Multi-Resolution Universal Horizon Address (UHA) Tensor Calibration API

Supplementary Technical Documentation

Eric D. Martin
All Your Baseline LLC
look@allyourbaseline.com

October 30, 2025

Abstract

This document provides comprehensive technical documentation for the Multi-Resolution Universal Horizon Address (UHA) Tensor Calibration API. This proprietary implementation enables systematic bias correction across multiple spatial scales, achieving superior concordance in cosmological parameter estimation. The method is protected under US Patent Application 63/902,536.

Contents

1 Introduction

The Multi-Resolution UHA Tensor Calibration system addresses a fundamental limitation in cosmological systematic bias correction: single-resolution spatial encoding cannot capture multi-scale systematic effects. Our iterative tensor refinement was previously stuck at $\Delta_T = 0.6255$ due to fixed spatial resolution (10 Mpc cells).

1.1 Key Innovation

Progressive refinement through a resolution hierarchy ($8 \rightarrow 12 \rightarrow 16 \rightarrow 20 \rightarrow 24 \rightarrow 28 \rightarrow 32$ bits per dimension) captures systematic biases at all spatial scales simultaneously, from local structures (< 1 Mpc) to global voids and superclusters (> 100 Mpc).

1.2 Expected Performance

Typical convergence progression:

- 8-bit resolution: $\Delta_T \approx 1.20$, Gap ≈ 5.42 km/s/Mpc
- 16-bit resolution: $\Delta_T \approx 0.45$, Gap ≈ 1.85 km/s/Mpc
- 24-bit resolution: $\Delta_T \approx 0.18$, Gap ≈ 0.45 km/s/Mpc
- 32-bit resolution: $\Delta_T < 0.05$, Gap < 0.01 km/s/Mpc

Final concordance typically exceeds 99.8%.

2 API Access

2.1 Requesting API Credentials

API access is available through authenticated token-based authorization. Tokens can be requested via the web interface or programmatically.

2.1.1 Web Interface

Visit: https://allyourbaseline.com/multiresolution-uha-api

Complete the form with:

- Full name and institutional affiliation
- Email address
- Access tier selection (Academic/Commercial/Enterprise)
- Research use case description
- Estimated daily API call volume

2.1.2 Programmatic Token Request

Listing 1: cURL token request

```
curl -X POST https://got.gitgap.org/api/request-token \
  -H "Content-Type:␣application/json" \
  -d '{
␣␣␣␣"name":␣"Dr.␣Jane␣Smith",
␣␣␣␣"institution":␣"University␣of␣Example",
␣␣␣␣"email":␣"jane.smith@example.edu",
␣␣␣␣"access_tier":␣"academic",
␣␣␣␣"use_case":␣"Hubble␣tension␣systematic␣bias␣correction",
␣␣␣␣"daily_limit":␣100
␣␣}'
```

2.1.3 Response Format

Listing 2: Token response

```
{
  "success": true,
  "token": "uha.admin.Fp4WStMJiEc7OZWjUcdBnQ.Dr. Jane Smith (University of
      Example).read,write",
  "endpoint": "https://got.gitgap.org/v1/merge/multiresolution/",
  "daily_limit": 1000,
  "message": "Token created successfully. Documentation sent to jane.
     smith@example.edu."
}
```

2.2 Access Tiers

Tier	Daily Limit	Price	Use Case
Academic	1,000 calls	Free	Peer-reviewed publications
Commercial	10,000 calls	$5,000/year	Commercial research
Enterprise	100,000 calls	Contact	Large-scale analysis

Table 1: API access tiers and limits

3 API Specification

3.1 Endpoint

URL: `https://got.gitgap.org/v1/merge/multiresolution/`
Method: POST
Authentication: Token-based (required)
Content-Type: application/json

