, 1); // hyperprior
    sigma ~ normal(0, 1); // hyperprior
    alpha_std ~ normal(0, 1); // prior
    y ~ binomial_logit(K, phi); // likelihood
}

def model(N, K, y):
    mu = sample('mu', normal(-1, 1))
    sigma = sample('sigma', normal(0, 1))
    alpha_std = sample('alpha_std', normal(0, 1))
    observe('y', binomial_logit(K, phi), y)
    return phi

NameError: name 'mu' is not defined  Should be >0
Non-Generative Features

531 Stan models

- 69% generative
- 31% non-generative
Non-Generative Features

parameters {
  real<lower=0, upper=1> a;
  real b;
  real c;
  real d;
  real<lower=0, upper=1> e;
}
model {
  // where is a?
  // implicit prior
  b ~ normal(c, a); // automatic scheduling
  sqrt(c) ~ normal(2, 0.01); // left expression
  d ~ normal(10, 1); // double sampling
  d ~ normal(-10, 1); // double sampling
  e ~ normal(-10, 1); // missing constraint
}

<table>
<thead>
<tr>
<th>name</th>
<th>Mean</th>
<th>MCSE</th>
<th>StdDev</th>
</tr>
</thead>
<tbody>
<tr>
<td>lp__</td>
<td>-156.555000</td>
<td>0.059876</td>
<td>1.810810</td>
</tr>
<tr>
<td>a</td>
<td>0.524915</td>
<td>0.010513</td>
<td>0.281969</td>
</tr>
<tr>
<td>b</td>
<td>3.990390</td>
<td>0.015614</td>
<td>0.602390</td>
</tr>
<tr>
<td>c</td>
<td>4.000020</td>
<td>0.000969</td>
<td>0.039872</td>
</tr>
<tr>
<td>d</td>
<td>0.010637</td>
<td>0.014740</td>
<td>0.700678</td>
</tr>
<tr>
<td>e</td>
<td>0.094737</td>
<td>0.001897</td>
<td>0.093924</td>
</tr>
</tbody>
</table>
Comprehensive Translation

Key idea:
- Compile all ~ statement to `observe` statement
- Add uniform priors for all parameters

```python
parameters {
    real<lower=0, upper=1> a;
    real b;
    real c;
    real d;
    real<lower=0, upper=1> e;
}
model {
    b ~ normal(c, a);
    sqrt(c) ~ normal(2, 0.01);
    d ~ normal(10, 1);
    d ~ normal(-10, 1);
    e ~ normal(-10, 1);
}
def model():
    a = sample('a', uniform(0, 1))
    b = sample('b', improper_uniform(shape=[]))
    c = sample('c', improper_uniform(shape=[]))
    d = sample('d', improper_uniform(shape=[]))
    e = sample('e', uniform(0, 1))
    observe('_b__1', normal(c, a), b)
    observe('_expr__2', normal(2, 0.01), sqrt_real(c))
    observe('_d__3', normal(10, 1), d)
    observe('_d__4', normal(-10, 1), d)
    observe('_e__5', normal(-10, 1), e)
```
Semantics

Stan and GProb
Stan Syntax

\[
\text{program ::= functions } \{\text{fundecl}*\} \? \\
\text{data } \{\text{decl}*\} \? \\
\text{transformed data } \{\text{decl}^* \text{ stmt}\} \? \\
\text{parameters } \{\text{decl}*\} \? \\
\text{transformed parameters } \{\text{decl}^* \text{ stmt}\} \? \\
\text{model } \{\text{decl}^* \text{ stmt}\} \\
\text{generated quantities } \{\text{decl}^* \text{ stmt}\} \? \\
\]

\[
\text{stmt ::= } x = e \\
| x[e_1, ..., e_n] = e \\
| stmt_1; stmt_2 \\
| for (x in e_1:e_2) \{stmt\} \\
| for (x in e) \{stmt\} \\
| while (e) \{stmt\} \\
| if (e) stmt_1 else stmt_2 \\
| skip \\
| target += e \\
| e \sim f(e_1, ..., e_n)
\]

\[
decl ::= \text{base_type constraint } x \\
| \text{base_type constraint } x [\text{shape}] \\
base_type ::= \text{real} | \text{int} \\
| \text{vector}[\text{size}] | \text{matrix}[\text{size}, \text{size}] \\
constraint ::= \epsilon | \langle \text{lower} = e, \text{upper} = e \rangle \\
| \langle \text{lower} = e \rangle | \langle \text{upper} = e \rangle
\]
Stan Semantics

A small imperative language with:
- A global accumulator \texttt{target}
- Two constructs to update this accumulator: \sim and \texttt{target +=}

Model semantics: unnormalized log-density of the model
\[ \{p\} : \mathcal{D} \to \Sigma_x \to [0,\infty), \text{ where } X \text{ is the parameters domain.} \]

Given an initial environment \( D \) containing the (observed) data we have:
\[
\{p\}_D = \lambda U \int_{U} \exp([\text{model}(p)]_{D,\theta}(\text{target})) \, d\theta
\]
a measure mapping sets of parameter values \( U \) to a score in \([0,\infty)\).
Stan Semantics

Statements: $[[s]] : (\text{Var} \rightarrow \text{Val}) \rightarrow (\text{Var} \rightarrow \text{Val})$

$[[x = e]]_\gamma = \gamma[x \leftarrow [[e]]_\gamma]$  
$[[x[e_1, ..., e_n] = e]]_\gamma = \gamma[x \leftarrow (x[[e_1]]_\gamma, ..., [[e_n]]_\gamma) \leftarrow [[e]]_\gamma]$  
$[[s_1; s_2]]_\gamma = [[s_2]][[s_1]]_\gamma$  
$[[\text{for} \ (x \ \text{in} \ e_1:e_2) \ \{s\}}]]_\gamma =$  
  
  let $n_1 = [[e_1]]_\gamma$ in let $n_2 = [[e_2]]_\gamma$ in  
  
  if $n_1 > n_2$ then $\gamma$ else $[[\text{for} \ (x \ \text{in} \ n_1 + 1:n_2) \ \{s\}}]]_\gamma[x \leftarrow n_1$  

$[[\text{while} \ (e) \ \{s\}}]]_\gamma = \text{if} \ [[e]]_\gamma = 0 \ \text{then} \ \gamma \ \text{else} \ [[\text{while} \ (e) \ \{s\}}]]_\gamma$  

$[[\text{if} \ (e) \ s_1 \ \text{else} \ s_2]]_\gamma = \text{if} \ [[e]]_\gamma \neq 0 \ \text{then} \ [[s_1]]_\gamma \ \text{else} \ [[s_2]]_\gamma$  

