ChatGPT

Sleep Performance Calculation Overview

Sleep performance is a composite score reflecting various aspects of sleep quality and quantity. We break it down into four key components and an overall score, as described below:

1. Hours vs. Needed (Sleep Quantity)

This metric compares the **total sleep obtained last night** to the **personalized sleep need** for that night. The personalized *Sleep Need* can be defined as:

Sleep Need = Baseline Sleep + Strain + Sleep Debt – Naps.

- **Baseline Sleep** is the base amount of sleep an individual generally needs (often around 7–8 hours for adults; WHOOP uses ~7.6 hours as a baseline [1]).
- **Strain** represents additional sleep required due to high exertion or stress during the day (e.g. intense workouts can increase needed sleep [2]).
- **Sleep Debt** is the accumulated deficit from prior nights of insufficient sleep [3]. If you slept less than needed in previous days, the deficit adds to tonight's requirement.
- **Naps** are subtracted, as daytime naps reduce the remaining sleep need for the night [3].

By incorporating these factors, the algorithm personalizes nightly sleep need [2]. The **Hours vs. Needed** score is then the percentage of last night's sleep achieved vs. this need. For example, if you slept 7 hours out of a needed 8 hours, this component would be 87.5%. A value around 100% means you met your sleep need, whereas below 100% indicates a shortfall (and above 100% would mean you exceeded the need).

2. Sleep Consistency (Bed/Wake Timing Variance)

Sleep consistency evaluates how regular your sleep schedule is, focusing on bedtimes and wake-up times consistency across days. In practice, we compare last night's sleep timing with the previous 4 nights to gauge variability. WHOOP defines sleep consistency as the likelihood of being asleep or awake at the same clock time across 4 consecutive days [4]. In simpler terms, if you go to bed and wake up at roughly the same times every day, your consistency is high.

To quantify this, we look at the variance in bedtimes and wake times over the last ~5 nights (last night + 4 prior nights). We calculate how much last night's bedtime and wake time deviated from the average of previous nights (or how much they shifted day-to-day on average). The **Sleep Consistency** score is then derived from that variance – smaller timing differences yield a higher score (better consistency), while large swings (e.g. staying up or sleeping in several hours later than usual) lower the score. This can be scaled to a 0–100 range for convenience, where 100 means perfectly consistent sleep times and lower values indicate irregular schedules.

3. Sleep Efficiency (Sleep Quality)

Sleep efficiency is the ratio of time actually slept ("total sleep time") to the total time spent in bed attempting to sleep [5]. It reflects sleep quality – a high efficiency means you fell asleep quickly and stayed asleep with minimal awakenings. We compute:

Sleep Efficiency = (Actual Sleep Time / Time in Bed) × 100% [5].

Here, *Time in Bed* is from the moment you went to bed to the moment you got up, and *Actual Sleep Time* sums only the periods you were asleep (excluding any time lying awake). For instance, if you were in bed for 8 hours but slept only 6 hours (due to awakenings or insomnia), your efficiency is 75%. Healthy young adults often have efficiency above 90% [6], whereas difficulties like insomnia yield lower efficiency. This metric is returned as a percentage. Improving sleep hygiene (consistent schedule, limiting disturbances) can increase sleep efficiency.

4. Sleep Stress (Physiological Stress During Sleep)

Sleep Stress gauges the level of physiological stress your body experienced during sleep. Even while asleep, stressors (like illness, anxiety, or nightmares) can manifest as **elevated heart rate**, **suppressed heart rate variability (HRV)**, or **increased respiratory rate**. Research shows that higher stress correlates with a **higher sustained heart rate and lower HRV** during rest [7]. In other words, a calm, low-stress sleep is usually marked by a low resting heart rate and higher HRV, whereas stress causes heart rate to spike and HRV to drop.

We estimate **high-stress time during sleep** by analyzing biometric data throughout the night: - **Heart Rate (HR)**: Periods where HR is significantly above your nightly baseline (e.g., above your average or normal sleeping HR) are flagged as potential stress episodes. - **Heart Rate Variability (HRV)**: Periods of abnormally low HRV (relative to your baseline) indicate stress activation of the nervous system [7]. - **Respiratory Rate**: An unusually high breathing rate during sleep (relative to your norm) can also signal stress or unrest.