3.2 Request Format

3.2.1 Required Parameters

- `planck_chain`: MCMC posterior samples from Planck or equivalent CMB measurement
  - Format: 2D array [samples][parameters]
  - Minimum samples: 100
  - Maximum samples: 50,000
  - Parameters: Typically $[H_0, \Omega_m, \ldots]$
- `shoes_chain`: MCMC posterior samples from SH0ES or equivalent distance ladder measurement
  - Format: 2D array [samples][parameters]
  - Minimum samples: 100
  - Maximum samples: 50,000
  - Parameters: Typically $[H_0, \mathrm{RA}, \mathrm{Dec}, \mathrm{Distance}, \ldots]$

3.2.2 Optional Parameters

- `cosmo_params_planck`: Cosmological parameters for Planck
  - Default: {`h0: 67.4, omega_m: 0.315, omega_lambda: 0.685`}
- `cosmo_params_shoes`: Cosmological parameters for SH0ES
  - Default: {`h0: 73.04, omega_m: 0.300, omega_lambda: 0.700`}
- `resolution_schedule`: Array of Morton encoding bit depths
  - Default: $[8, 12, 16, 20, 24, 28, 32]$
  - Fast mode: $[8, 16, 24, 32]$
  - Fine mode: $[8, 10, 12, 14, \ldots, 30, 32]$

3.3 Request Example

Listing 3: Complete Python example

```python
import requests
import numpy as np

# API credentials
API_TOKEN = "your_token_here"
API_URL = "https://got.gitgap.org/v1/merge/multiresolution/"

# Generate or load MCMC chains
# (Replace with actual Planck/SH0ES posteriors)
planck_chain = np.random.normal(67.4, 0.5, (1000, 2))
shoes_chain = np.random.normal(73.04, 1.04, (1000, 4))

# Prepare request payload
```

```python
payload = {
    "planck_chain": planck_chain.tolist(),
    "shoes_chain": shoes_chain.tolist(),
    "cosmo_params_planck": {
        "h0": 67.4,
        "omega_m": 0.315,
        "omega_lambda": 0.685
    },
    "cosmo_params_shoes": {
        "h0": 73.04,
        "omega_m": 0.300,
        "omega_lambda": 0.700
    },
    "resolution_schedule": [8, 12, 16, 20, 24, 28, 32]
}

# Make API request
response = requests.post(
    API_URL,
    json=payload,
    headers={
        'Authorization': f'Token {API_TOKEN}',
        'Content-Type': 'application/json'
    },
    timeout=120
)

# Process response
if response.status_code == 200:
    result = response.json()

    print(f"Convergence: {result['convergence_achieved']}")
    print(f"Final Delta_T: {result['final_delta_t']:.4f}")
    print(f"Final Gap: {result['final_gap_km_s_mpc']:.2f} km/s/Mpc")
    print(f"Concordance: {result['final_concordance_pct']:.1f}%")
    print(f"Merged H0: {result['merged_h0']:.2f} +/- {result['
        merged_uncertainty']:.2f}")
else:
    print(f"Error {response.status_code}: {response.text}")
```

3.4 Response Format

3.4.1 Success Response (HTTP 200)

Listing 4: Complete response structure


```json
{
  "success": true,
  "convergence_achieved": true,
  "final_resolution_bits": 32,
  "final_delta_t": 0.0512,
  "final_gap_km_s_mpc": 0.01,
  "final_concordance_pct": 99.8,
```

```
  "results_by_resolution": [
    {
      "resolution_bits": 8,
      "cell_size_mpc": 3.90625,
      "delta_t": 1.20,
      "gap_km_s_mpc": 5.42,
      "concordance_pct": 10.0,
      "n_cells_planck": 245,
      "n_cells_shoes": 189,
      "tensor_planck": [0.95, 0.01, -0.02, -0.05],
      "tensor_shoes": [0.78, 0.02, -0.05, 0.50]
    },
    {
      "resolution_bits": 12,
      "cell_size_mpc": 0.244,
      "delta_t": 0.85,
      "gap_km_s_mpc": 3.18,
      "concordance_pct": 35.2,
      "n_cells_planck": 512,
      "n_cells_shoes": 387
    }
  ],

  "merged_h0": 70.22,
  "merged_uncertainty": 0.05,
  "merged_interval_low": 70.17,
  "merged_interval_high": 70.27,
  "processing_time_ms": 4523
}
```