$[[\text{skip}}]]_\gamma = \gamma$  

$[[e_1 = e_2]]_\gamma = \text{let} \ D = [[e_2]]_\gamma \ \text{in} \ [[\text{target} += D_{lpdf}(e_1)]]_\gamma$
GProb: A simple Generative PPL

GProb is an expression language

\[ e ::= c \mid x \mid \{e_1, \ldots, e_n\} \mid [e_1, \ldots, e_n] \mid e_1[e_2] \mid f(e_1, \ldots, e_n) \]
\[ \mid \text{let } x = e_1 \text{ in } e_2 \mid \text{let } x[e_1, \ldots, e_n] = e \text{ in } e' \]
\[ \mid \text{if } (e) \text{ e}_1 \text{ else } e_2 \mid \text{for } \chi \ (x \text{ in } e_1 : e_2) \ e_3 \mid \text{while } \chi \ (e_1) \ e_2 \]
\[ \mid \text{factor}(e) \mid \text{sample}(e) \mid \text{return}(e) \]

- Loops are parameterized by the set \( \chi \) of variables updated in their body
- Classic probabilistic constructs
- \textbf{factor}: smooth conditioning
- \textbf{observe}(D, v) \equiv \textbf{factor}(D_{pdf}(v))
GProb Semantics

The semantics of an expression is a kernel:

\[ \{e\} : \mathcal{D} \to \Sigma_E \to [0,\infty) \]

\[
\{\text{return}(e)\}_y = \lambda U. \delta_{\{e\}_y}(U)
\]

\[
\{\text{let } x = e_1 \text{ in } e_2\}_y = \lambda U. \int_X \{e_1\}_y(dv) \times \{e_2\}_{y[x\leftarrow v]}(U)
\]

\[
\{\text{let } x[e_1,\ldots,e_n] = e \text{ in } e'\}_y = \\
\hspace{1cm} \lambda U. \int_X \{e\}_y(dv) \times \{e'\}_{y[x\leftarrow (x[e_1,\ldots,[e_n]_y\leftarrow v])]}(U)
\]

\[
\{\text{if } (e) \text{ e}_1 \text{ else } e_2\}_y = \lambda U. \text{if } \{e\} \neq 0 \text{ then } \{e_1\}_y(U) \text{ else } \{e_2\}_y(U)
\]

\[
\{\text{sample}(e)\}_y = \lambda U. \{e\}_y(U)
\]

\[
\{\text{factor}(e)\}_y = \lambda U. \exp(\{e\}_y)\delta(\cdot)(U)
\]
GProb Semantics: Loops

Stan: loops cannot depend on probabilistic parameters
- No need for fix-point operator in probabilistic space
- Semantics of loops capture 2 steps: elaboration (unfolding) + evaluation (integration)

$$\{\text{for}_{\mathcal{X}} (x \in e_1:e_2) \ e_3\}_Y =$$

$$\lambda U. \text{let } n_1 = [e_1]_Y \text{ in let } n_2 = [e_2]_Y \text{ in}$$

$$\quad \text{if } n_1 > n_2 \text{ then } \delta_{Y(\mathcal{X})}(U)$$

$$\quad \text{else } \int_{\mathcal{X}} \{e_3\}_Y[x \leftarrow n_1](dX) \times \{\text{for}_{\mathcal{X}} (x \in n_1 + 1:n_2) \ e_3\}_{Y,\mathcal{X}(U)}$$

$$\{\text{while}_{\mathcal{X}} (e_1) \ e_2\}_Y =$$

$$\lambda U. \text{if } [e_1]_Y = 0 \text{ then } \delta_{Y(\mathcal{X})}(U)$$

$$\quad \text{else } \int_{\mathcal{X}} \{e_2\}_Y(dX) \times \{\text{while}_{\mathcal{X}} (e_1) \ e_2\}_{Y,\mathcal{X}(U)}$$
Compilation

Stan to GProb
Comprehensive Compilation

Key idea:
- Sample all parameter from priors with a constant density that can be normalized away
- Compile all $\sim$ statements as observe statements

Compilation functions are parameterized by a continuation $k$
- $\mathcal{P}_k(params(p))$: compile parameters block, i.e., introduces the priors
- $\mathcal{S}_k(model(p))$: compile model block, imperative style to functional style
- Finally add a return statement for all the parameters

$$\mathcal{C}(p) = \mathcal{P}_{\mathcal{S}_{\text{return}(params(p))}(model(p))}(params(p))$$

```
let $x_1 = \text{sample}(U_1)$ in ... in let $x_n = \text{sample}(U_n)$ in
let $y_1 = \text{return}(e_1)$ in let () = factor($e_1'$) in
...
let $y_m = \text{return}(e_m)$ in let () = factor($e_m'$) in
return ($x_1, \ldots, x_n$)
```
Correctness

**Theorem:** For all Stan program $p$, the semantics of the source and the compiled programs are equal up to a constant:

$$\{p\}_D \propto \{C(p)\}_D$$
Correctness

Theorem: For all Stan program $p$, the semantics of the source and the compiled programs are equal up to a constant:

$$\{[p] \}_D \propto \{[\mathcal{C}(p)] \}_D$$

Lemma 1: For all Stan program $p$ and environment $\gamma$:

$$\{[\mathcal{C}(p)] \}_\gamma = \lambda U \cdot \int_{X_1} D_1(dx_1) \ldots \int_{X_n} D_n(dx_n) \{[\mathcal{S}_{\text{return}}()]\} \gamma,\{x_1,\ldots,x_n\}(U)$$

$$\propto \lambda U \cdot \int_{U} \{[\mathcal{S}_{\text{return}}()]\} \gamma,\emptyset(\{\}) d\theta$$

```
let x_i = sample(U_i) in ... in let x_n = sample(U_n) in
let y_i = return(e_i) in let () = factor(e'_i) in ...
let y_m = return(e_m) in let () = factor(e'_m) in
return (x_1, ..., x_n)
```

Lemma 2: For all Stan statements $stmt$ compiled with a continuation $k$, if $\gamma(\text{target}) = 0$, and $\{[stmt]\}_\gamma = \gamma'$:

$$\{[\mathcal{S}_k(stmt)] \}_\gamma = \lambda U \cdot \exp(\gamma'(\text{target})) \times \{[k] \}_\gamma |_{\text{target} \leftarrow 0}(U)$$
Lemma 2: For all Stan statements \( stmt \) compiled with a continuation \( k \), if \( \gamma(\text{target}) = 0 \), and \( \{ stmt \}_\gamma = \gamma' \):
\[
\{ S_k(stmt) \}_\gamma = \lambda U \cdot \exp(\gamma'(\text{target})) \times \{ k \}_\gamma[\text{target} \leftarrow 0](U)
\]

Proof: By induction on the structure of \( stmt \).