Using these signals, the function estimates the cumulative duration (or percentage of the night) that your body was under high stress while asleep. The **Sleep Stress** component can be expressed as the percentage of sleep time that was stressful, or inverted into a score where 100 means no stress detected (ideal) and 0 would mean very high stress throughout the night. In our implementation, we derive a score out of 100 by subtracting the stress-time percentage from 100, so that higher values indicate more restful (less stressed) sleep.

With all components computed, we also provide an **overall Sleep Performance score**. This can simply be the average of the four component scores (each scaled 0–100). This overall score gives a quick holistic view of last night's sleep. For instance, if Hours vs. Needed, Consistency, Efficiency, and Stress all score in the 80–100 range, overall sleep performance would also be high. If one or more areas are low, the overall score will reflect that, highlighting room for improvement in that aspect of sleep.

TypeScript Implementation

Below is a TypeScript function that computes these metrics using Apple HealthKit data (via the `@kingstinct/react-native-healthkit` library). It fetches the necessary HealthKit samples – including sleep analysis, heart rate, HRV, and respiratory rate – and then calculates each component and the overall score. The code is modular and commented for clarity, following the style of typical usage examples of the HealthKit library:

```
import HealthKit, {
  HKCategoryTypeIdentifier,
  HKQuantityTypeIdentifier
  // HKUnit can be imported if needed for specifying units
} from '@kingstinct/react-native-healthkit';

interface SleepPerformanceMetrics {
  hoursVsNeeded: number;      // Percentage of sleep vs needed (0–100)
  sleepConsistency: number;   // Consistency score (0–100)
  sleepEfficiency: number;    // Sleep efficiency percentage (0–100)
  sleepStress: number;        // Inverted stress percentage (0–100, higher is
better)
  overallScore: number;       // Overall sleep performance (0–100)
}

/**
 * Calculates Sleep Performance metrics using HealthKit data.
 * Assumes HealthKit permissions have already been granted for sleep, heart
rate, HRV, and respiration data.
 */
async function calculateSleepPerformance(): Promise<SleepPerformanceMetrics> {
  // 1. Fetch the last 5 days of sleep analysis data (to evaluate consistency
and last night's sleep).
  const now = new Date();
  const startDate = new Date(now.getTime() - 5 * 24 * 60 * 60 *
1000); // 5 days ago
  const sleepSamples = await
HealthKit.getCategorySamples(HKCategoryTypeIdentifier.sleepAnalysis, {
    startDate,
    endDate: now
  });
  if (!sleepSamples || sleepSamples.length === 0) {
    throw new Error("No sleep data available");
  }

  // Helper to determine if a sleepAnalysis sample value represents "asleep" (as
opposed to in-bed or awake).
  // HealthKit sleepAnalysis values: 0 = inBed, 1 = asleep (unspecified), 2 =
```

```
awake, 3 = asleep (core), 4 = asleep (deep), 5 = asleep (REM).
  const isAsleepValue = (value: number) => {
    // Treat any value indicating sleep as asleep (value 1 or >=3 are sleep
stages)
    return value !== 0 && value !== 2;
  };

  // 2. Group sleep samples into contiguous sleep sessions (clusters).
  // This will separate main overnight sleep vs naps or separate sleep bouts.
  sleepSamples.sort((a, b) => new Date(a.startDate).getTime() - new
Date(b.startDate).getTime());
  interface SleepCluster { start: Date; end: Date; asleepMs: number; }
  const clusters: SleepCluster[] = [];
  let curCluster: SleepCluster = {
    start: new Date(sleepSamples[0].startDate),
    end: new Date(sleepSamples[0].endDate),
    asleepMs: isAsleepValue(sleepSamples[0].value)
              ? (new Date(sleepSamples[0].endDate).getTime() - new
Date(sleepSamples[0].startDate).getTime())
              : 0
  };

  // Define a gap threshold to start a new cluster (e.g., >3 hours gap breaks a
sleep session)
  const MAX_GAP_MS = 3 * 60 * 60 * 1000;  // 3 hours
