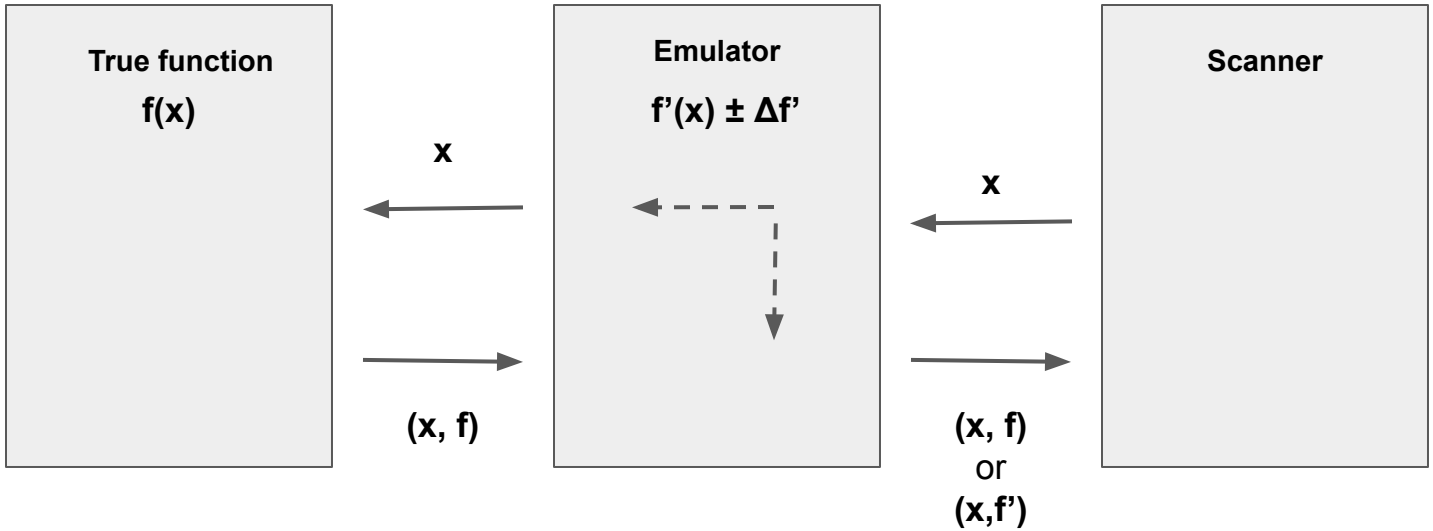
Continual learning + GAMBIT?

- Continual learning
- Online learning
- Incremental learning
- On-the-fly regression
- Online emulation
- Online surrogate models
- …

- := Learn an approximator/emulator for a function by training incrementally on a continual stream data points

- Much explored in robotics: data stream is sensor data
- Emulator has no say in data sampling – passively receives a data stream
- How to best make use of this data stream, given an emulator with finite computational resources?
- Translated to GAMBIT case:
    - stream of data: scan points (x) received from scanner
    - emulator learns some function f(x) on-the-fly during the scan
    - f(x) : observable, likelihood component, total scan likelihood, …
- Typically larger timescale per point than in robotics – can afford longer per-point training

- “Emulator-in-the-middle”
- A layer between the scanner and the likelihood/observable computation



- If **Δf’** is large, run full, expensive **f(x)** computation and use point **(x,f)** to update emulator
- If **Δf’** is small, short-circuit computation by returning **(x,f’)** to the scanner

- Emulator requirements:
  - Can be trained sequentially
  - Computationally cheap in both training and evaluation
  - Independent of parameter sampling algorithm
  - Reliable per-point uncertainty estimates

- Why not use a regular pre-trained emulator?
  - Usefulness of pre-trained emulator depends on *reusability*
  - Some target functions change for every new GAMBIT study (e.g. likelihoods change with new data)
  - Sometimes training a global emulator is as expensive as the GAMBIT scan itself → on-the-fly training focuses on the parameter subspace relevant for the given scan

- Examples in the literature:
  - BAMBI: MultiNest + NN emulator for the likelihood [arxiv:1110.2997]
    - NN trained in batches, not single-point updating
    - No per-point uncertainty estimate
  - Cosmo++: Local, polynomial fits to training points [arxiv:1506.01079]
    - Learns error model to give per-point error estimates
    - Number of polynomial coefficients increases rapidly with the dimensionality of the input space and the degree of the polynomial → in practice restricted to linear or quadratic polynomials
- For a realistic use-case, Cosmo++ demonstrates speedup of 6.5 in a MCMC fit, and 3.5 in a nested sampling fit.