Assignment:
\[
\{ S_k(x = e) \}_\gamma = \{ \text{let } x = \text{return}(e) \text{ in } k \}_\gamma
\]
\[
= \lambda U \int_X \{ \text{return}(e) \}_\gamma(dv) \times \{ k \}_\gamma[x\leftarrow v](U)
\]
\[
= \lambda U \int_X \delta_{\{e\}_\gamma}(dv) \times \{ k \}_\gamma[x\leftarrow v](U)
\]
\[
= \lambda U \cdot 1 \times \{ k \}_\gamma[x\leftarrow \{e\}_\gamma](U)
\]
\[
= \lambda U \cdot \exp(\gamma'(\text{target})) \times \{ k \}_\gamma[\text{target} \leftarrow 0](U) \quad \text{with } \{x = e\}_\gamma = \gamma[x \leftarrow \{e\}_\gamma] = \gamma'
\]
and \( \gamma'[\text{target} \leftarrow 0] = \gamma' \)
**Lemma 2**: For all Stan statements $stmt$ compiled with a continuation $k$, if $\gamma(\text{target}) = 0$, and $[[stmt]]_\gamma = \gamma'$:

$$[[\mathcal{S}_k(stmt)]]_\gamma = \lambda U. \exp(\gamma'(\text{target})) \times \{k\}_\gamma[\text{target} \leftarrow 0](U)$$

**Proof**: By induction on the structure of $stmt$.

Target update:

$$[[\mathcal{S}_k(\text{target} += e)]]_\gamma = [[\text{let }() = \text{factor}(e) \text{ in } k]]_\gamma$$

$$= \lambda U. \int_0^\infty \exp(\{e\}_\gamma) \delta_0(v) \, dv \times \{k\}_\gamma(U)$$

$$= \lambda U. \exp(\{e\}_\gamma) \times \{k\}_\gamma(U)$$

$$= \lambda U. \exp(\gamma'(\text{target})) \times \{k\}_\gamma[\text{target} \leftarrow 0](U)$$

with $[[\text{target} += e]]_\gamma = \gamma'$ and $\gamma = \gamma'[\text{target} \leftarrow 0]$
Lemma 2: For all Stan statements \textit{stmt} compiled with a continuation \(k\), if \(\gamma(\text{target}) = 0\), and \([\text{stmt}]_\gamma = \gamma'\):
\[
[\mathcal{S}_k(\text{stmt})]_\gamma = \lambda U \cdot \exp(\gamma'(\text{target})) \times \{k\}_{\gamma[\text{target} \leftarrow 0]}(U)
\]

Proof: By induction on the structure of \textit{stmt}.

Sequence: with \(\gamma_1 = [\text{stmt}_1]_\gamma\) and \(\gamma_2 = [\text{stmt}_2]_{\gamma_1[\text{target} \leftarrow 0]}\) we have:
\[
[\mathcal{S}_k(\text{stmt}_1 ; \text{stmt}_2)]_\gamma = [\mathcal{S}_{\mathcal{S}_k(\text{stmt}_2)}(\text{stmt}_1)]_\gamma
\]
\[
= \lambda U \cdot \exp(\gamma_1(\text{target})) \times \{\mathcal{S}_k(\text{stmt}_2)\}_{\gamma_1[\text{target} \leftarrow 0]}(U)
\]
\[
= \lambda U \cdot \exp(\gamma_1(\text{target})) \times \exp(\gamma_2(\text{target})) \times \{k\}_{\gamma_2[\text{target} \leftarrow 0]}(U)
\]
\[
= \lambda U \cdot \exp(\gamma_1(\text{target}) + \gamma_2(\text{target})) \times \{k\}_{\gamma_2[\text{target} \leftarrow 0]}(U)
\]
And in Stan:
\[
([\text{stmt}_1 ; \text{stmt}_2]_\gamma(\text{target}) = \gamma_1(\text{target}) + \gamma_2(\text{target})
\]
Mixed Compilation: Optimization

Key idea:
- Schedule statements using dependencies
- Use informative priors whenever possible

We can recover generative translation when possible

Proof:
- Move \textit{sample} as close as possible to \textit{observe} (commutativity [Staton'17])
- Merge: \( x = \text{sample}(\text{uniform}); \text{observe}(D, x) \equiv x = \text{sample}(D) \)
Experiments

Stan to (Num)Pyro
Stan to (Num)Pyro

Fork of the Stanc3 compiler (OCaml) with two new target languages:
- Pyro: Python + PyTorch
- NumPyro: Python + JAX

Implementation
- Compiler: 2k LoC (OCaml)
- Runtime: 2 x 100 LoC (Python)
- Standard library and distributions: 2 x 2k LoC (Python)

Challenges
- Different name spaces, keywords etc...
- Function overloading: postfix function name with type annotation
- One-based vs. zero-based indexing
- Mismatches Python/Stan implementation (e.g., type of bernoulli(0.5)?)
Stan to NumPyro

NumPyro is impressively fast but:
- Only works on purely functional code
- Control structures (e.g., loops) are still experimental
- Limited support of dynamic features (e.g., dynamic slices)

data {
    int N;
    int[N]<lower=0, upper=1> x;
}

parameter {
    real<lower=0, upper=1> z;
}

model {
    z ~ beta(1, 1);
    for i in 1:N
        x[i] ~ bernoulli(z);
}

def model(N, x):
    z = sample(beta(1,1))
    def fori__0(i, acc):
        observe(bernoulli(z), x[i-1])
    fori_loop(1, N+1, fori__0, None)
Experimental Evaluation

RQ1: Can we compile and run all Stan models?

RQ2: What is the impact of the compilation on accuracy?