  for (let i = 1; i < sleepSamples.length; i++) {
    const sample = sleepSamples[i];
    const sampleStart = new Date(sample.startDate);
    const sampleEnd = new Date(sample.endDate);
    if (sampleStart.getTime() - curCluster.end.getTime() <= MAX_GAP_MS) {
      // Continue the current cluster
      if (sampleEnd.getTime() > curCluster.end.getTime()) {
        curCluster.end = sampleEnd;
      }
      if (isAsleepValue(sample.value)) {
        curCluster.asleepMs += (sampleEnd.getTime() - sampleStart.getTime());
      }
    } else {
      // Gap too large – end current cluster and start a new one
      clusters.push(curCluster);
      curCluster = {
        start: sampleStart,
        end: sampleEnd,
        asleepMs: isAsleepValue(sample.value) ? (sampleEnd.getTime() -
sampleStart.getTime()) : 0
      };
    }
  }
```

```
  // Push the final cluster
  clusters.push(curCluster);

  if (clusters.length === 0) {
    throw new Error("Could not derive any sleep clusters");
  }

  // Identify the main sleep cluster for last night – usually the longest
cluster (by asleep duration)
  // (This assumes the longest sleep session in the last couple of days is the
primary overnight sleep.)
  let mainCluster = clusters[0];
  for (const cluster of clusters) {
    if (cluster.asleepMs > mainCluster.asleepMs) {
      mainCluster = cluster;
    }
  }

  // Calculate total actual sleep and time in bed for the main cluster
  const totalSleepMs = mainCluster.asleepMs;
  const timeInBedMs = mainCluster.end.getTime() - mainCluster.start.getTime();
  const totalSleepHours = totalSleepMs / (1000 * 60 * 60);
  const timeInBedHours = timeInBedMs / (1000 * 60 * 60);

  // 3. Compute "Hours vs. Needed"
  // Here we assume baseline, strain, sleep debt, and naps are available from
elsewhere (could be inputs or calculated).
  // For demonstration, these are treated as variables. In practice, they might
come from user settings or prior computations.
  const baselineNeedHours = 8.0;      // e.g., baseline sleep need (hours) –
typically ~7-8 hours for an adult
  const todayStrainHours = 0.0;       // e.g., additional need from strain
(hours) – e.g., heavy exercise might add, say, 0.5h
  const sleepDebtHours = 0.0;         // e.g., carry-over sleep debt from
previous nights (hours)
  const napHours = 0.0;               // e.g., hours of naps taken today (that
count toward reducing need)
  const sleepNeedHours = baselineNeedHours + todayStrainHours + sleepDebtHours
- napHours;
  const hoursVsNeededPct = sleepNeedHours > 0
    ? Math.min(100, (totalSleepHours / sleepNeedHours) * 100)
    : 100;  // if no need (shouldn't happen realistically), treat as 100%.

  // 4. Compute "Sleep Consistency"
  // Use the last up to 5 main sleep clusters (including last night and 4
previous nights) for consistency calculation.
  // Filter clusters that likely represent main overnight sleeps (e.g., duration
>= 3h) to exclude short naps.
```

```
  const mainSleepClusters = clusters.filter(c => (c.asleepMs / (1000*60*60)) >=
3);
  // Sort by end time (chronological order)
  mainSleepClusters.sort((a, b) => a.end.getTime() - b.end.getTime());
  // Take the last 5 clusters (if fewer, use what is available)
  const recentClusters = mainSleepClusters.slice(-5);
  let consistencyScore = 100;
  if (recentClusters.length >= 2) {
    // Calculate day-to-day shifts in bedtimes and wake times
    let totalBedDiffMinutes = 0;
    let totalWakeDiffMinutes = 0;
    let countDiffs = 0;
    for (let i = 1; i < recentClusters.length; i++) {
      const prev = recentClusters[i - 1];
      const curr = recentClusters[i];
      // Difference in bedtime from one night to the next (deviation from ~24h
apart)
      const idealBedInterval = 24 * 60 * 60 * 1000; // 24h in ms
      const actualBedInterval = curr.start.getTime() - prev.start.getTime();
      const bedDiff = Math.abs(actualBedInterval - idealBedInterval);
      totalBedDiffMinutes += bedDiff / (1000 * 60);
      // Difference in wake-up time from one morning to the next
