

SynthStats ARIA Milestone 3

Quantitative evaluation, scaling, and public alpha release

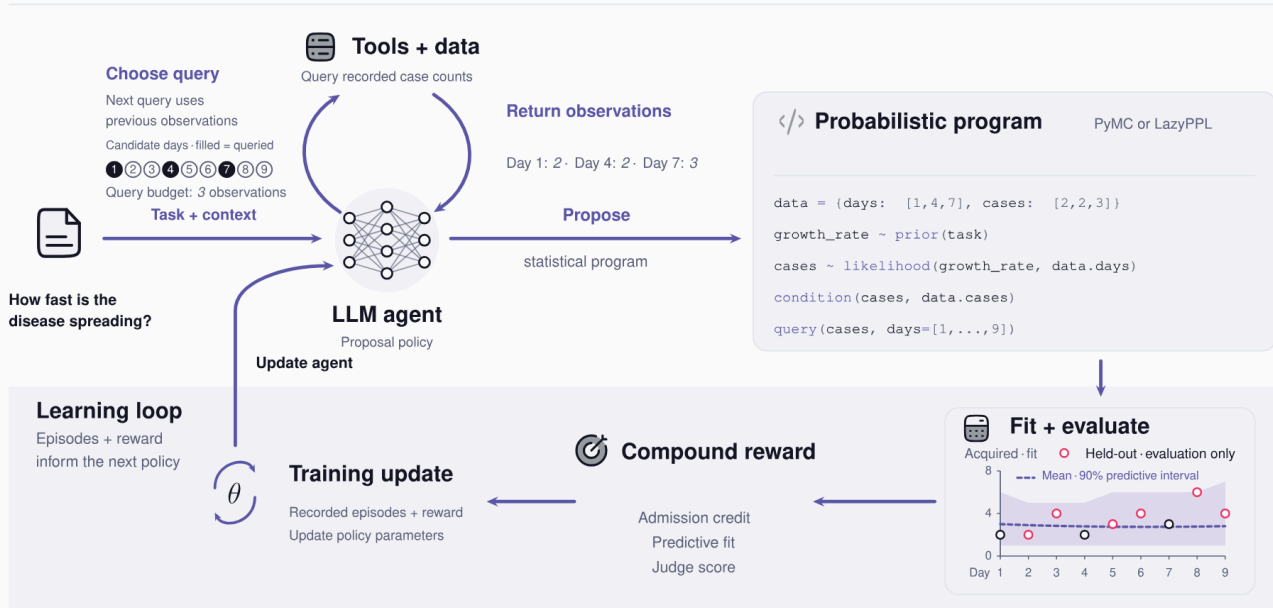
ARIA Brief

Given a dugong's age, what lengths should a statistical model predict? We asked a language model to write a program that answers this question from observed data. The language models in this report write PyMC programs: Python descriptions of probability models.

We ask for a program with AI safety in mind: a statistician who opens the dugong program below sees, in a few lines of Python, that it makes length level off with age and uses the same observation-noise scale at every age, and can object to either before relying on its predictions. Before submitting a program, the AI may also request observations and run code of its own.

SynthStats · Architecture overview

Tool choices, program proposals and learning from reward



The agent chooses a query, receives observations and proposes a statistical program. A program that passes the checks and can be scored receives admission credit and a score for its predictions on unseen observations. We combine these in the reward used to update the agent; one separate training comparison also adds a language-model judge's score.

Quantitative evaluation

Two modelling demonstrations

All eight dugong-growth programs written by the prompted model Qwen3.6-35B-A3B scored above the floor model. Two scored above the simulator-informed reference by more than two Monte Carlo standard errors (the errors from estimating scores with 200 posterior draws) on this one simulated dataset of 20 held-out observations. Single attempts 2 and 4 exceeded the reference by 1.4059 (Monte Carlo standard error 0.1048) and 1.3761 (0.1051). Loop proposal 4 had a positive gain of 0.0907 (0.1355), within two Monte Carlo standard errors of the reference. All six scored disease-spread programs also beat the floor model, but none beat the reference.

Each comparison here concerns one simulated dataset per environment in BoxingGym, a benchmark of scientific environments: the model receives an English description of the task and 80 observations for fitting, while another 20 responses are held out to score its predictions against a simulator-informed reference program and a floor model that uses one predictive distribution at every input. Predictions are scored by *held-out log predictive density*, defined as the sum of the log predictive densities assigned to unseen responses (higher values within a task mean greater predictive density at those responses).

For each demonstration, we compared four independent attempts with four programs produced in a critic loop: two proposals in each of two rounds. The critic reviewed the proposals and gave feedback for the next round.

Monte Carlo standard errors (MCSEs) use batch means of each draw's delta-method influence on the total held-out log score, in ten consecutive batches of 20 draws; MCSEs for differences combine the two fits in quadrature under the independent-fit approximation.

Scores below are shown as estimate $\pm$ MCSE.

Demonstration result	Dugong growth	Disease spread
Reference held-out log predictive density	-4.6028 ± 0.1031	-45.6085 ± 0.0452
Floor held-out log predictive density	-14.0898 ± 0.1042	-262.1291 ± 0.5574
Model-written programs scored	8 of 8	6 of 8
Lowest and highest model-written score estimates	-7.4306 ± 0.1294 to -3.1969 ± 0.0190	-60.7518 ± 0.1383 to -59.8279 ± 0.2392
Model-written programs above floor / reference by more than two MCSEs	8 of 8 / 2 of 8; loop proposal 4 within two MCSEs of the reference	6 of 6 / 0 of 6
Two highest single-attempt score estimates	Single 2 and single 4; single 4 log predictive density -3.2267 ± 0.0204	Single 2 and single 1; single 1 log predictive density -59.9733 ± 0.2544
Highest loop score estimate	-4.5121 ± 0.0879 (proposal 4)	-59.9324 ± 0.1301 (proposal 1)
Model-written programs' coverage of 90% predictive intervals	16 to 18 of the held-out responses; both best single attempts cover 18	17 to 19 of the held-out responses

Generation and fitting costs	Dugong growth	Disease spread
Candidate programs, single attempts / two-round critic loop	4 (4 model calls) / 4 (2 per round + 2 critic calls = 6 model calls)	4 (4 model calls) / 4 (2 per round + 2 critic calls = 6 model calls)
Input tokens, single attempts / loop	13,612 / 46,558	8,672 / 35,536
Model-request seconds, single attempts	29.99	24.18
Final scoring time (separate runs, summed seconds), single attempts / loop proposals	75.51 / 116.02	58.96 / 64.28
Whole critic-loop elapsed seconds, including its own proposal fits	87.87	45.80
Further records and settings	Demonstration records (output tokens, model-request, scoring and sampler time)	Methods page (fitting and generation settings)