RQ3: What is the impact of the compilation on speed?
Can we compile and run all models?

stan-dev/example-models
- 531 models
- Errors: missing truncations

<table>
<thead>
<tr>
<th></th>
<th>compr</th>
<th>mixed</th>
<th>gen</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>522</td>
<td>522</td>
<td>166</td>
</tr>
</tbody>
</table>

stan-dev/posteriordb
- 98 pairs (model, data)
- Run 1 inference step
- Errors: missing library functions (e.g., ode solver)
- NumPyro: missing dynamic features

<table>
<thead>
<tr>
<th></th>
<th>compr</th>
<th>mixed</th>
<th>gen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pyro</td>
<td>88</td>
<td>88</td>
<td>36</td>
</tr>
<tr>
<td>NumPyro</td>
<td>83</td>
<td>83</td>
<td>35</td>
</tr>
</tbody>
</table>
Impact on accuracy and speed?

stan-dev/posteriordb
- 49 tuples (model, data, reference draws, configurations)
- Configuration: iterations, warmups, chains, thinning
- Stan regression test: $|\text{mean}(\theta_{\text{ref}}) - \text{mean}(\theta)| < 0.3 \ \text{stddev}(\theta_{\text{ref}})$
- 31 models pass the accuracy test with Stan
- 2 models have a very high runtime relative std (> 1)

Experience:
- NUTS with the same configuration for Stan, Pyro, and NumPyro
- Success if successful with given seed
- Average runtime over the 5 runs with different seeds
<table>
<thead>
<tr>
<th>Model</th>
<th>Dataset</th>
<th>Stan</th>
<th>Pyro_Comp.</th>
<th>NumPyro_Compr.</th>
<th>NumPyro_Mixed</th>
<th>NumPyro_Generic</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>arK</td>
<td>arK</td>
<td>✓</td>
<td>00:00:57</td>
<td>39:23:54</td>
<td>✓</td>
<td>✓</td>
<td>1.43</td>
</tr>
<tr>
<td>dogs</td>
<td>dogs</td>
<td>✓</td>
<td>00:01:05</td>
<td>24:58:30</td>
<td>✓</td>
<td>✓</td>
<td>0.17</td>
</tr>
<tr>
<td>dogs_log</td>
<td>dogs</td>
<td>✓</td>
<td>00:00:32</td>
<td>21:52:11</td>
<td>✓</td>
<td>✓</td>
<td>0.13</td>
</tr>
<tr>
<td>earn_height</td>
<td>earnings</td>
<td>✓</td>
<td>00:01:17</td>
<td>00:52:06</td>
<td>✓</td>
<td>✓</td>
<td>5.34</td>
</tr>
<tr>
<td>eight_schools_centered</td>
<td>eight_schools</td>
<td>✓</td>
<td>00:00:02</td>
<td>00:36:37</td>
<td>✓</td>
<td>✓</td>
<td>0.34</td>
</tr>
<tr>
<td>eight_schools_noncentered</td>
<td>eight_schools</td>
<td>✓</td>
<td>00:00:01</td>
<td>00:12:53</td>
<td>✓</td>
<td>✓</td>
<td>0.18</td>
</tr>
<tr>
<td>garch11</td>
<td>garch</td>
<td>✓</td>
<td>00:00:16</td>
<td>11:16:26</td>
<td>✓</td>
<td>✓</td>
<td>0.21</td>
</tr>
<tr>
<td>gp_regr</td>
<td>gp_pois_regr</td>
<td>✓</td>
<td>00:00:02</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>0.34</td>
</tr>
<tr>
<td>hmm_drive_0</td>
<td>bbball_drive_event_0</td>
<td>✓</td>
<td>00:03:47</td>
<td>69:11:24</td>
<td>✓</td>
<td>✓</td>
<td>0.15</td>
</tr>
<tr>
<td>hmm_example</td>
<td>hmm_example</td>
<td>✓</td>
<td>00:00:28</td>
<td>08:02:47</td>
<td>✓</td>
<td>✓</td>
<td>0.45</td>
</tr>
<tr>
<td>kidscore_interaction</td>
<td>kidiq</td>
<td>✓</td>
<td>00:01:43</td>
<td>01:26:07</td>
<td>✓</td>
<td>✓</td>
<td>7.76</td>
</tr>
<tr>
<td>kidscore_interaction_c2</td>
<td>kidiq_with_mom_work</td>
<td>✓</td>
<td>00:00:10</td>
<td>00:08:08</td>
<td>✓</td>