3.4.2 Response Fields

Global Metrics

- `success`: Boolean indicating successful completion
- `convergence_achieved`: True if $\Delta_T < 0.15$
- `final_resolution_bits`: Highest resolution level processed
- `final_delta_t`: Epistemic distance at final resolution
- `final_gap_km_s_mpc`: Remaining H_0 discrepancy
- `final_concordance_pct`: Agreement percentage (0-100)

Merged Results

- `merged_h0`: Final calibrated Hubble constant (km/s/Mpc)
- `merged_uncertainty`: 1σ uncertainty
- `merged_interval_low`: Lower 95% confidence bound
- `merged_interval_high`: Upper 95% confidence bound

Per-Resolution Results For each resolution level in `results_by_resolution`:

- `resolution_bits`: Morton encoding bit depth
- `cell_size_mpc`: Physical cell size at this resolution
- `delta_t`: Epistemic distance between observer tensors
- `gap_km_s_mpc`: H_0 gap at this scale
- `concordance_pct`: Concordance at this resolution
- `n_cells_planck/shoes`: Number of occupied spatial cells
- `tensor_planck/shoes`: 4-component observer tensors $[P_m, \zeta_t, \zeta_m, \zeta_a]$

3.4.3 Error Responses

HTTP 400 - Bad Request

```
{
  "error": "Planck chain must have at least 100 samples",
  "status_code": 400
}
```

HTTP 401 - Unauthorized

```
{
  "error": "Authentication credentials were not provided",
  "status_code": 401
}
```

HTTP 429 - Too Many Requests

```
{
  "error": "Daily limit exceeded (1000 calls)",
  "error_code": "rate_limit_exceeded",
  "status_code": 429
}
```

4 Interpreting Results

4.1 Convergence Criteria

4.1.1 Successful Convergence

- $\Delta_T < 0.15$ (epistemic distance threshold)
- Final concordance $> 95\%$
- Final gap < 0.5 km/s/Mpc
- Monotonic decrease of Δ_T through resolutions

4.1.2 Partial Success

- $0.15 < \Delta_T < 0.5$
- Concordance $70 - 95\%$
- May require:
  - Increased sample sizes
  - Extended resolution schedule
  - Outlier removal

4.1.3 Non-Convergence

- $\Delta_T > 0.6$
- Concordance $< 70\%$
- Possible causes:
  - Insufficient sampling
  - Systematic errors in input chains
  - Incompatible datasets

4.2 Resolution Progression Analysis

The algorithm progressively refines spatial resolution. Typical progression:

$$\text{Cell size}(b) = \frac{1000 \text{ Mpc}}{2^{b/3}} \tag{1}$$

where b is the number of bits per dimension.

Bits	Cell Size (Mpc)	Scale
8	3.906	Galaxy cluster
12	0.976	Small group
16	0.244	Galaxy halo
20	0.061	Sub-halo
24	0.015	Local ISM
28	0.004	Molecular cloud
32	0.001	Star-forming region

Table 2: Spatial scales probed at each resolution

5 Best Practices

5.1 Data Preparation

5.1.1 MCMC Chain Requirements

1. **Convergence:** Ensure chains have converged (Gelman-Rubin $\hat{R} < 1.1$)

2. **Thinning:** Thin chains to remove autocorrelation (typical: keep every 10th sample)
3. **Burn-in:** Remove initial burn-in period (typical: first 20-30%)
4. **Sample size:** Optimal range: 1,000-10,000 samples per chain
5. **Outliers:** Remove samples with $|\chi^2 - \langle\chi^2\rangle| > 5\sigma$