Checks before fitting

In PyMC, a single `pm.Potential` statement adds an arbitrary term to a model's log density, so a program that uses one can raise a reward computed from its likelihood without defining the intended probability model. The first check below rejects such a model before fitting, along with any model using the other constructions in its row; the other two test whether a program conditions on the observations that we supplied:

Check	What it guards against	Rule
Restricted PyMC profile: checks of the built model before fitting	<u>Likelihood reward</u> inflation without defining the intended probability model	Rejects custom densities (densities that the program defines itself), arbitrary terms added to the log density through <code>pm.Potential</code> , priors and branches that depend on PyMC data containers
Fabricated-data check	A model that does not condition on the supplied observations	Compares the size of the observations that the model declares with the size of the supplied data
Sensitivity check	A model that does not condition on the supplied observations	Compares log-density terms after shortening the input arrays

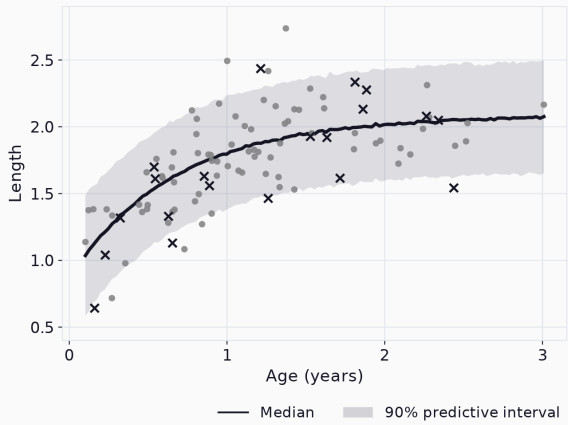
Programs and predictions

Dugong growth: similar curves, different programs

Qwen3.6-35B-A3B single attempt

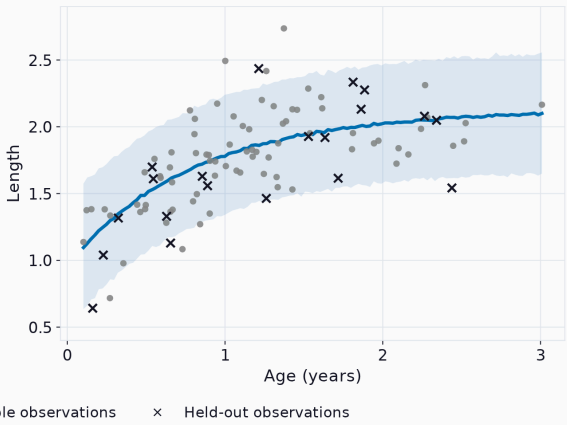
A Reference

Held-out score -4.6028 · Within 90% band: 15/20



B Single attempt 1

Held-out score -4.6292 · Within 90% band: 16/20



A Reference program excerpt

Source lines 13-17

```
13     alpha = pm.Normal("alpha", mu=2.0, sigma=0.2)
14     beta = pm.Normal("beta", mu=1.5, sigma=0.5)
15     lam = pm.Normal("lam", mu=0.4, sigma=0.5)
16     mu = alpha - beta * pm.math.abs(lam) ** age_data
17     pm.Normal("length", mu=mu, sigma=0.25, observed=y)
```

Normal noise scale fixed at 0.25.
Program file: 01-reference.py

B Single attempt 1 program excerpt

Source lines 27, 30, 34, 37, 41, 43

```
27     L_inf = pm.HalfNormal("L_inf", sigma=2.0)
28     [lines 28-29 omitted]
30     k = pm.HalfNormal("k", sigma=1.0)
31     [lines 31-33 omitted]
34     t0 = pm.Normal("t0", mu=-0.5, sigma=0.5)
35     [lines 35-36 omitted]
37     mu = L_inf * (1 - pm.math.exp(-k * (age - t0)))
38     [lines 38-40 omitted]
41     sigma = pm.HalfNormal("sigma", sigma=0.2)
42     [line 42 omitted]
43     length_obs = pm.Normal("length_obs", mu=mu, sigma=sigma, observed=length)
```

Normal noise scale inferred.
Program file: 03-single-attempt-1.py

The programs make similar slowing-growth predictions with different priors and noise assumptions: the reference fixes the noise scale; single attempt 1 infers it. The same 80 visible simulated observations inform each program. Their respective 90% bands contain 15/20 and 16/20 held-out points.

The symbol : marks an omission.

Dugong growth: circles mark the observations supplied to the model and crosses mark the held-out responses, with predictive medians (lines) and central 90% intervals of predictive draws (bands) for the reference model and single attempt 1. Single attempt 2 is shown in the listing below.

The programs behind the curves

Selected model statements.

Reference program

```
alpha = pm.Normal("alpha", mu=2.0, sigma=0.2)
beta = pm.Normal("beta", mu=1.5, sigma=0.5)
lam = pm.Normal("lam", mu=0.4, sigma=0.5)
mu = alpha - beta * pm.math.abs(lam) ** age_data
pm.Normal("length", mu=mu, sigma=0.25, observed=y)
```

First single-attempt program

```
L_inf = pm.HalfNormal("L_inf", sigma=2.0)
k = pm.HalfNormal("k", sigma=1.0)
t0 = pm.Normal("t0", mu=-0.5, sigma=0.5)
mu = L_inf * (1 - pm.math.exp(-k * (age - t0)))
sigma = pm.HalfNormal("sigma", sigma=0.2)
length_obs = pm.Normal("length_obs", mu=mu, sigma=sigma, observed=length)
```

[Full reference program](#) · [Full generated program](#)