<td>✓</td>
<td>1.64</td>
</tr>
<tr>
<td>kidscore_mom_work</td>
<td>kidiq_with_mom_work</td>
<td>✓</td>
<td>00:00:14</td>
<td>00:10:23</td>
<td>✓</td>
<td>✓</td>
<td>2.12</td>
</tr>
<tr>
<td>kidscore_momhs</td>
<td>kidiq</td>
<td>✓</td>
<td>00:00:05</td>
<td>00:10:02</td>
<td>✓</td>
<td>✓</td>
<td>0.86</td>
</tr>
<tr>
<td>kidscore_momhsiq</td>
<td>kidiq</td>
<td>✓</td>
<td>00:00:28</td>
<td>00:32:28</td>
<td>✓</td>
<td>✓</td>
<td>3.40</td>
</tr>
<tr>
<td>kidscore_momiq</td>
<td>kidiq</td>
<td>✓</td>
<td>00:00:13</td>
<td>00:21:47</td>
<td>✓</td>
<td>✓</td>
<td>1.81</td>
</tr>
<tr>
<td>kilpisjarvi</td>
<td>kilpisjarvi_mod</td>
<td>✓</td>
<td>00:00:60</td>
<td>10:46:20</td>
<td>✓</td>
<td>✓</td>
<td>2.88</td>
</tr>
<tr>
<td>log10earn_height</td>
<td>earnings</td>
<td>✓</td>
<td>00:01:22</td>
<td>00:52:30</td>
<td>✓</td>
<td>✓</td>
<td>5.32</td>
</tr>
<tr>
<td>logearn_height</td>
<td>earnings</td>
<td>✓</td>
<td>00:01:20</td>
<td>00:44:05</td>
<td>✓</td>
<td>✓</td>
<td>5.38</td>
</tr>
<tr>
<td>logearn_height_male</td>
<td>earnings</td>
<td>✓</td>
<td>00:03:42</td>
<td>01:18:10</td>
<td>✓</td>
<td>✓</td>
<td>9.76</td>
</tr>
<tr>
<td>logearn_interaction</td>
<td>earnings</td>
<td>✓</td>
<td>00:09:05</td>
<td>02:05:15</td>
<td>✓</td>
<td>✓</td>
<td>13.76</td>
</tr>
<tr>
<td>logearn_logheight_male</td>
<td>earnings</td>
<td>✓</td>
<td>00:14:33</td>
<td>05:14:46</td>
<td>✓</td>
<td>✓</td>
<td>12.51</td>
</tr>
<tr>
<td>logmesquite</td>
<td>mesquite</td>
<td>✓</td>
<td>00:00:08</td>
<td>00:23:48</td>
<td>✓</td>
<td>✓</td>
<td>1.10</td>
</tr>
<tr>
<td>logmesquite_logva</td>
<td>mesquite</td>
<td>✓</td>
<td>00:00:05</td>
<td>00:22:44</td>
<td>✓</td>
<td>✓</td>
<td>0.82</td>
</tr>
<tr>
<td>logmesquite_logvas</td>
<td>mesquite</td>
<td>✓</td>
<td>00:00:14</td>
<td>00:44:15</td>
<td>✓</td>
<td>✓</td>
<td>1.76</td>
</tr>
<tr>
<td>logmesquite_logvash</td>
<td>mesquite</td>
<td>✓</td>
<td>00:00:12</td>
<td>00:40:41</td>
<td>✓</td>
<td>✓</td>
<td>1.58</td>
</tr>
<tr>
<td>mesquite</td>
<td>mesquite</td>
<td>✓</td>
<td>00:00:16</td>
<td>00:55:36</td>
<td>✓</td>
<td>✓</td>
<td>1.95</td>
</tr>
<tr>
<td>nes</td>
<td>nes1980</td>
<td>✓</td>
<td>00:03:55</td>
<td>00:47:02</td>
<td>✓</td>
<td>✓</td>
<td>13.19</td>
</tr>
<tr>
<td>nes</td>
<td>nes1976</td>
<td>✓</td>
<td>00:07:13</td>
<td>00:45:15</td>
<td>✓</td>
<td>✓</td>
<td>21.30</td>
</tr>
<tr>
<td>nes</td>
<td>nes1984</td>
<td>✓</td>
<td>00:07:14</td>
<td>00:40:15</td>
<td>✓</td>
<td>✓</td>
<td>18.15</td>
</tr>
<tr>
<td>nes</td>
<td>nes1972</td>
<td>✓</td>
<td>00:07:49</td>
<td>00:40:30</td>
<td>✓</td>
<td>✓</td>
<td>18.54</td>
</tr>
<tr>
<td>nes</td>
<td>nes2000</td>
<td>✓</td>
<td>00:02:40</td>
<td>00:48:24</td>
<td>✓</td>
<td>✓</td>
<td>11.71</td>
</tr>
<tr>
<td>nes</td>
<td>nes1996</td>
<td>✓</td>
<td>00:06:37</td>
<td>00:46:08</td>
<td>✓</td>
<td>✓</td>
<td>18.71</td>
</tr>
<tr>
<td>nes</td>
<td>nes1988</td>
<td>✓</td>
<td>00:05:30</td>
<td>00:41:48</td>
<td>✓</td>
<td>✓</td>
<td>15.69</td>
</tr>
<tr>
<td>one_comp_mm_elim_abs</td>
<td>one_comp_mm_elim_abs</td>
<td>✓</td>
<td>00:15:58</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

