Genesys Predictive Routing Deployment and Operations Guide

Routing scenarios using GPR

2/21/2026

Contents

- Administrator

Learn about Predictive Routing interaction flows, how the URS Strategy Subroutines work together to score agents and identify a routing target, how URS ranks agents by score, and how GPR handles agent reservation.

Related documentation:

-

If you would like to evaluate Genesys Predictive Routing for use with service-level routing or business-objective routing, contact Genesys Professional Services for a review of your routing environment.

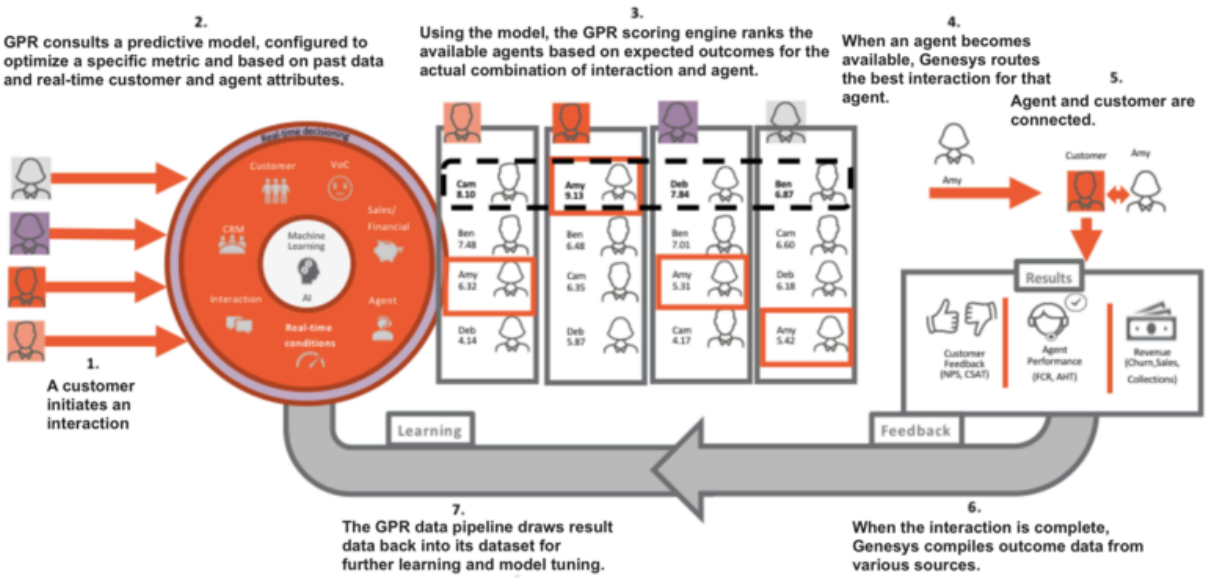
If your environment uses multiple URS instances receiving interactions from a single T-Server, the only criterion used to select the next interaction for routing is priority.

This topic assumes that you are using a virtual agent group (VAG) as the target for your routing. If you route using a skill expression to identify your targets, convert it to a VAG string expression using the IRD MultiSkill or CreateSkillGroup function before passing the resulting string as an argument to the ActivatePredictiveRouting subroutine. See Using Agent Skills for Ideal Agent Selection in the *Supplement to the Universal Routing 8.1 Reference Manual* for more information.

High-Level Predictive Routing interaction flow

The graphic in this section shows a very general interaction flow using Predictive Routing. Refinements to the flow depend greatly on details of your environment. Key aspects that differ in various environments:

- Your data - That is, the interaction types supported and the applications that might have relevant information. Genesys Info Mart is a key data source, but CRM systems and other applications in your environment can also provide important data. See Set up data for import for more information.
- Your pre-routing data flow - This depends on the interaction type and the exact architecture in your environment. For example, is this a chat interaction or a call? Do you use an IVR, and if so, what information do you attach?
- The Genesys routing solution you are using - Predictive Routing supports routing with IRD/URS.
- Your reporting solution for Predictive Routing - Whether you are using GCXI, Genesys Pulse, or another solution to present the data stored in Genesys Info Mart.