Read the generated dugong program (single attempt 2)

```
import pymc as pm
import numpy as np

def model(data):
    age = data['age']
    length = data['length']

    with pm.Model() as dugong_model:
        # Priors for the von Bertalanffy growth parameters
        # L_infinity: asymptotic length
        L_inf = pm.HalfNormal('L_inf', sigma=2)

        # k: growth rate
        k = pm.HalfNormal('k', sigma=1)

        # Observation noise
        sigma = pm.HalfNormal('sigma', sigma=0.2)

        # Expected length
        # Using the von Bertalanffy growth function:  $L(t) = L_{\text{inf}} * (1 - \exp(-k * t))$ 
        mu = L_inf * (1 - pm.math.exp(-k * age))

        # Likelihood
        likelihood = pm.Normal('likelihood', mu=mu, sigma=sigma, observed=length)

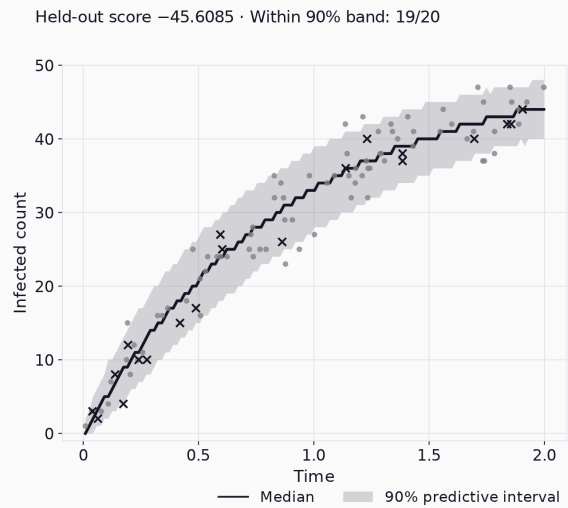
    return dugong_model
```

The critic reviewed the first-round dugong programs and recommended "Exploring alternative likelihoods such as Log-Normal if the data is strictly positive and skewed." The resulting Log-Normal proposal had the lowest dugong log predictive density among the model-written programs.

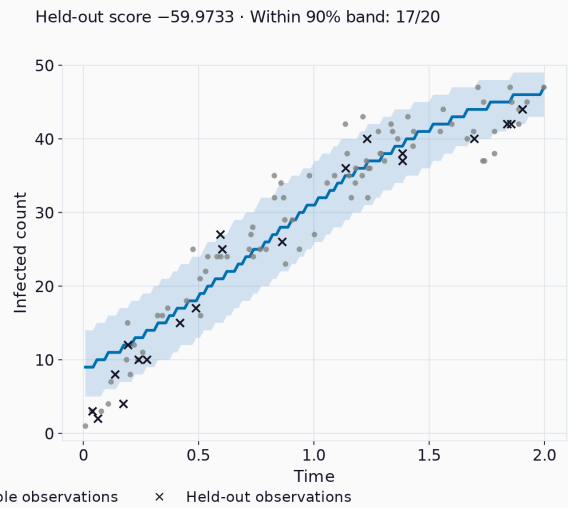
Disease spread: programs and their predictions

Qwen3.6-35B-A3B single attempt

A Reference



B Single attempt 1



A Reference program excerpt

Source lines 8-12

```
8 with pm.Model() as result:
9     t = pm.Data("t", data["Time"])
10    y = pm.Data("y", data["Infected_Count"])
11    theta = pm.TruncatedNormal("theta", mu=1, sigma=1, lower=0, upper=2)
12    pm.Binomial("Infected_Count", n=50, p=1-pm.math.exp(-theta*t), observed=y)
```

Exponential-approach probability; Binomial population 50.

Program file: 01-reference.py

B Single attempt 1 program excerpt

Source lines 25-26, 29-30, 33

```
25     alpha = pm.Normal('alpha', mu=0, sigma=10)
26     beta = pm.Normal('beta', mu=0, sigma=10)
: [lines 27-28 omitted]
29     logit_p = alpha + beta * times
30     p = pm.math.sigmoid(logit_p)
: [lines 31-32 omitted]
33     infected_observed = pm.Binomial('infected_observed', n=N, p=p, observed=infected_counts)
```

Input arrays and N = 50 are defined on source lines 14-16.

Program file: 03-single-attempt-1.py

Both programs use the same 80 visible simulated observations; lines show medians and shading shows 90% predictive intervals.

Among 20 held-out points, 17 fall within single attempt 1's band and 19 within the reference's; single attempt 1 also has a lower score.

The symbol : marks an omission.

Disease spread: the programs predict infected counts from time in a population of 50, and the plot shows the predictive medians and central 90% intervals for the reference model and single attempt 1. Single attempt 2 is shown in the listing below.

The programs behind the curves

Selected model statements.

Reference program

```
with pm.Model() as result:
    t = pm.Data("t", data["Time"])
    y = pm.Data("y", data["Infected_Count"])
    theta = pm.TruncatedNormal("theta", mu=1, sigma=1, lower=0, upper=2)
    pm.Binomial("Infected_Count", n=50, p=1-pm.math.exp(-theta*t), observed=y)
```

First single-attempt program

```
times = np.array(data['Time'])
infected_counts = np.array(data['Infected_Count'])
N = 50 # Total population size
with pm.Model() as pmodel:
    alpha = pm.Normal('alpha', mu=0, sigma=10)
    beta = pm.Normal('beta', mu=0, sigma=10)
    logit_p = alpha + beta * times
    p = pm.math.sigmoid(logit_p)
    infected_observed = pm.Binomial('infected_observed', n=N, p=p, observed=infected_counts)
```