✓ match, ✗ mismatch, ✗ error. Durations are reported in **hh:mm:ss** format. **Speedup** = **Stan** / **NumPyro Compr.**

**Average speedup:** 2.3x

(geometric mean)

Without experimental loops:

**Average speedup:** 3.4x
<table>
<thead>
<tr>
<th>Model</th>
<th>Dataset</th>
<th>Stan</th>
<th>Compr.</th>
<th>NumPyro</th>
<th>Gener.</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>arK</td>
<td>arK</td>
<td>✓ 00:00:57</td>
<td>✓ 39:23:54</td>
<td>✓ 00:00:40</td>
<td>✓ 00:00:35</td>
<td>00:00:39</td>
</tr>
<tr>
<td>dogs</td>
<td>dogs</td>
<td>✓ 00:01:05</td>
<td>✓ 24:58:30</td>
<td>✓ 00:00:20</td>
<td>✓ 00:00:22</td>
<td>00:00:21</td>
</tr>
<tr>
<td>dogs_log</td>
<td>dogs</td>
<td>✓ 00:00:32</td>
<td>✓ 21:52:11</td>
<td>✓ 00:03:58</td>
<td>✓ 00:03:39</td>
<td>×</td>
</tr>
<tr>
<td>earn_height</td>
<td>earnings</td>
<td>✓ 00:01:17</td>
<td>✓ 00:52:06</td>
<td>✓ 00:00:14</td>
<td>✓ 00:00:14</td>
<td>×</td>
</tr>
<tr>
<td>eight_schools_centered</td>
<td>eight_schools</td>
<td>✓ 00:00:02</td>
<td>✓ 00:36:37</td>
<td>✓ 00:00:06</td>
<td>✓ 00:00:06</td>
<td>✓ 00:00:06</td>
</tr>
<tr>
<td>eight_schools_noncentered</td>
<td>eight_schools</td>
<td>✓ 00:00:01</td>
<td>✓ 00:42:53</td>
<td>✓ 00:00:06</td>
<td>✓ 00:00:06</td>
<td>✓ 00:00:06</td>
</tr>
<tr>
<td>garch11</td>
<td>garch</td>
<td>✓ 00:00:16</td>
<td>□ 11:16:26</td>
<td>□ 00:01:25</td>
<td>□ 00:01:21</td>
<td>×</td>
</tr>
<tr>
<td>gp_regr</td>
<td>gp_pois_regr</td>
<td>✓ 00:00:02</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hmm_drive_0</td>
<td>bball_drive_event_0</td>
<td>✓ 00:03:47</td>
<td>✓ 69:11:24</td>
<td>✓ 00:25:38</td>
<td>✓ 00:25:15</td>
<td>×</td>
</tr>
<tr>
<td>hmm_example</td>
<td>hmm_example</td>
<td>✓ 00:00:28</td>
<td>✓ 00:02:47</td>
<td>✓ 00:00:03</td>
<td>✓ 00:00:58</td>
<td></td>
</tr>
<tr>
<td>kidscore_interaction</td>
<td>kidsq</td>
<td>✓ 00:00:43</td>
<td>✓ 01:26:47</td>
<td>✓ 00:00:13</td>
<td>✓ 00:00:12</td>
<td>×</td>
</tr>
<tr>
<td>kidscore_interaction_c2</td>
<td>kidsq</td>
<td>✓ 00:00:49</td>
<td>✓ 01:48:48</td>
<td>✓ 00:00:06</td>
<td>✓ 00:00:06</td>
<td>×</td>
</tr>
<tr>
<td>kidscore_mom_work</td>
<td>kidsq_with_mom_work</td>
<td>✓ 00:00:19</td>
<td>✓ 00:16:23</td>
<td>✓ 00:00:07</td>
<td>✓ 00:00:06</td>
<td>×</td>
</tr>
<tr>
<td>kidscore_momhs</td>
<td>kidsq</td>
<td>✓ 00:00:05</td>
<td>✓ 00:10:02</td>
<td>✓ 00:00:06</td>
<td>✓ 00:00:05</td>
<td>×</td>
</tr>
<tr>
<td>kidscore_momhsiq</td>
<td>kidsq</td>
<td>✓ 00:00:05</td>
<td>✓ 00:10:02</td>
<td>✓ 00:00:06</td>
<td>✓ 00:00:05</td>
<td>×</td>
</tr>
<tr>
<td>kidscore_momiq</td>
<td>kidsq</td>
<td>✓ 00:00:05</td>
<td>✓ 00:10:02</td>
<td>✓ 00:00:06</td>
<td>✓ 00:00:05</td>
<td>×</td>
</tr>
<tr>
<td>kilpisjarvi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>log10earn_height</td>
<td>earnings</td>
<td>✓ 00:01:22</td>
<td>✓ 00:32:30</td>
<td>✓ 00:00:13</td>
<td>✓ 00:00:14</td>
<td>×</td>
</tr>
<tr>
<td>logearn_height</td>
<td>earnings</td>
<td>✓ 00:01:20</td>
<td>✓ 00:44:05</td>
<td>✓ 00:00:15</td>
<td>✓ 00:00:14</td>
<td>×</td>
</tr>
<tr>
<td>logearn_height_male</td>
<td>earnings</td>
<td>✓ 00:03:42</td>
<td>✓ 01:18:10</td>
<td>✓ 00:00:23</td>
<td>✓ 00:00:21</td>
<td>×</td>
</tr>
<tr>
<td>logearngeration</td>
<td>earnings</td>
<td>✓ 00:09:05</td>
<td>✓ 02:05:15</td>
<td>✓ 00:00:40</td>
<td>✓ 00:00:38</td>
<td></td>
</tr>
<tr>
<td>logearng_height_male</td>
<td>earnings</td>
<td>✓ 00:14:33</td>
<td>✓ 01:44:46</td>
<td>✓ 00:01:07</td>
<td>✓ 00:01:07</td>
<td></td>
</tr>
<tr>
<td>logmesquite</td>
<td>mesquite</td>
<td>✓ 00:00:08</td>
<td>✓ 00:23:48</td>
<td>✓ 00:00:07</td>
<td>✓ 00:00:06</td>
<td>×</td>
</tr>
<tr>
<td>logmesquite_logva</td>
<td>mesquite</td>
<td>✓ 00:00:05</td>
<td>✓ 00:22:44</td>
<td>✓ 00:00:06</td>
<td>✓ 00:00:06</td>
<td>×</td>
</tr>
<tr>
<td>logmesquite_logvsq</td>
<td>mesquite</td>
<td>✓ 00:00:14</td>
<td>✓ 00:44:15</td>
<td>✓ 00:00:08</td>
<td>✓ 00:00:08</td>
<td>×</td>
</tr>
<tr>
<td>logmesquite_logvash</td>
<td>mesquite</td>
<td>✓ 00:00:12</td>
<td>✓ 00:40:41</td>
<td>✓ 00:00:07</td>
<td>✓ 00:00:07</td>
<td>×</td>
</tr>
<tr>
<td>mesquite</td>
<td>mesquite</td>
<td>✓ 00:00:16</td>
<td>✓ 00:55:36</td>
<td>✓ 00:00:08</td>
<td>✓ 00:00:07</td>
<td>×</td>
</tr>
<tr>
<td>nes</td>
<td>nes1980</td>
<td>✓ 00:03:55</td>
<td>✓ 00:47:02</td>
<td>✓ 00:00:16</td>
<td>✓ 00:00:15</td>
<td>×</td>
</tr>
<tr>
<td>nes</td>
<td>nes1976</td>
<td>✓ 00:07:13</td>
<td>✓ 00:45:15</td>
<td>✓ 00:00:20</td>
<td>✓ 00:00:19</td>
<td>×</td>
</tr>
<tr>
<td>nes</td>
<td>nes1984</td>
<td>✓ 00:07:14</td>
<td>✓ 00:40:15</td>
<td>✓ 00:00:24</td>
<td>✓ 00:00:22</td>
<td>×</td>
</tr>
<tr>
<td>nes</td>
<td>nes1972</td>
<td>✓ 00:07:49</td>
<td>✓ 00:40:30</td>
<td>✓ 00:00:25</td>
<td>✓ 00:00:24</td>
<td>×</td>
</tr>
<tr>
<td>nes</td>
<td>nes2000</td>
<td>✓ 00:02:40</td>
<td>✓ 00:48:24</td>
<td>✓ 00:00:14</td>
<td>✓ 00:00:13</td>
<td>×</td>
</tr>
<tr>
<td>nes</td>
<td>nes1996</td>
<td>✓ 00:06:37</td>
<td>✓ 00:46:08</td>
<td>✓ 00:00:21</td>
<td>✓ 00:00:20</td>
<td>×</td>
</tr>
<tr>
<td>nes</td>
<td>nes1988</td>
<td>✓ 00:05:30</td>
<td>✓ 00:41:48</td>
<td>✓ 00:00:21</td>
<td>✓ 00:00:21</td>
<td>×</td>
</tr>
<tr>
<td>one Comp_mm Elim abs</td>
<td>one Comp_mm Elim abs</td>
<td>✓ 00:15:58</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

✓ match, ○ mismatch, × error. Durations are reported in hh:mm:ss format. **Speedup** = **Stan** / **NumPyro Compr.**

Average speedup: 2.3x (geometric mean)

Constraint between parameters:

real<lower=0, upper=1> alpha1;
real<lower=0, upper=(1-alpha1)> beta1;