- Prior work on this in GAMBIT context:
  - 2019: Ben, Pat and I supervised two MSci students (Iza Veliscek, Nicholas Reed)
  - David involved as examiner
  - Project explored regression with *local* Gaussian processes

- Current work:
  - Collaboration with Riccardo De Bin and Timo Lohrmann in Oslo
  - Extending a method known as *Dividing Local Gaussian Processes* (DLGP) [arxiv:2006.09446]
  - Writing an R package, will use old GAMBIT scan data for motivation and testing
  - I will turn it into a C++/Python package later

- Training GPs scale as n^3 (n = number of training points)
- A single, growing GP is unusable for on-the-fly emulation in a GAMBIT setting (millions of points in a scan)
- DLGP:
  - Use data stream to dynamically grow a probabilistic binary tree
  - Each node is a *small* GP (e.g. size cap of training 100 points)
  - New training point: the tree assigns it to one GP
    - Fast updating since the GP is small
  - If a GP reaches it size cap, it splits into two GPs (tree grows)
  - New point for prediction: tree determines probability for point to "belong to" different GPs
    - Fast prediction: combined prediction from small number of small GPs

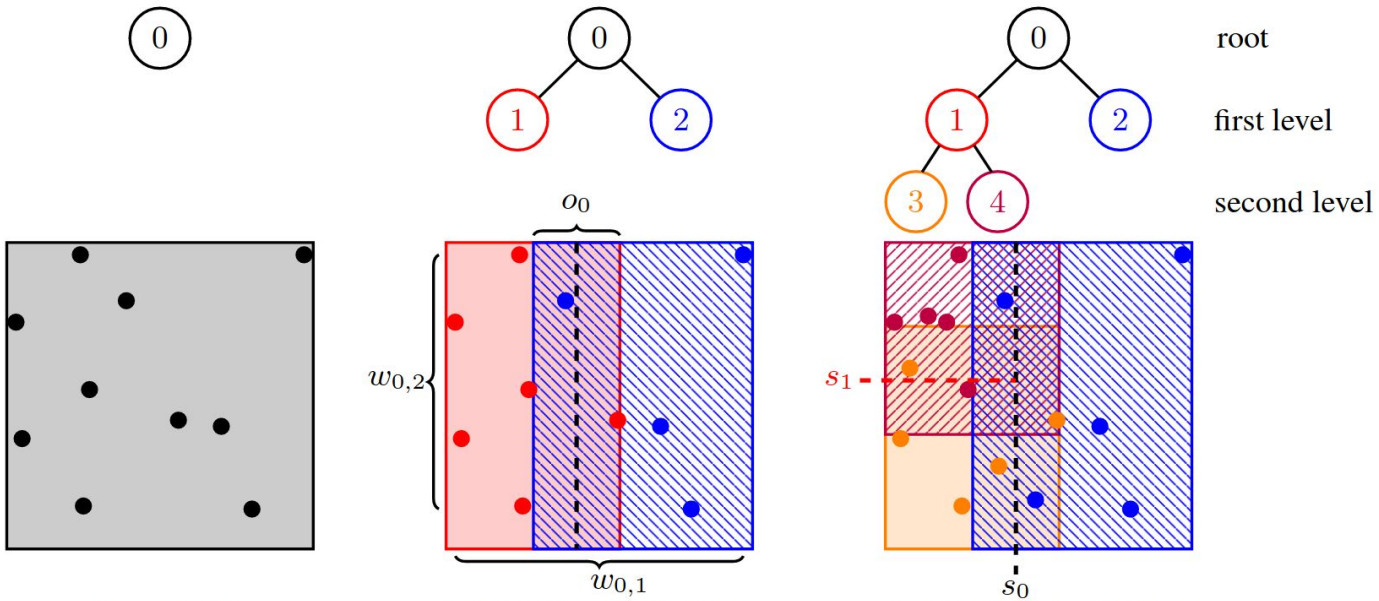


Figure 1: Iterative model tree construction and corresponding layout of the input space: active regions and training samples belonging to the same node are depicted in the same color.