Full reference program · Full generated program

Read the generated disease-spread program (single attempt 2)

```
import pymc as pm
import numpy as np

def model(data):
    """
    Defines a PyMC model for predicting infected counts in a population of 50.

    Args:
    data (dict): A dictionary containing 'Time' and 'Infected_Count' arrays.

    Returns:
    pm.Model: The defined PyMC model.
    """
    time = np.array(data['Time'])
    infected_counts = np.array(data['Infected_Count'])
    N = 50 # Total population size

    with pm.Model() as m:
        # Priors for the logistic regression coefficients
        # alpha: intercept, beta: slope
        alpha = pm.Normal('alpha', mu=0, sigma=5)
        beta = pm.Normal('beta', mu=0, sigma=5)

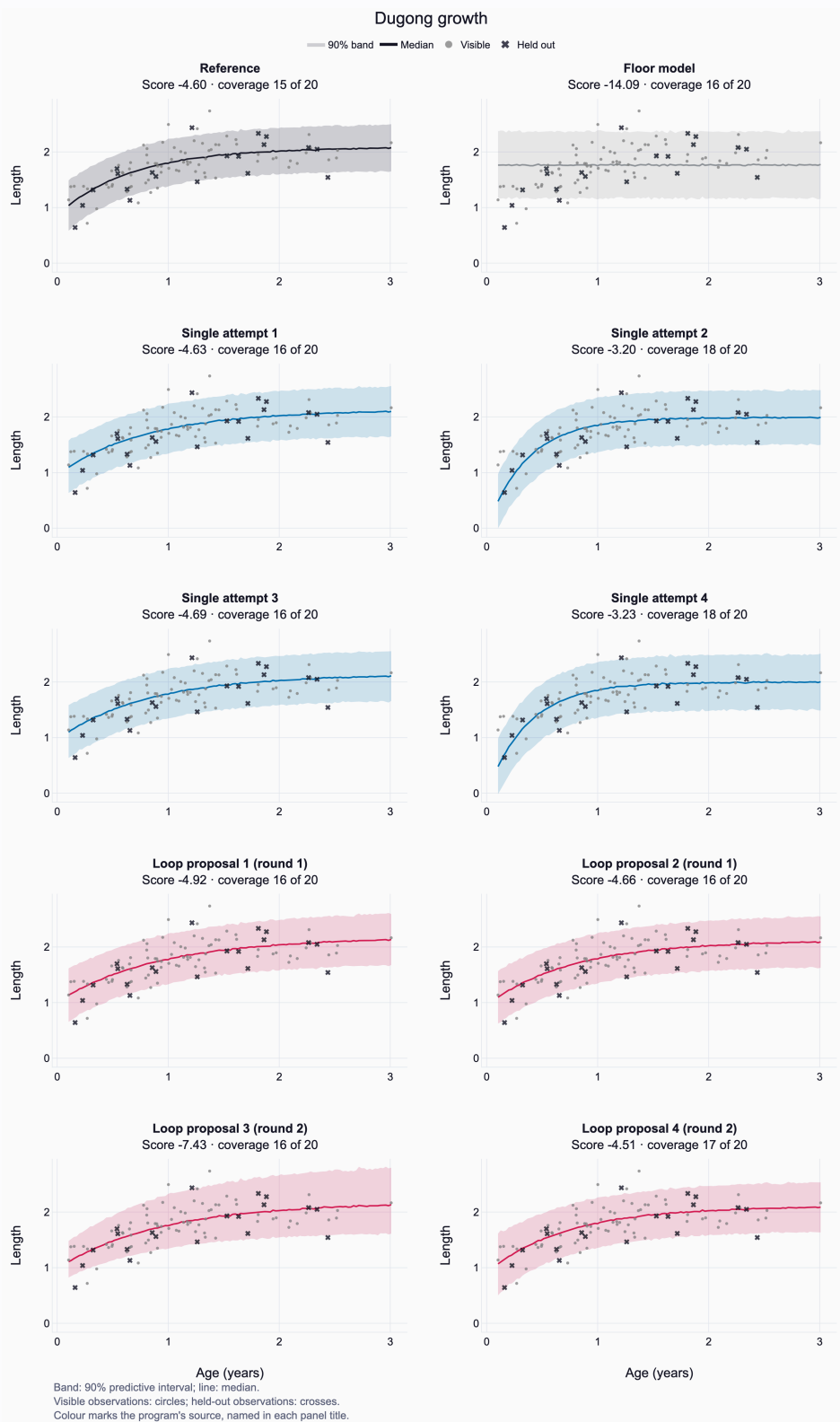
        # Compute the probability of infection at each time point
        # Using the logistic function:  $p = 1 / (1 + \exp(-(\alpha + \beta * t)))$ 
        logit_p = alpha + beta * time
        p = pm.math.invlogit(logit_p)

        # Observed likelihood: Binomial distribution
        # The number of infected individuals is binomially distributed with N trials and
        probability p
        infected_observed = pm.Binomial(
            'infected_observed',
            n=N,
            p=p,
            observed=infected_counts
        )

    return m
```

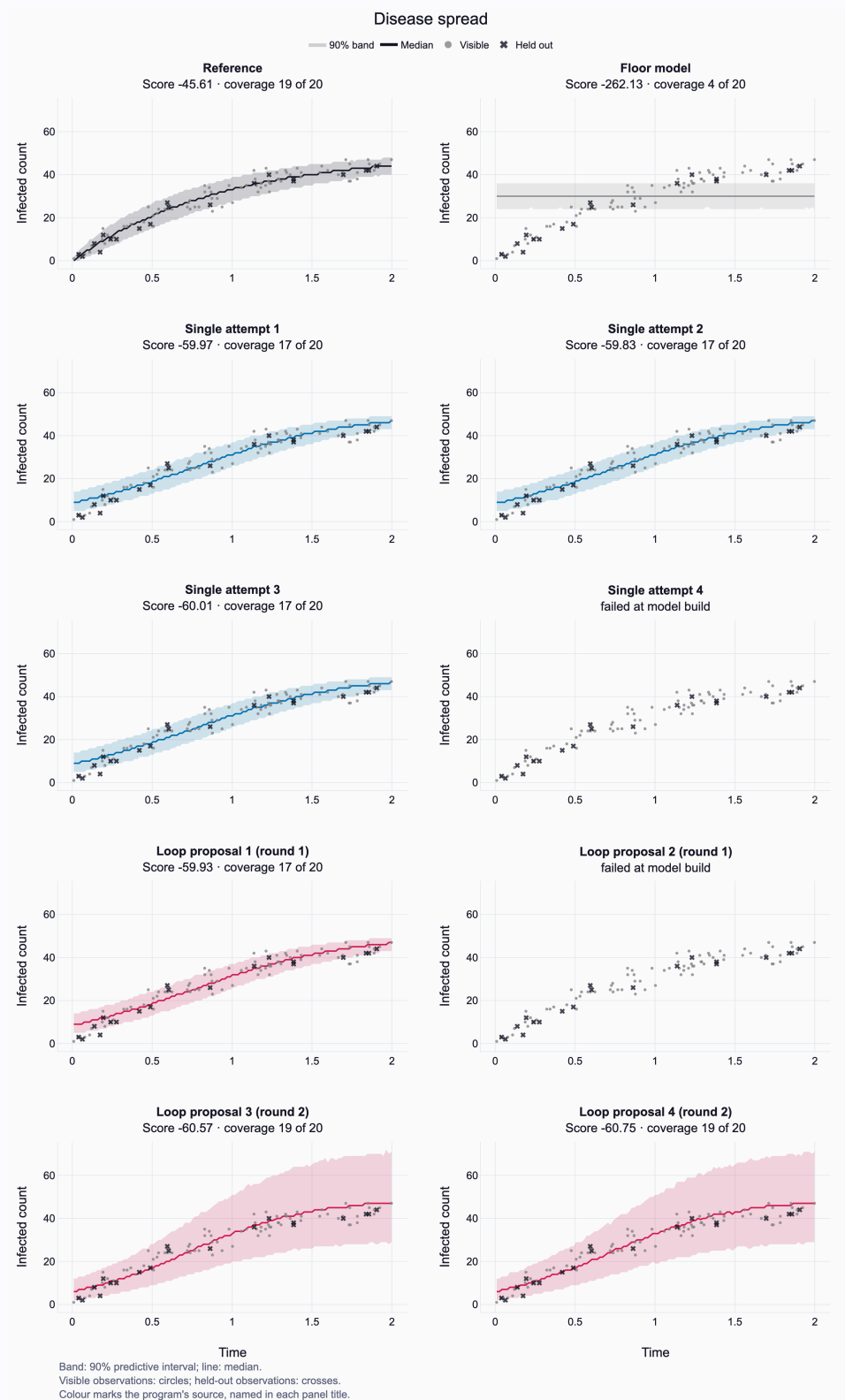
Inspect every program, score and cost, or follow one dugong prediction through its score.