5.1.2 Parameter Format

- **Planck chain:** Minimum: $[H_0, \Omega_m]$. Optional: additional cosmological parameters
- **SH0ES chain:** Minimum: $[H_0, \mathrm{RA}, \mathrm{Dec}, \mathrm{Distance}]$. RA/Dec in degrees, Distance in Mpc
- **Units:** H_0 in km/s/Mpc, distances in Mpc, angles in degrees

5.2 Resolution Schedule Selection

Schedule	Use Case	Typical Runtime
Fast	Quick validation	30-60 seconds
Standard	Production analysis	2-5 minutes
Fine	High-precision calibration	10-20 minutes

Table 3: Resolution schedule recommendations

5.3 Batch Processing

For multiple chain pairs, process sequentially:

Listing 5: Batch processing example

```python
import time

chains_to_process = [
    (planck_chain1, shoes_chain1, "Bootstrap 1"),
    (planck_chain2, shoes_chain2, "Bootstrap 2"),
    # ... more chains
]

results = []
for planck, shoes, label in chains_to_process:
    print(f"Processing: {label}")

    response = requests.post(
        API_URL,
        json={"planck_chain": planck.tolist(),
              "shoes_chain": shoes.tolist()},
        headers={'Authorization': f'Token {API_TOKEN}'},
        timeout=180
    )

    if response.status_code == 200:
```

```
        results.append(response.json())

    # Respect rate limits
    time.sleep(1)
```

6 Troubleshooting

6.1 Common Errors

6.1.1 Authentication Errors

```
# Error: "Authentication credentials were not provided"
# Solution: Include Authorization header
curl ... -H "Authorization:␣Token␣YOUR_TOKEN_HERE"
```

6.1.2 Chain Size Errors

```
# Error: "Planck chain must have at least 100 samples"
# Solution: Ensure sufficient samples
if len(planck_chain) < 100:
    raise ValueError(f"Insufficient samples: {len(planck_chain)}")

# Error: "Planck chain cannot exceed 50,000 samples"
# Solution: Downsample large chains
if len(planck_chain) > 50000:
    indices = np.random.choice(len(planck_chain), 50000, replace=False)
    planck_chain = planck_chain[indices]
```

6.1.3 Timeout Errors

```
# Solution: Increase timeout for large datasets
response = requests.post(..., timeout=300)  # 5 minutes
```

6.2 Performance Optimization

1. **Reduce sample count:** Use 1,000-5,000 samples for faster processing
2. **Use fast schedule:** $[8, 16, 24, 32]$ for initial validation
3. **Pre-compute coordinates:** Cache RA/Dec/Distance calculations
4. **Parallel requests:** Process independent chain pairs in parallel

7 Publication Guidelines

7.1 Citation

When using this method in publications, cite:

Systematic bias correction was performed using the multi-resolution Universal Horizon Address (UHA) tensor calibration method [1]. The implementation is available via authenticated API access at `https://got.gitgap.org/v1/merge/multiresolution/`. Contact the authors for API credentials.

The method employs progressive spatial refinement from coarse (8-bit) to fine (32-bit) Morton encoding precision, capturing systematic biases from local (< 1 Mpc) to global (> 100 Mpc) scales [2].

7.2 Example Methods Section

Multi-Resolution Systematic Bias Calibration. We employed a hierarchical spatial encoding scheme with progressive refinement at multiple scales to correct for systematic biases in the Hubble constant measurement [1].

The method encodes measurement locations using Morton Z-order curves with variable precision (8-32 bits per dimension), corresponding to physical scales from galaxy clusters (4 Mpc) to star-forming regions (<0.001 Mpc). At each resolution level, we construct 4-component observer tensors $\mathbf{T} = [P_m, \zeta_t, \zeta_m, \zeta_a]$ characterizing the local measurement context, where P_m represents the projection onto the measurement subspace and ζ_i quantify zero-inflation components.