[arxiv:2006.09446]

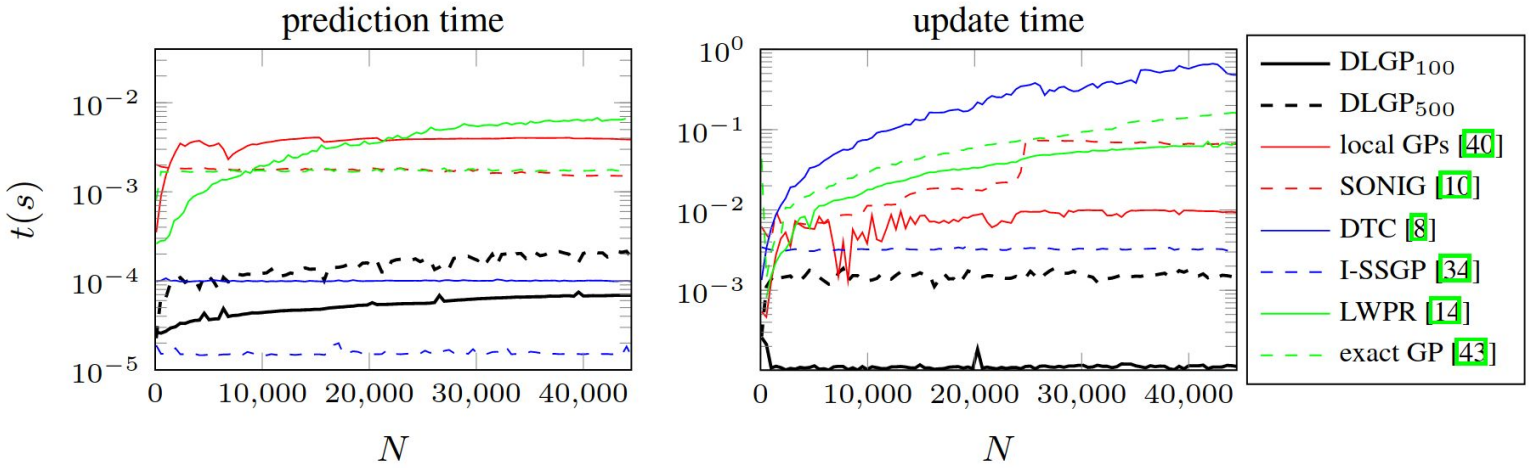
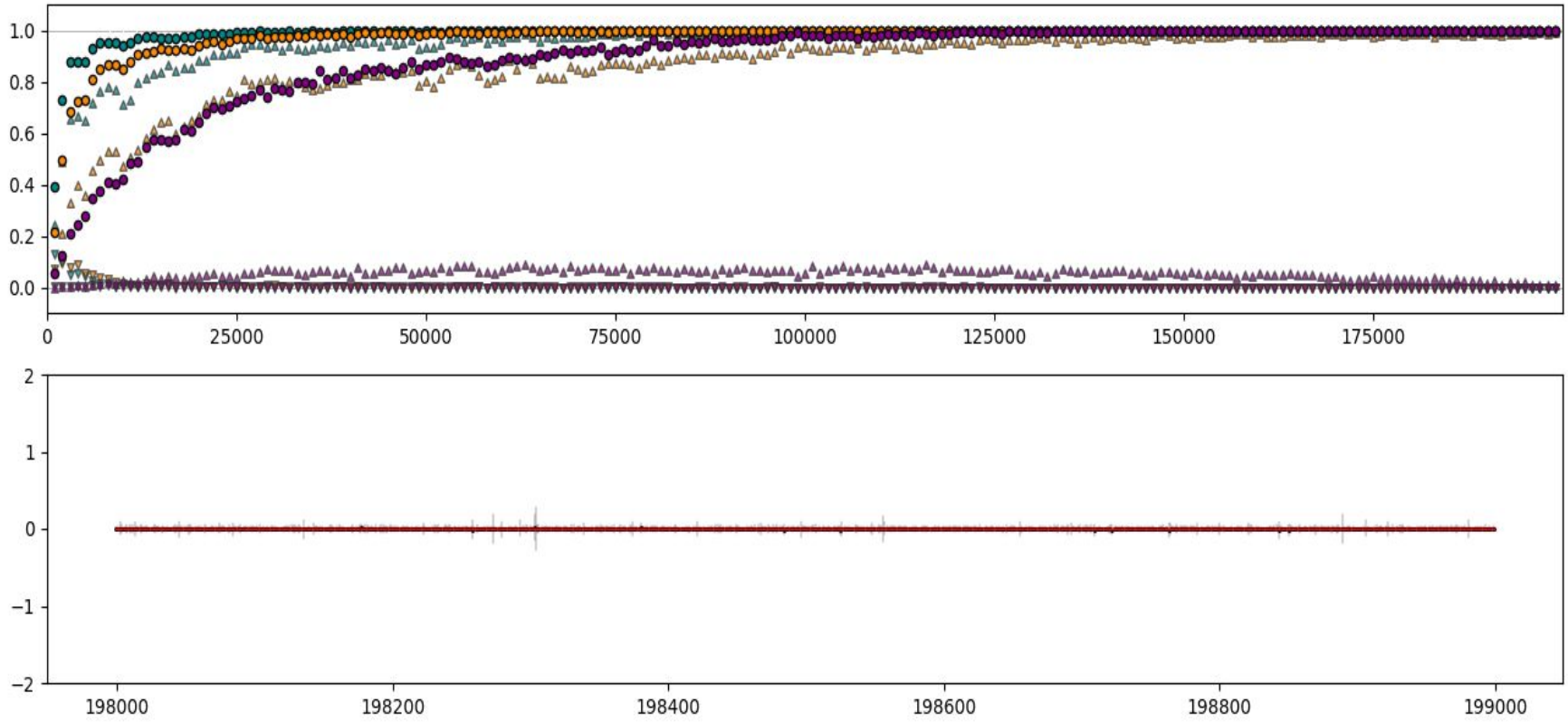