All ten programs in each environment



Length against age: posterior predictions, one panel per program

Full-screen interactive predictions



Infected count against time: posterior predictions, one panel per program

Full-screen interactive predictions

Reward design

Asking the same question twice can produce different programs: the AI has a distribution over dugong programs. Each program, in turn, describes a distribution of lengths at a given age, possibly through unnormalised weights that need not sum to one. Our research goal is to learn a distribution over plausible programs, so a scientist can compare different modelling assumptions about the same observations.

For a dugong program, we measure predictive fit on the responses marked by crosses in the figure, which were withheld from the AI. We also give credit for passing the modelling checks, and call a program that passes them *admitted*. A scorable admitted program earns admission credit plus a predictive-fit term, q , which compares its held-out expected log predictive density (ELPD) with the task's baseline ELPD:

Quantity	Definition
Utility of a scorable admitted program	$1.5 + q$
Predictive-fit term	$q = \tanh[(\text{ELPD} - \text{baseline ELPD}) / n]$
baseline ELPD	The task's reference program refitted on its full visible rows with its ordinary predictors replaced by constants computed from those rows alone (a predictor's mean, or its most frequent value if it is categorical) and its structural inputs, such as group membership, offsets and censoring indicators, left at their real values, then scored on its withheld rows with the same constants
n	Number of held-out observations
Range of q	$-1 < q < 1$
Episode with a program rejected by the checks, or with no submitted program	0

Admission rules

Stage	Requirement
Before fitting	The program passes the syntax and safety checks of the verifier (the software that inspects submitted programs)
After PyMC builds the model, under the strict PyMC profile	No custom densities or arbitrary additions to the log density, no priors or branches that depend on PyMC data containers
After fitting	The held-out ELPD is finite

Matched training comparisons

GLM-5.3-Flash is the larger model in these comparisons, a tool-using agent trained by reinforcement learning through Tinker, a hosted training service. An attempt, which we call an *episode*, can include data requests and a program submission.

Setting	Value
Training	220 training packages, four episodes on each, over 14 updates
Held-out tasks	The same 103 task packages for trained and untrained policies, each containing a prompt, data and reference scores
Mean utility	Episode scores averaged within each package, with equal weight on each package mean
Training and evaluation split	Related tasks kept together; 39 scenario groups in the held-out set
95% interval for the paired change	10,000 resamples of whole scenario groups
Scorer	The same scorer for both arms within each comparison
Separate experiment, third row: training	14 updates; a language-model judge's score of the submitted program at weight 0.25, alongside fit at weight 1.0
Separate experiment, third row: evaluation	Fit alone, against the experiment's own matched untrained arm, using its own scorer

GLM-5.3-Flash training	Scorer and untrained comparison	Episodes per package and arm
Fit only, 3 updates	Fit-only scorer; matched untrained arm	2
Fit only, 14 updates	Fit-only scorer; matched untrained arm	8
Fit weight 1.0 + judge weight 0.25, 14 updates	Fit-only scorer; this experiment's matched untrained arm	8

The fit-only and judge-assisted comparisons sampled their untrained episodes separately and used different verifier versions.

For any one program, admission credit is larger than any gain or loss from q.

Training methods for the main comparisons

GLM-5.3-Flash training	Setting
Recipe	Reinforcement learning from the cookbook for Tinker, a hosted training service
Tools in the 14-update evaluations	GLM tool calls to list candidates, query observations, inspect data, run programs and submit a program
Training update	The Tinker cookbook baseline: each episode's reward is compared with the rewards of the other episodes on the same task
Penalty for moving away from the untrained model	None
Weights trained	Low-rank adapters (LoRA) of rank 32
Learning rate	1×10^{-5}
Episodes per training task	4
Training runs per comparison	One
Judge (judge-assisted training only)	DeepSeek V4.1 Flash, fixed binary rubric, weight 0.25

Budgeted observation acquisition and three arms of eight updates

The small agent is Qwen3.5-0.8B with tools. We first trained it to imitate positively rewarded episodes from other language models, which we call teachers. After this supervised warm start, we trained it with Subtrajectory Balance (SubTB) in two arms, at learning rates of 1×10^{-5} and 1×10^{-6} , and with group relative policy optimization (GRPO) in a third, each over eight updates on 220 training packages in 81 scenario groups.

In each episode, we gave the agent a query budget to spend on selecting observations (in the common buzzard task quoted below, survey routes) before writing a program. During training, an episode's *reward* adds an admission component, for passing the program checks, to a fit term that measures how well the program predicts relative to the task's baseline. All three arms trained on the agent's sequences of tool actions and their rewards.