      const idealWakeInterval = 24 * 60 * 60 * 1000;
      const actualWakeInterval = curr.end.getTime() - prev.end.getTime();
      const wakeDiff = Math.abs(actualWakeInterval - idealWakeInterval);
      totalWakeDiffMinutes += wakeDiff / (1000 * 60);
      countDiffs++;
    }
    // Average differences in minutes
    const avgBedDiffMin = totalBedDiffMinutes / countDiffs;
    const avgWakeDiffMin = totalWakeDiffMinutes / countDiffs;
    // Combine bed and wake variability (simple average)
    const avgTimingDiffMin = (avgBedDiffMin + avgWakeDiffMin) / 2;
    // Scale to a 0–100 score (assuming 0 min diff = 100, and large diffs reduce
score).
    // Here we deduct 1 point for each 6 minutes of timing variance as an
example (so ~2 hours variance => ~20 points off).
    consistencyScore = Math.max(0, 100 - (avgTimingDiffMin / 6));
  }

  // 5. Compute "Sleep Efficiency"
  const sleepEfficiencyPct = timeInBedMs > 0
    ? Math.min(100, (totalSleepMs / timeInBedMs) * 100)
    : 0;

  // 6. Compute "Sleep Stress"
  // Fetch heart rate, HRV, and respiratory rate samples during the main sleep
period:
```

```
  const start = mainCluster.start;
  const end = mainCluster.end;
  const hrSamples = await
HealthKit.getQuantitySamples(HKQuantityTypeIdentifier.heartRate, { startDate:
start, endDate: end });
  const hrvSamples = await
HealthKit.getQuantitySamples(HKQuantityTypeIdentifier.heartRateVariabilitySDNN,
{ startDate: start, endDate: end });
  const respSamples = await
HealthKit.getQuantitySamples(HKQuantityTypeIdentifier.respiratoryRate, {
startDate: start, endDate: end });

  // Extract values from samples
  const hrValues: number[] = hrSamples ? hrSamples.map(s => s.quantity) : [];
  const hrvValues: number[] = hrvSamples ? hrvSamples.map(s => s.quantity) : [];
  const respValues: number[] = respSamples ? respSamples.map(s => s.quantity) :
[];

  let stressPct = 0;
  if (hrValues.length > 0) {
    // Determine elevated heart rate threshold (e.g., 90th percentile of HR
during sleep)
    const sortedHR = [...hrValues].sort((a, b) => a - b);
    const hrThreshold = sortedHR[Math.floor(0.9 *
sortedHR.length)];  // top 10% HR
    const highHrCount = hrValues.filter(hr => hr > hrThreshold).length;
    const highHrFraction = highHrCount / hrValues.length;
    stressPct = highHrFraction * 100;
  }
  // (Optional) Incorporate HRV and respiratory data into stress calculation:
  if (hrvValues.length > 0) {
    // Low HRV can indicate stress – find 10th percentile of HRV as threshold
    const sortedHRV = [...hrvValues].sort((a, b) => a - b);
    const hrvThreshold = sortedHRV[Math.floor(0.1 * sortedHRV.length)];
    const lowHrvCount = hrvValues.filter(v => v < hrvThreshold).length;
    const lowHrvFraction = lowHrvCount / hrvValues.length;
    // If a significant portion of the night had low HRV, increase stress
percentage
    stressPct += lowHrvFraction * 100;
    stressPct = Math.min(100, stressPct);
  }
  if (respValues.length > 0) {
    // Elevated respiration rate threshold (e.g., 90th percentile)
    const sortedResp = [...respValues].sort((a, b) => a - b);
    const respThreshold = sortedResp[Math.floor(0.9 * sortedResp.length)];
    const highRespCount = respValues.filter(r => r > respThreshold).length;
    const highRespFraction = highRespCount / respValues.length;
    // Add to stress percentage (weighted a bit less than HR perhaps)
```

```
    stressPct += highRespFraction * 50;  // e.g., give respiration half weight
    stressPct = Math.min(100, stressPct);
  }

  const sleepStressScore = Math.max(0, 100 -
stressPct);  // invert so that less stress = higher score

  // 7. Overall Sleep Performance (average of components, optionally weighted
equally)
  const overallScore = Math.round((hoursVsNeededPct + consistencyScore +
sleepEfficiencyPct + sleepStressScore) / 4);

  return {
    hoursVsNeeded: Math.round(hoursVsNeededPct),
    sleepConsistency: Math.round(consistencyScore),
    sleepEfficiency: Math.round(sleepEfficiencyPct),
    sleepStress: Math.round(sleepStressScore),
    overallScore
  };
}
```