How the Strategy Subroutines work

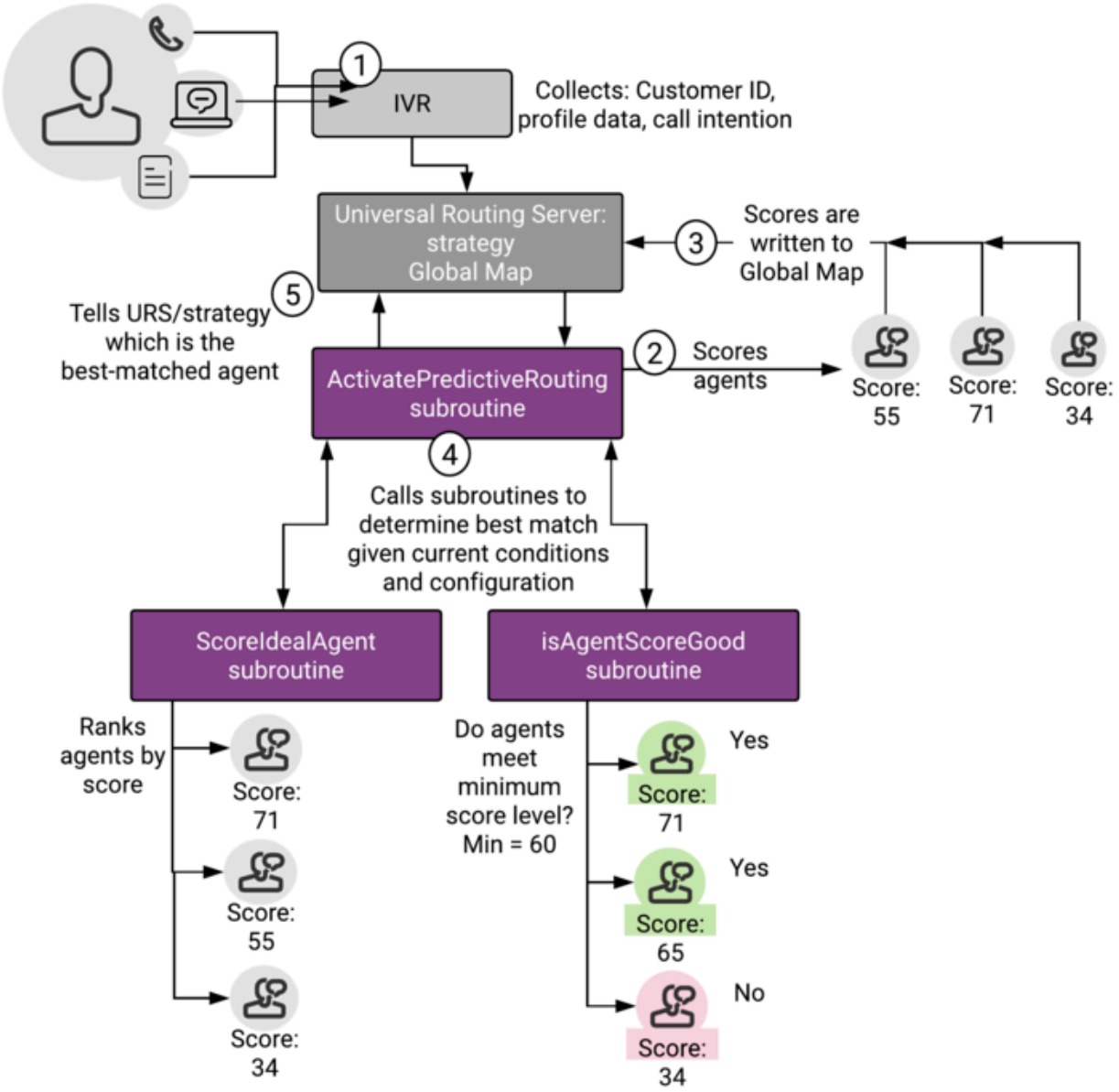
The following sequence provides a basic overview of the way the various GPR subroutines work together to evaluate agent scores and determine the best match given the currently available agents and the currently waiting interactions.