Small-agent training, Qwen3.5-0.8B with tools	Setting
Training stack	An open stack in place of Tinker: Harbor runs task episodes, mini-SWE-agent runs the tool-using agent and SkyRL trains the policy
Model identifier	Qwen/Qwen3.5-0.8B
Warm start	LoRA rank 32; learning rate 2×10^{-5} ; batch size 4; two epochs; 98 updates
Training packages	220, in 81 scenario groups
Episodes per update	32 prompts $\times$ 4 episodes = 128
Updates per arm	8
Low-rank adapters	LoRA rank 32
Learner temperature	0.7
Reward temperature	0.25

Small-agent arm	Objective	Learning rate
SubTB	Subtrajectory Balance on segments of the agent's sequence of actions and its rewards	1×10^{-5}
GRPO	Group relative policy optimization: each episode's reward against those of the other episodes on the same task	1×10^{-5}
SubTB, reduced learning rate	The same SubTB objective	1×10^{-6}

The Swiss breeding-bird survey records whether Common Buzzard was detected on the first visit of each route. Site covariates describe elevation in metres, forest cover in percent, and route length in kilometres. The analyst wants to learn how these covariates relate to first-visit detection and predict whether the species is detected on other routes. Replicate structure and imperfect detection are not modelled.

Held-out comparison with the warm start

Small-agent comparison with the warm start	Design
Held-out evaluation packages	103, in 39 scenario groups
Arm	The policy after eight updates, compared on each package with the warm start it began from
Paired change	Mean utility after eight updates minus the warm start's mean utility
Interval	95%, from 10,000 resamples of whole scenario groups
Three comparisons	Bonferroni interval at 98.333%, for three comparisons sharing one warm start

Predicting coral size

In one GLM-5.3-Flash training task, we wanted to predict the maximum diameter of brain coral at simulated reef sites from environmental observations, in a setting where the AI could request measurements and run code before submitting a statistical program to predict other sites. The coral prompt also contained outcome summaries computed from all 48 candidate sites.

The submitted coral program specifies its regression and its likelihood in ordinary PyMC.

Coral episode	Measurement
Policy and population	GLM-5.3-Flash; one training package in the 220-package run
Available candidate sites / requested observations	48 / 8
Regression predictors	Water depth; sedimentation rate; years since bleaching
Utility / admission / predictive fit	2.2276 / 1.5 / 0.7276
Effect of rescaling the weights	Multiplying every unnormalised weight by the same positive constant changes neither the normalised probabilities nor the score on the withheld rows, while a reward computed from the raw weights could rise

Scale of the task collection and the training workload

The coral task and the common buzzard survey belong to a collection of 323 PyMC task packages that covers both simulated processes and species surveys, and when we split the collection, we kept related tasks together in scenario groups: 220 packages in 81 groups trained the models, and the other 103 packages in 39 groups were held out for evaluation. Each task comes as a package containing the prompt and data, rows withheld for scoring, and the scores of a reference program:

PyMC task packages	Count
Forward-modelling: outcomes from a generating process that the prompt describes	60
Survey-count: simulated counts, calibrated on a real species survey	134
Survey-detection: simulated binary detections, calibrated on a real species survey	129
All packages	323

On the held-out tasks, we compared the larger comparator, GLM-5.3-Flash, and the small agent, each against the policy it started from.

The larger comparator trained on its own attempts, such as the coral episode. For the small agent's warm start, we used positively rewarded teacher episodes, including the tool actions that led to a submitted program:

Training episodes and warm start	Count
Training episodes of the larger comparator, four on each training package	880
Audited teacher episodes with a positive reward, imitated in the small agent's supervised warm start	193
Training scenario groups covered by those teacher episodes	63 of 81
Tool actions in those teacher episodes	1,930
Warm-start updates, over two epochs	98

We also collected programs written by teacher language models, in two separate collections:

Teacher program collections	Count
Teacher programs collected from GPT-6 Astra and GPT-6.1 Sol	8,468 (7,280 and 1,188)
Programs from a separate ensemble of five teachers, on 135 training packages	1,346 distinct

Supporting measurements

Datasets and training populations

The four-task pilot of 20 October trained on four packaged tasks from four scenario groups, outside the 323 packages of the task collection above. See also the [supporting results page](#).

Collection	Size and population	Composition
PyMC reinforcement-learning environments, 30 September to 1 October	375 tasks: 358 new and 17 earlier (8 for training, 9 for evaluation)	New tasks in four families
The same 375 tasks: LazyPPL versions	342 of the 358 new tasks also have a version in LazyPPL, a probabilistic programming library for Haskell	Counts and detections calibrated on a real species survey
Packaged tasks, 20 October	4 tasks, one from each of 4 scenario groups	Fixed reference fits; outside the 323-package collection
LazyPPL (Haskell) prompt-program pairs	3,671 pairs on 155 eligible tasks	Admitted pairs
Agent trajectories stored in ATIF (an interchange format), March to August	189 trajectories	155 PyMC and 34 LazyPPL

Tasks, packages and episodes

Term	Meaning
Task	One modelling problem
Package	A task's prompt and data, together with its withheld rows and reference scores
Episode	One attempt at a task, including the agent's data requests and its submitted program
In-sample	Measured on the tasks that the policy trained on
Development / reserved evaluation	Separate populations: packages used for intermediate measurements / the 103 held-out packages used for the main trained-versus-untrained comparison

What the reference tasks assume

For each forward task in this comparison, we fitted its PyMC and LazyPPL (Haskell) versions to the same visible rows. For each parameter compared, we measured the gap between its posterior means in the two versions in units of its PyMC posterior standard deviation (SD). Every measured gap was smaller than one SD.

Forward tasks compared	Parameters	Median gap between the two means (SD)	Largest gap (SD)
69 task specifications	371	0.056 (95th percentile 0.24)	0.56
Second forward cohort, 46 specifications	261	0.074	0.70

We examined the reference programs of the 220 training packages for assumptions absent from their prompts. Almost all confirmed unstated assumptions are generic modelling choices; only three are domain facts.