Explanation of Implementation:

- We use `HealthKit.getCategorySamples` to retrieve sleep analysis samples (`HKCategoryTypeIdentifier.sleepAnalysis`) for the past 5 days. Each sample has a `value` indicating whether the user was in bed, asleep, awake, or in a specific sleep stage (e.g. core, deep, REM). We group these samples into sleep sessions (clusters) by merging continuous or closely spaced samples, thereby separating distinct sleep occurrences (e.g. nighttime sleep vs. a separate nap). We sum up the duration of all segments where the user was actually asleep (using `isAsleepValue` to check the category value).

- We identify the **main sleep session** (presumably the longest one) as last night's primary sleep. For that session, we compute **Total Sleep Time** and **Time in Bed**. These feed into:

- **Hours vs. Needed:** Using an assumed or provided sleep need (baseline + strain + debt – naps), we calculate what percentage of that need was met by the total sleep. In code, this is `hoursVsNeededPct`. (If sleep need is, say, 8 hours and the user slept 6 hours, this would be 75%.) We cap it at 100 since exceeding 100% just means all needed sleep was obtained (or surpassed).

- **Sleep Efficiency:** Calculated as `totalSleepMs / timeInBedMs * 100` (converted to percentage). This tells how efficiently the user slept while in bed [5]. We also cap this at 100 (100% efficiency would mean they were asleep the entire time in bed).

- For **Sleep Consistency**, we consider up to 5 recent main sleep sessions. We compute the day-to-day differences in bedtimes and wake times. The code calculates how much each night's bedtime and wake-up deviated from a perfect 24-hour rhythm relative to the prior night. These differences (in minutes) are averaged (for bedtimes and wake times) to get an overall measure of schedule

variability. We then convert that into a score out of 100 – in the code, we simply subtract a scaled version of the average difference from 100. If bed/wake times varied only by a few minutes on average, the consistency score will be very high (close to 100). If there was a large swing (several hours difference) in sleep schedule, the score will be much lower.

- For **Sleep Stress**, we fetch heart rate, HRV, and respiratory rate data during the main sleep. We then:

- Determine periods of **elevated heart rate** by finding the 90th percentile of HR values as a threshold. The fraction of heart-rate readings above this threshold indicates the portion of the night with unusually high HR (potential stress response or restlessness).
- Similarly, we identify **low HRV** periods (e.g. readings in the lowest 10th percentile of HRV) and **high respiratory rate** periods (90th percentile of breaths per minute). These are additional indicators of stress or arousal during sleep.
- We combine these by increasing the estimated stress percentage for any significant time spent in these elevated-stress states. (In the code, heart rate is given full weight, while respiratory rate is given a half weight, assuming HR and HRV are more direct stress indicators; these weightings can be adjusted based on empirical tuning.)

- Finally, we invert this to a stress score: if, for example, 10% of the night showed high stress signs, the Sleep Stress score would be 90 (out of 100). A night with virtually no signs of stress would score near 100, whereas a highly restless night (e.g. 50% of time showing stress responses) would score around 50.

- The **Overall Sleep Performance** is then computed as the average of the four component scores. We round the individual metrics and overall score to whole numbers for presentation. (In an actual app, you might choose to weight these components differently or leave them as precise decimals, but equal weighting and simple averaging is a straightforward approach.)

This function returns a structured object with each metric and the overall score, which can be easily integrated into a React Native app's UI. Each component offers insight into different aspects of sleep: quantity vs. need, consistency of schedule, quality/efficiency, and physiological stress. By examining them individually and together, one can get a comprehensive view of their sleep performance and identify areas for improvement.

---

[1] [3] Whoop vs. RISE Sleep App: Which Sleep Tracker Is Best?
https://www.risescience.com/blog/whoop-vs-rise-sleep-app

[2] 10 (Hidden) WHOOP Features You Need to Know
https://www.whoop.com/us/en/thelocker/10-whoop-features-you-need-to-know/?srsltid=AfmBOorDQ8aejtfRBu3ZZWdagDoyE9JWz1o9DiHA4ZxwH4-tfG0lJN5Q

[4] The Importance of Sleep Consistency [+How to Measure]
https://www.whoop.com/us/en/thelocker/sleep-consistency-more-to-sleep-than-sleep-need/?srsltid=AfmBOooqVcQO1e8gS3MMaIKyv8qPqUBvl71CBUrxIuArjYNSycXBb6n8Y

[5] [6] Sleep efficiency - Wikipedia
https://en.wikipedia.org/wiki/Sleep_efficiency

7 Stress and heart rate variability: Relationship and management
https://www.medicalnewstoday.com/articles/stress-and-heart-rate-variability