The epistemic distance between CMB and distance ladder measurements,

$$\Delta_T = \sqrt{\sum_i (\mathbf{T}^i_{\text{CMB}} - \mathbf{T}^i_{\text{ladder}})^2} \tag{2}$$

decreases monotonically with increasing spatial resolution as systematic biases are progressively captured. Convergence is achieved when $\Delta_T < 0.15$, typically at 28-32 bit resolution.

Our Planck (N=10,000 samples) and SH0ES (N=5,000 samples) MCMC chains were processed through the standard resolution schedule $[8, 12, 16, 20, 24, 28, 32]$ bits, achieving final convergence with $\Delta_T = 0.048$, concordance = 99.8%, and merged $H_0 = 70.18 \pm 0.04$ km/s/Mpc.

7.3 References

References

[1] Martin, E.D. (2025). Multi-Resolution Universal Horizon Address System for Cosmological Systematic Bias Correction. *In preparation.*

[2] Martin, E.D. (2025). Multi-Resolution Universal Horizon Address System for Cosmological Systematic Bias Correction. US Patent Application 63/902,536.

8 Support and Contact

8.1 Technical Support

For technical issues, implementation questions, or bug reports:

- Email: `look@allyourbaseline.com`
- Include: API token (first 8 characters only), error message, chain sizes, timestamp

8.2 API Access

To request API credentials or increase rate limits:

- Email: `look@allyourbaseline.com`
- Web: https://allyourbaseline.com/multiresolution-uha-api
- Include: Name, institution, research description, estimated usage

8.3 Collaboration

For research collaborations or commercial licensing:

- Contact: Eric D. Martin
- Email: `look@allyourbaseline.com`

A Complete Working Example

Listing 6: Production-ready implementation

```python
#!/usr/bin/env python3
"""
Complete example: Multi-resolution UHA API usage
For publication-quality systematic bias correction
"""

import requests
import numpy as np
import json
from typing import Tuple, Dict, Any

# Configuration
API_TOKEN = "your_token_here"
API_URL = "https://got.gitgap.org/v1/merge/multiresolution/"

def load_planck_chain(filename: str) -> np.ndarray:
    """Load and prepare Planck MCMC chain"""
    # Load your actual chain data
    chain = np.loadtxt(filename)

    # Extract H0 and Omega_m columns (adjust indices as needed)
    h0 = chain[:, 0]
    omega_m = chain[:, 1]

    return np.column_stack([h0, omega_m])

def load_shoes_chain(filename: str) -> np.ndarray:
    """Load and prepare SH0ES MCMC chain"""
    # Load your actual chain data
    chain = np.loadtxt(filename)

```

```python
    # Extract H0, RA, Dec, Distance columns
    h0 = chain[:, 0]
    ra = chain[:, 1]
    dec = chain[:, 2]
    distance = chain[:, 3]

    return np.column_stack([h0, ra, dec, distance])

def preprocess_chain(chain: np.ndarray,
                     max_samples: int = 10000,
                     thin: int = 10,
                     burnin_frac: float = 0.3) -> np.ndarray:
    """Preprocess MCMC chain: burn-in, thinning, downsampling"""

    # Remove burn-in
    n_burnin = int(len(chain) * burnin_frac)
    chain = chain[n_burnin:]

    # Thin chain
    chain = chain[::thin]

    # Downsample if needed
    if len(chain) > max_samples:
        indices = np.random.choice(len(chain), max_samples, replace=False)
        chain = chain[indices]

    return chain

def call_multiresolution_api(
    planck_chain: np.ndarray,
    shoes_chain: np.ndarray,
    token: str,
    resolution_schedule: list = None
) -> Dict[str, Any]:
    """
    Call multi-resolution API with error handling

    Parameters
    ----------
    planck_chain : ndarray, shape (n_samples, n_params)
        Planck MCMC posterior samples
    shoes_chain : ndarray, shape (n_samples, n_params)
        SHOES MCMC posterior samples
    token : str
        API authentication token
    resolution_schedule : list, optional
        Custom resolution schedule

    Returns
    -------
    result : dict
        API response containing calibration results
    """