Assumption not stated in the prompt	Count
Generic modelling choice	1,196
Domain fact that the prompt does not supply	3
Other assumption	44
All confirmed assumptions	1,243

Training data and exposure

Run or restriction	Data used or held out
Larger-model run, GLM-5.3-Flash, 21 October	220 training packages in package order, in updates of up to 16 packages × 4 episodes; 14 updates, 880 episodes; first 3 updates: 48 packages, 192 episodes
Judge-assisted larger-model run, GLM-5.3-Flash, 21 October	220 training packages × 4 episodes; 14 updates
Small-agent warm start, Qwen3.5-0.8B, 21 October	193 verified teacher episodes with 1,930 tool actions, from 63 of the 81 training groups; 2 epochs, 98 updates
Small-agent reward training, Qwen3.5-0.8B, 26 October	220 training packages in 81 groups; 32 prompts × 4 episodes per update; 8 updates in each of three arms
GLM-5.3-Flash, 20 October	4 packaged tasks; 32 episodes over 2 optimizer steps
GLM-5.3-Flash, GRPO, 19 October	One forward training package; 3 updates of 8 episodes
Qwen3.5-4B, TB control, 18 October	One update from a schedule of 24 prompt rows on 23 training packages
GLM-5.3-Flash, 16 October	3 updates of 4 prompts × 8 answers from training-side packages
Qwen3.5-0.8B, supervised pilot, 5 October	92 teacher programs on 32 LazyPPL tasks; one program held out for validation
Qwen models, May	1 training example, 1 update
Teacher programs excluded from training and tuning	879 written for evaluation packages; 393 for packages held back from the split
Evaluation populations	103 reserved packages; 9 earlier held-out tasks; other measurements on the 323-package collection use its training side

Training pilots and objective tests

Pilot	Training and measurement
Reference-weighted trajectory balance (TB), GLM-5.3-Flash, 16 October	3 updates of 4 prompts × 8 answers from training-side packages; measured on 10 fixed development packages, one of them from a training group, with 1 answer per package and 1 seed
Eight answers per development problem, 18 October	40 development packages: the 10 fixed ones and 30 from groups unused in training
Tool-using group relative policy optimization (GRPO), GLM-5.3-Flash, 19 October	3 updates of 8 episodes on one forward training package; scorer fitted each program on all 64 rows
Four tool-using tasks, GLM-5.3-Flash, 20 October	4 packaged tasks; 32 episodes over 2 optimizer steps; query budget 8; measured in-sample on 16 episodes per policy, the initial policy's being the training episodes collected at the initial weights
Cached-program objectives, Qwen3.5-4B, 30 September	8 updates per objective on 128 cached programs; 1 seed; scored rates estimated by reweighting the untrained model's samples

Evaluations and baselines

LazyPPL programs

In LazyPPL, a probabilistic programming library for Haskell, the untrained Qwen3.5-4B wrote four programs for each of 155 tasks from the training split.

Model	Tasks and programs, training split	Result
Untrained Qwen3.5-4B, separate draw with answers of up to 8,192 tokens	Same 155 tasks	2 of 620 through the type checker
GPT-6 Astra	155 tasks; 620 programs	604 accepted
GPT-6.1 Sol	155 tasks; 620 programs	605 accepted
Kimi K3	155 tasks; 620 programs	510 accepted
Claude Sonnet 4.6	310 programs	249 accepted
Haiku 4.5	310 programs	163 accepted
Accepted program	LazyPPL scoring	Accepted by the scorer at run time and passed its numerical check that the probabilities add up to one

Experimental design and discovery in BoxingGym

We took a snapshot of BoxingGym runs on 13 February 2026: seven untrained API models in ten environments of the benchmark behind both demonstrations. Every one of them predicted worse than BoxingGym's own baseline on average:

Snapshot item	Description
Kept	1,946 of 5,710 rows
Selection	Query budget of at least 10; mean z-scores between -100 and 100, inclusive
Baseline	For dugongs, the baseline draws 10,000 simulated measurements, averages them, and predicts that average length at every age; each environment defines one, which BoxingGym's code calls the naive predictor, and records the mean and standard deviation of its errors
Score	Z-score: a model's mean error minus the BoxingGym baseline's mean error, divided by the baseline's standard deviation; 0 matches the baseline, +1 is a mean error one baseline standard deviation larger, -1 is one baseline standard deviation smaller; lower is better
Matched cell	A group of runs with matching task and replication settings; 20 such cells contain runs from all seven models at a query budget of 10
Matched comparison	Uses only those 20 cells; each of the ten environments has equal weight, and the 95% intervals come from resampling whole environments

At a query budget of 10, all seven models ran 20 cells in common.

Model	Pooled mean z-score (95% bootstrap interval)	Matched mean z-score (95% interval)	Matched rows
DeepSeek-V3.2	+0.296 [+0.122, +0.532]	+0.245 [-0.194, +0.760]	295
GPT-4o	+0.255 [+0.136, +0.387]	+0.258 [-0.126, +0.648]	157
MiniMax-M2.1	+0.185 [+0.071, +0.315]	+0.262 [-0.128, +0.694]	273
GPT-5.1-Codex-Mini	+0.586 [+0.310, +0.966]	+0.279 [-0.158, +0.771]	174
Kimi-K2	+0.262 [+0.130, +0.408]	+0.352 [-0.105, +0.863]	271
GLM-4.7	+0.546 [+0.257, +0.904]	+0.405 [-0.127, +1.061]	260
Qwen3-32B	+0.275 [+0.115, +0.464]	+0.506 [+0.040, +0.993]	165

The kept runs had a higher mean z-score in discovery mode than in optimal experimental design:

Runs compared	Runs	Mean z-score
Optimal experimental design / discovery	700 / 1,246	+0.122 / +0.466
With / without probabilistic-program assistance	115 / 1,831	+0.538 / +0.330; two-sided Welch t test, p = 0.134

We also ran the untrained Qwen3-32B on five seeds, asking it to predict dugong lengths directly, with no program: each seed ended its ten experiments with a larger error than before the first. In a separate study, five API models wrote a PyMC and a LazyPPL (Haskell) program at each budget, and far more of the PyMC programs could be scored on the withheld points:

Run	Population and design	Result
Dugong predictions without a program	Untrained Qwen3-32B on Bedrock; seeds 1 to 5; 10 evaluation questions at each budget, scored by mean squared error (MSE, lower is better); information gain and program assistance off	Mean MSE 0.149 before any experiment and 0.640 after 10 (0.631, 0.650, 0.537 and 0.552 after 1, 3, 5 and 7); every seed ends above its starting error
Programs after successive queries	Five API models; a PyMC and a LazyPPL program after 0, 1, 3, 5, 7 and 10 successful queries; 20 withheld points per episode	225 of 420 planned episodes recorded, 191 with both programs at every query count; finite held-out scores: PyMC 761 of 1,165, LazyPPL 235 of 1,167

Further evaluations

The held-out populations of the 3-update and 14-update GLM-5.3-Flash comparisons:

GLM-5.3-Flash against its untrained arm	Held-out population
Fit only, 3 updates	103 held-out packages in 39 groups; 206 episodes per arm
Fit only, 14 updates	103 held-out packages in 39 groups; 824 episodes per arm