Without experimental loops:
Average speedup: 3.4x
<table>
<thead>
<tr>
<th>Model</th>
<th>Dataset</th>
<th>Pyro</th>
<th>NumPyro</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STAN</td>
<td>Compr.</td>
<td>Mixed</td>
</tr>
<tr>
<td>arK</td>
<td>arK</td>
<td>✓ 00:00:57</td>
<td>✓ 39:23:54</td>
</tr>
<tr>
<td>dogs</td>
<td>dogs</td>
<td>✓ 00:01:05</td>
<td>✓ 24:58:30</td>
</tr>
<tr>
<td>dogs_log</td>
<td>dogs</td>
<td>✓ 00:00:52</td>
<td>✓ 21:52:11</td>
</tr>
<tr>
<td>earn_height</td>
<td>earnings</td>
<td>✓ 00:01:17</td>
<td>✓ 00:52:06</td>
</tr>
<tr>
<td>eight_schools_centered</td>
<td>eight_schools</td>
<td>✓ 00:00:02</td>
<td>✓ 00:36:37</td>
</tr>
<tr>
<td>eight_schools_noncentered</td>
<td>eight_schools</td>
<td>✓ 00:00:01</td>
<td>✓ 00:12:53</td>
</tr>
<tr>
<td>garch11</td>
<td>garch</td>
<td>✓ 00:00:16</td>
<td>✓ 11:16:26</td>
</tr>
<tr>
<td>gp_regr</td>
<td>gp_pois_regr</td>
<td>✓ 00:00:02</td>
<td>x x x x x</td>
</tr>
<tr>
<td>hmm_drive_0</td>
<td>bball_drive_event_0</td>
<td>✓ 00:03:47</td>
<td>✓ 69:11:24</td>
</tr>
<tr>
<td>hmm_example</td>
<td>hmn_example</td>
<td>✓ 00:00:28</td>
<td>✓ 00:02:47</td>
</tr>
<tr>
<td>kidscore_interaction</td>
<td>kidiq</td>
<td>✓ 00:01:43</td>
<td>✓ 01:26:07</td>
</tr>
<tr>
<td>kidscore_interaction_c2</td>
<td>kidiq_with_mom_work</td>
<td>✓ 00:00:10</td>
<td>✓ 00:08:08</td>
</tr>
<tr>
<td>kidscore_mom_work</td>
<td>kidiq_with_mom_work</td>
<td>✓ 00:00:14</td>
<td>✓ 00:10:23</td>
</tr>
<tr>
<td>kidscore_momhs</td>
<td>ds</td>
<td>✓ 00:00:32</td>
<td>✓ 00:46:28</td>
</tr>
<tr>
<td>kidscore_momhsq</td>
<td>kidsq</td>
<td>✓ 00:00:29</td>
<td>✓ 00:32:28</td>
</tr>
<tr>
<td>kidscore_momiq</td>
<td>kidiq</td>
<td>✓ 00:00:13</td>
<td>✓ 00:21:47</td>
</tr>
<tr>
<td>kilpisjarvi</td>
<td>kilpisjarvi_mod</td>
<td>✓ 00:00:60</td>
<td>✓ 10:46:20</td>
</tr>
<tr>
<td>log10earn_height</td>
<td>earnings</td>
<td>✓ 00:01:22</td>
<td>✓ 00:52:30</td>
</tr>
<tr>
<td>logearn_height</td>
<td>earnings</td>
<td>✓ 00:01:20</td>
<td>✓ 00:44:05</td>
</tr>
<tr>
<td>logearn_height_male</td>
<td>earnings</td>
<td>✓ 00:03:42</td>
<td>✓ 01:18:10</td>
</tr>
<tr>
<td>logearn_interaction</td>
<td>earnings</td>
<td>✓ 00:09:05</td>
<td>✓ 02:05:15</td>
</tr>
<tr>
<td>logearlog_height_male</td>
<td>earnings</td>
<td>✓ 00:14:33</td>
<td>✓ 04:04:46</td>
</tr>
<tr>
<td>logmesquite</td>
<td>mesquite</td>
<td>✓ 00:00:08</td>
<td>✓ 00:23:48</td>
</tr>
<tr>
<td>logmesquite_logva</td>
<td>mesquite</td>
<td>✓ 00:00:05</td>
<td>✓ 00:22:44</td>
</tr>
<tr>
<td>logmesquite_logvas</td>
<td>mesquite</td>
<td>✓ 00:00:14</td>
<td>✓ 00:44:15</td>
</tr>
<tr>
<td>logmesquite_logvash</td>
<td>mesquite</td>
<td>✓ 00:00:12</td>
<td>✓ 00:40:41</td>
</tr>
<tr>
<td>mesquite</td>
<td>mesquite</td>
<td>✓ 00:00:16</td>
<td>✓ 00:55:36</td>
</tr>
<tr>
<td>nes</td>
<td>nes1980</td>
<td>✓ 00:03:55</td>
<td>✓ 00:47:02</td>
</tr>
<tr>
<td>nes</td>
<td>nes1976</td>
<td>✓ 00:07:13</td>
<td>✓ 00:45:15</td>
</tr>
<tr>
<td>nes</td>
<td>nes1984</td>
<td>✓ 00:07:14</td>
<td>✓ 00:40:15</td>
</tr>
<tr>
<td>nes</td>
<td>nes1972</td>
<td>✓ 00:07:49</td>
<td>✓ 00:40:30</td>
</tr>
<tr>
<td>nes</td>
<td>nes2000</td>
<td>✓ 00:02:40</td>
<td>✓ 00:48:24</td>
</tr>
<tr>
<td>nes</td>
<td>nes1996</td>
<td>✓ 00:06:37</td>
<td>✓ 00:46:08</td>
</tr>
<tr>
<td>nes</td>
<td>nes1988</td>
<td>✓ 00:05:30</td>
<td>✓ 00:41:48</td>
</tr>
<tr>
<td>one_comp_mm_elim_abs</td>
<td>one_comp_mm_elim_abs</td>
<td>✓ 00:15:58</td>
<td>x x x x x</td>
</tr>
</tbody>
</table>

✓ match, ❌ mismatch, ❌ error. Durations are reported in **hh:mm:ss** format. **Speedup = STAN / NumPyro Compr.**