```

```python
    if resolution_schedule is None:
        resolution_schedule = [8, 12, 16, 20, 24, 28, 32]

    payload = {
        "planck_chain": planck_chain.tolist(),
        "shoes_chain": shoes_chain.tolist(),
        "cosmo_params_planck": {
            "h0": 67.4,
            "omega_m": 0.315,
            "omega_lambda": 0.685
        },
        "cosmo_params_shoes": {
            "h0": 73.04,
            "omega_m": 0.300,
            "omega_lambda": 0.700
        },
        "resolution_schedule": resolution_schedule
    }

    headers = {
        'Authorization': f'Token {token}',
        'Content-Type': 'application/json'
    }

    try:
        response = requests.post(
            API_URL,
            json=payload,
            headers=headers,
            timeout=300
        )
        response.raise_for_status()
        return response.json()

    except requests.exceptions.Timeout:
        raise TimeoutError(
            "API request timed out. Try reducing chain size or "
            "using a faster resolution schedule."
        )
    except requests.exceptions.HTTPError as e:
        raise RuntimeError(f"API error {e.response.status_code}: {e.response.text}")

def print_results(result: Dict[str, Any]) -> None:
    """Print formatted results"""

    print("=" * 70)
    print("Multi-Resolution Tensor Calibration Results")
    print("=" * 70)
    print()

    # Summary
    print(f"Convergence: {'YES' if result['convergence_achieved'] else 'NO'}")
```

```python
    print(f"Final Resolution: {result['final_resolution_bits']} bits")
    print(f"Epistemic Distance (Delta_T): {result['final_delta_t']:.4f}")
    print(f"H0 Gap: {result['final_gap_km_s_mpc']:.2f} km/s/Mpc")
    print(f"Concordance: {result['final_concordance_pct']:.1f}%")
    print()

    # Merged result
    print(f"Merged H0: {result['merged_h0']:.2f} +/- "
          f"{result['merged_uncertainty']:.2f} km/s/Mpc")
    print(f"95% CI: [{result['merged_interval_low']:.2f}, "
          f"{result['merged_interval_high']:.2f}]")
    print()

    # Resolution progression
    print("Resolution Progression:")
    print(f"{'Bits':<6} {'Cell(Mpc)':<12} {'Delta_T':<10} "
          f"{'Gap(km/s/Mpc)':<15} {'Concordance'}")
    print("-" * 70)

    for res in result['results_by_resolution']:
        print(f"{res['resolution_bits']:<6} "
              f"{res['cell_size_mpc']:<12.6f} "
              f"{res['delta_t']:<10.4f} "
              f"{res.get('gap_km_s_mpc', 0):<15.2f} "
              f"{res.get('concordance_pct', 0):<10.1f}%")

def main():
    """Main analysis pipeline"""

    print("Loading MCMC chains...")
    planck_raw = load_planck_chain("planck_chain.txt")
    shoes_raw = load_shoes_chain("shoes_chain.txt")

    print(f"Raw chains: Planck={len(planck_raw)}, SH0ES={len(shoes_raw)}")

    print("Preprocessing chains...")
    planck = preprocess_chain(planck_raw, max_samples=10000, thin=10)
    shoes = preprocess_chain(shoes_raw, max_samples=5000, thin=10)

    print(f"Processed: Planck={len(planck)}, SH0ES={len(shoes)}")

    print("\nCalling multi-resolution API...")
    result = call_multiresolution_api(planck, shoes, API_TOKEN)

    print_results(result)

    # Save results
    output_file = 'multiresolution_results.json'
    with open(output_file, 'w') as f:
        json.dump(result, f, indent=2)
    print(f"\nResults saved to: {output_file}")

if __name__ == "__main__":
    main()
```

B Acknowledgments

This work is supported by All Your Baseline LLC. The multi-resolution UHA method is protected under US Patent Application 63/902,536. We thank the research community for their continued interest and feedback.