Reward measurements

Of 44 reference models, 25 scored below the baseline (the same program refitted with its predictors held at constants) on held-out ELPD, the sum of log predictive densities over a task's withheld rows, from which the predictive-fit term q is computed. By kind of split, the reference model scored below the baseline in 4 of the 18 random-row evaluations and in 21 of the 26 evaluations whose withheld rows were ordered tails (the final rows after sorting by time-related columns), whole groups or spatial regions:

How the withheld rows were chosen	Reference model below the baseline	Reference model not below it
Ordered tails: final rows in that order	14	3
Group hold-out: whole groups	6	2
Spatial regions	1	0
Random rows	4	14

When we rescored the same programs under four scoring seeds, about a quarter of program pairs swapped order, but few of the pairs that were more than one standard error apart did:

Measurement	Population and comparison	Result
Program order across scoring seeds	17 tasks; 4 seeds; 896 jobs, 802 scored and 94 rejected by the program checks	201 of 818 program pairs (24.6%) change order; 52 of the 669 pairs more than 1 standard error apart (7.8%)
Reference ELPD protocol against long runs of PyMC's No-U-Turn sampler (NUTS)	5 reference programs; difference as a share of the reference-baseline gap	Equal-weight mean -0.5%; nominal 95% Student t interval across the five program percentages (4 degrees of freedom) [-1.3%, +0.3%]
LazyPPL posteriors against PyMC NUTS	18 posteriors from posteriordb, a public reference collection; translated PyMC and LazyPPL models; LazyPPL fixed-mixture Metropolis-Hastings; visible rows and full rows fitted separately	After convergence and reference checks pass, agreement means every Monte Carlo interval (error multiplier 4) lies within 0.2 PyMC SD for mean gaps and 0.8 to 1.25 for SD ratios; any interval wholly outside means disagreement. Other cases are inconclusive. 7 agree in both scopes, 11 inconclusive, 0 disagree; across the two scopes: 14 of 36 agree, 22 inconclusive, 0 disagree
The same saved answers on two machines	4 LazyPPL answers; 4 PyMC answers	LazyPPL: all 4 identical; PyMC: all 4 within 7.99 nats (1 identical)

We tested four constructed forgeries and one honest control against one verifier version: all four forgeries were refused at model build and the control scored. In a separate LazyPPL scoring test, a program returning an arbitrary finite log-likelihood outscored an honest one by a wide margin, an instance of the inflation of likelihood rewards studied in the likelihood-hacking paper below:

Constructed programs	Population	Result
Constructed forgeries	4 forgeries and 1 honest control, run against one verifier version on 20 Oct	All 4 forgeries refused at model build; the control scored
Arbitrary finite log-likelihood output in LazyPPL scoring	Constructed program against an honest program	Training log reward 233,827.998 against 0.912
Language-model judge contrast set	One judge model; good and flawed constructed programs	Good program: judge score 1.0 (3 of 3); flawed programs: 0.5 to 0.625

Public alpha release

The release provides executable modelling examples with their observations and predictions, evaluation tools, and shared task, trace and reward schemas. The dugong program listed above as single attempt 2 is one of its Python files: the four commands below download it with the observations, fit it to the visible rows and compute its held-out score.

Part of the release	Contents and links
Code repositories	BoxingGym evaluation (the benchmark, with tools that evaluate several models on it) and the likelihood-hacking repository (the scoring and model checks of our UAI 2026 paper on likelihood hacking)
Task, trace and reward schemas	SynthStats/synthstats-schemas
PyMC and LazyPPL program datasets and package export	SynthStats/ppl-synth-data
Interactive pages and an HTML report	Prediction and score figures for dugong growth and disease spread , running in the browser without a model account, and a self-contained HTML report of both demonstrations
Programs (20 Python files, 10 per environment)	The reference program, the floor model, 4 single attempts and 4 loop proposals for each environment; the dugong results and disease-spread results show which programs scored, and the dugong and disease-spread collections hold every program's source
Data (10 CSV files, 5 per environment)	Observations, program summaries, held-out predictions, predictive curves and four-chain fit data, on the data page

Fit a generated program

The fitting script selects the visible rows of the dugong observations (those supplied to the model) and fits the displayed dugong program to them, then prints the first posterior draw, the mean of its posterior parameters and its held-out score. The demonstration was scored under `PyMC 5.21.1` and `ArviZ 0.23.4`. In an empty directory, the commands below use `curl` and `uv` to download the program and the observations along with the script, and then run the fit.

```
curl -fsS https://synthstats.org/programs/dugongs/04-single-attempt-2.py -o dugong_model.py
curl -fsS https://synthstats.org/data/dugongs/dugongs-observations.csv -o observations.csv
curl -fsS https://synthstats.org/examples/fit-dugong.py -o fit-dugong.py
uv run fit-dugong.py
```

UAI 2026 paper on likelihood hacking

In Likelihood hacking in probabilistic program synthesis, published in the UAI 2026 proceedings, we trained a language model to write PyMC programs with the log marginal likelihood as its reward. In SynthStats, the strict PyMC checks before fitting refuse `pm.Potential`, with which a program can add an arbitrary term to its log density and so inflate a likelihood reward.

The paper's setting, separate from the training runs above	Value
Model trained	Qwen3-4B-Instruct-2507
Training	Group relative policy optimization (GRPO) with low-rank adapters (LoRA) of rank 32
Data	Three-dimensional Bernoulli (yes-or-no) data, $d = 3$
Reward	Log marginal likelihood of the training split

Training run	Model outputs	Valid	Using <code>pm.Potential</code>
29 training steps	23,053	17,522	804
28 training steps	22,257	17,042	606

Check	Programs checked	Result
SafePyMC, which checks the model a PyMC program builds	37,339 valid generated records from four runs (two large and two short)	Rejected all 116 programs with positive reward; in a separate curated test, rejected all 20 discovered high-reward programs and accepted the honest baseline
Numerical audit of whether the probabilities add up to one	108 programs from the two runs above, 2 of them from a final step outside the run totals	86 violated normalization; 22 unresolved
SafeStan, a static checker for the Stan probabilistic programming language	1,163 public Stan models	A language model, asked to translate each model for SafeStan, labelled 721 (62%) of the translations 'as-is' (no change) or 'minor' (a mechanical change)
Heuristic classifier for likelihood hacking	Pooled outputs of 7 untrained language models	Fewer than 1% flagged

Team

SynthStats is the work of Jacek Karwowski, Younesse Kaddar, Daniella Ye, Esmeralda S. Whitammer and Sam Staton.