Figure 3: Left: The prediction time of DLGPs increases logarithmitcally on the SARCOS data set (joint 1), and remains smaller than for many existing methods. Right: The update time of DLGPs is almost constant due to a slow logarithmic growth, such that the DLGP method is significantly faster than all state-of-the-art methods.

[arxiv:2006.09446]


1.0
0.8
0.6
0.4
0.2
0.0
0
25000
50000
75000
100000
125000
150000
175000
2
1
0
−1
−2
198000
198200
198400
198600
198800
199000

- Proposed goal: **have a general emulator system in GAMBIT**
- Not directly linked to DLGP – if possible it should be agnostic to both scanner and emulator algorithm
- Should take advantage of the fact that we run GAMBIT with ~10k MPI processes
    - Individual emulators learn individual target functions?
    - Multiple emulators for one target function? (Boosting?)
    - A combination of the two?
- How we do the MPI communication will be important…

```yaml
ObsLikes:

  - capability: some_slow_computation
    purpose:    LogLike
    - options:
        train_emulator: true
        apply_emulator: true
        emulator_threshold: 0.01

  - capability: another_slow_computation
    purpose:    Observable
    - options:
        train_emulator: true
        apply_emulator: false

  - capability: yet_another_slow_computation
    purpose:    Observable
    - options:
        apply_emulator: true
        emulator_threshold: 0.05
```

- Possible benefits:
  - Speed up a single GAMBIT scan (factor 2? 5? 10?)
  - Emulators trained in one scan can be applied and further refined in the next scan, and in the next scan, …
    - Subsequent scans can spend more of their CPU resources on exploring the so-far undersampled parameter regions
  - *If emulating the total scan likelihood:* by choosing acceptance threshold relative to the current best-fit point, can accept fast likelihood estimate for points that will be well outside n-sigma region
  - After a GAMBIT scan: release trained emulators along with scan data on Zenodo
  - Can be used with new scanners to balance exploration against optimisation
  - Use GAMBIT to train “global” emulators that we release as standalone tools
  - More?

- A possible first step to estimate potential speed gains:
  - Pick a GAMBIT module function with an expensive likelihood/observable computation
  - Connect an on-the-fly emulator directly as a backend
  - Modify module function: start by evaluating emulator, if acceptable don't run the rest of the function
  - Note: in this case each MPI process would be an independent emulator

- But a proper emulator system in GAMBIT with clever MPI communication will require Core + ScannerBit modifications
- Lots of potentially interesting technical work – if people are interested!