**Average speedup:** **2.3x** (geometric mean)

**Without experimental loops:** **Average speedup:** **3.4x**

---

cov_exp_quo not implemented
<table>
<thead>
<tr>
<th>Model</th>
<th>Dataset</th>
<th>Stan</th>
<th>Compr.</th>
<th>Compr.</th>
<th>Mixed</th>
<th>Gener.</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>arK</td>
<td>arK</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1.43</td>
</tr>
<tr>
<td>dogs</td>
<td>dogs</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>0.17</td>
</tr>
<tr>
<td>dogs_log</td>
<td>dogs</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>0.13</td>
</tr>
<tr>
<td>earn_height</td>
<td>eight_schools</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>5.34</td>
</tr>
<tr>
<td>eight_schools_centered</td>
<td>eight_schools_noncentered</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>0.34</td>
</tr>
<tr>
<td>garch1</td>
<td>garch</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>0.18</td>
</tr>
<tr>
<td>gp_regr</td>
<td>gp_pois_regr</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>hmm_drive_0</td>
<td>bball_drive_event_0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>0.15</td>
</tr>
<tr>
<td>hmm_example</td>
<td>hmm_example</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>0.45</td>
</tr>
<tr>
<td>kidscore_interaction</td>
<td>kidsiq</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>7.76</td>
</tr>
<tr>
<td>kidscore_interaction_c2</td>
<td>kidsiq_with_mom_work</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1.64</td>
</tr>
<tr>
<td>kidscore_monwork</td>
<td>kidsiq_with_mom_work</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2.12</td>
</tr>
<tr>
<td>kidscore_momhs</td>
<td>kidsiq</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>0.86</td>
</tr>
<tr>
<td>kidscore_momhsiq</td>
<td>kidsiq</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>3.40</td>
</tr>
<tr>
<td>kidscore_momiq</td>
<td>kidsiq</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1.81</td>
</tr>
<tr>
<td>kilpisjarvi</td>
<td>kilpisjarvi_mod</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2.88</td>
</tr>
<tr>
<td>log10earn_height</td>
<td>earnings</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>5.32</td>
</tr>
<tr>
<td>log10earn_height_male</td>
<td>earnings</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>5.38</td>
</tr>
<tr>
<td>logearn_height</td>
<td>earnings</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>9.76</td>
</tr>
<tr>
<td>logearn_height_male</td>
<td>earnings</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>13.76</td>
</tr>
<tr>
<td>logearn_logheight_male</td>
<td>earnings</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>12.51</td>
</tr>
<tr>
<td>logmesquite</td>
<td>mesquite</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1.10</td>
</tr>
<tr>
<td>logmesquite_logva</td>
<td>mesquite</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>0.82</td>
</tr>
<tr>
<td>logmesquite_logvgs</td>
<td>mesquite</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1.76</td>
</tr>
<tr>
<td>logmesquite_logvash</td>
<td>mesquite</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1.58</td>
</tr>
<tr>
<td>mesquite</td>
<td>mesquite</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1.95</td>
</tr>
<tr>
<td>nes</td>
<td>nes1980</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>13.19</td>
</tr>
<tr>
<td>nes</td>
<td>nes1976</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>21.30</td>
</tr>
<tr>
<td>nes</td>
<td>nes1984</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>18.15</td>
</tr>
<tr>
<td>nes</td>
<td>nes1972</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>18.54</td>
</tr>
<tr>
<td>nes</td>
<td>nes2000</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>11.71</td>
</tr>
<tr>
<td>nes</td>
<td>nes1996</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>18.71</td>
</tr>
<tr>
<td>nes</td>
<td>nes1988</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>15.69</td>
</tr>
</tbody>
</table>

**Average speedup: 2.3x**

**Without experimental loops: Average speedup: 3.4x**

**Missing ODE solver**

✓ match, ☞ mismatch, x error. Durations are reported in hh:mm:ss format. Speedup = Stan / NumPyro Compr.
Extensions

Stan to (Num)Pyro
Stochastic Variational Inference (SVI)

\[ p(z \mid x) = \frac{p(x \mid z)p(z)}{p(x)} = \frac{p(x \mid z)p(z)}{\int_z p(x \mid z)p(z)dz} \]

Metrics: Kullback-Leibler divergence

\[ KL(q_\phi(z \mid x) \mid \mid p(z \mid x)) \]

Optimize ELBO:

\[ \phi^* = \arg \max_\phi \mathbb{E}_{q_\phi(z \mid x)} \log p(x \mid z) - KL(q_\phi(z \mid x) \mid \mid p(z)) \]

Blei et al. 2017
Explicit Variational Guides

parameters {
  real cluster;
  real theta;
}
model {
  real mu;
  cluster ~ normal(0, 1);
  if (cluster > 0) mu = 20;
  else mu = 0;
  theta ~ normal(mu, 1);
}
guide parameters {
  real m1; real m2;
  real<lower=0> s1;
  real<lower=0> s2;
}
guide {
  cluster ~ normal(0, 1);
  if (cluster > 0) theta ~ normal(m1, s1);
  else theta ~ normal(m2, s2);
}

We need to be able to sample from the guide without inference

Generative translation + static constraints
Variational Auto-Encoder

networks {
  real[,] decoder(real[,] x);
  real[,] encoder(int[,] x);
}
data {
  int nz;
  int<lower=0, upper=1> x[28, 28];
}
parameters {
  real z[nz];
}
model {
  real mu[28, 28];
  z ~ normal(0, 1);
  mu = decoder(z);
  x ~ bernoulli(mu);
}
guide {
  real encoded[2, nz] = encoder(x);
  real mu_z[nz] = encoded[1];
  real sigma_z[nz] = encoded[2];
  z ~ normal(mu_z, sigma_z);
}
Bayesian Neural Networks

networks {
    vector mlp(int[,,] imgs);
}
data {
    int batch_size; int nx; int nh; int ny;
    int <lower=0, upper=1> imgs[28,28,batch_size];
    int <lower=1, upper=10> labels[batch_size];
}
parameters {
    real mlp.l1.weight[nh, nx]; real mlp.l1.bias[nh];
    real mlp.l2.weight[ny, nh]; real mlp.l2.bias[ny];
}
model {
    vector[batch_size] lambda;
    mlp.l1.weight ~ normal(0, 1); mlp.l1.bias ~ normal(0, 1);
    mlp.l2.weight ~ normal(0, 1); mlp.l2.bias ~ normal(0, 1);
    lambda = mlp(imgs);
    labels ~ categorical_logit(lambda);
}
guide parameters {
    real w1_mu[nh, nx]; real w1_sigma[nh, nx];
    real b1_mu[nh]; real b1_sigma[nh];
    real w2_mu[ny, nh]; real w2_sigma[ny, nh];
    real b2_mu[ny]; real b2_sigma[ny];
}
guide {
    mlp.l1.weight ~ normal(w1_mu, exp(w1_sigma)); mlp.l1.bias ~ normal(b1_mu, exp(b1_sigma));
    mlp.l2.weight ~ normal(w2_mu, exp(w2_sigma)); mlp.l2.bias ~ normal(b2_mu, exp(b2_sigma));
}
Conclusion

Compiling Stan to generative probabilistic languages
- Non-generative features
- Add priors and compile \( \sim \) to \textit{observe} statement
- Correctness proof
- Experimental evaluation with Stanc3 and stan-dev benchmarks

Extensions
- Explicit variational guides
- Interaction with neural networks (e.g., VAE)
- Bayesian neural networks (e.g., MLP)

Next?
- Inference on discrete latent variables with enumeration
- Automatic SVI guide synthesis
- Impact of the compilation on other inference scheme
- Compilation of generative programs to Stan?

https://github.com/deepppl
References

Stan: A probabilistic Programming Language

A Brief History of Stan
D. Lee, StanCon 2018

Hierarchical Partial Pooling for Repeated Binary Trials
B. Carpenter 2016, mc-stan.org

Probabilistic programming with densities in SlicStan: efficient, flexible, and deterministic
M.I. Gorinova, A.D. Gordon, C.A. Sutton. POPL 2019

Commutative Semantics for Probabilistic Programming
S. Staton. ESOP 2017

Variational Inference: A Review for Statisticians.