1. ActivatePredictiveRouting retrieves agent scores from the GPR Core Platform via REST API request and stores them in the global map in URS memory. The name of the map is the interaction ConnectionId (the original ID, if the interaction is a consult). This map contains pairs of agent employee IDs as the keys and their scores for the interaction as values. ActivatePredictiveRouting calls the SetIdealAndReadyCondition subroutine for further interaction processing.
2. SetIdealAndReadyCondition processes the different modes of Predictive Routing. It calls the SetIdealAgent IRD function to schedule the execution of the URS callback subroutines. It calls the ScoreIdealAgent subroutine to facilitate interaction queueing according to their scores, and calls SetReadyCondition (if enabled) to call the isAgentScoreGood subroutine.
   The parameters for these callback subroutines are retrieved and verified before any URS request is invoked to enable the callbacks.
3. After establishing a list of potential targets based on the target expression (Skill, Agent Group, and so on) SetIdealAndReadyCondition then executes the ScoreIdealAgent callback subroutine.
4. ScoreIdealAgent retrieves the scores for the potential target agents from the global map set in Step 1.
5. When an agent becomes ready, URS executes the isAgentScoreGood subroutine to determine whether that target is acceptable. If you enabled agent hold-out, URS executes the isAgentScoreGood subroutine when an agent becomes ready, which determines whether that agent reaches the specified threshold score. If not, URS waits for a configured timeout period, then checks whether any agent now satisfies the adjusted threshold value. See How Does GPR Score Agents? for a detailed discussion of how agent hold-out routing works.
6. Once the isAgentScoreGood subroutine locates an available agent who scores above the current threshold, it sends the target details to URS to initiate routing.

7. URS calls the GPRIxnCompleted subroutine as a custom step from a Routing Block in the strategy. It collects the Predictive Routing outcome for the successfully routed interaction (the DBID of the agent to whom it was distributed, the score of the agent, other interaction statistics relevant for the Predictive Routing performance) and prepares the user data for Predictive Routing reporting.
8. URS calls the GPRIxnCleanup subroutine from both the success and failure exits from the Routing block, or if the interaction is abandoned. The purpose of the subroutine is to publish the Predictive Routing reporting user data and to clean up the ScoreIdealAgent and isAgentScoreGood callback subroutines. GPRIxnCleanup publishes reporting data in two ways:
    - It sends a UserEvent containing the user data relevant for Predictive Routing to T-Server/SIP Server, from which is enters the Genesys historic reporting solution flow.
    - It can submit the same data AI Core Services via REST API request where it is stored in the score_log and can be retrieve using an API request.

Important

If your routing strategy uses the SelectDN and SuspendDN IRD functions instead of a Routing Block, consult with Genesys Professional services about how the ActivatePredictiveRouting, GPRIxnCompleted and GPRIxnCleanup subroutines can be integrated into your strategy.



Routing scenarios using Predictive Routing

When you are using Predictive Routing to route interactions, there are two main scenarios that affect how this matching plays out:

- **Agent Surplus** - There are relatively few interactions, which means there could be a number of high-score agents available. You can configure a minimum threshold so that, if the agents available are not very highly ranked, the strategy keeps the interaction in queue until a better-scoring agent becomes

available.

- **Interaction Surplus** - There are many interactions, so that most agents are busy and it might be more difficult to find an ideal agent for each interaction. In such a scenario, you can have agents matched to the interaction for which they have the highest probability of getting a positive result.

Agent Surplus Flow

In this case there are agents logged in and in the Ready state who can respond to interactions immediately. From a Virtual Agent Group that is defined by skill expression, URS first tries to route an interaction to an agent with the best score, using the following process to match agents and interactions:

1. An interaction arrives at the routing strategy, which has a target group of agents.
2. The ActivatePredictiveRouting subroutine sends a request to the Predictive Routing scoring server via HTTP request.
3. Predictive Routing returns scores for each agent in the target group based on the criteria you selected in the active model.
4. The ActivatePredictiveRouting subroutine updates a global cache in URS memory, which keeps agent scores for all interactions. When URS tries to route the current interaction to the agent group, it sorts the agents according to their scores, in descending order, and routes to the agents with the best score first.

When URS takes an interaction from the queue:

1. URS calls the ScoreIdealAgent subroutine, which reads the agent scores in the target group from global map and ranks the agents by score.
2. URS calls the IsAgentScoreGood subroutine, which selects the available agent with the highest score, assuming the agent has a score high enough to be selected for this interaction.
   In an agent-surplus scenario, it is typically not a problem to route to an agent with a good score. For scenarios where this is not the case, see **Interaction Surplus Flow**, below.
3. URS calls the GPRIxnCompleted subroutine, which updates user data with the scoring result for storage in Genesys Info Mart.
4. URS calls the PRRLog macro, which logs the result in the URS log file.

Interaction Surplus Flow

This scenario covers situations when all agents are already busy handling interactions and new interactions are queued. When one of the agents becomes ready, the system selects the interaction for which the agent has the best score. This is not necessarily the interaction that has been in the queue longest.

When interactions are waiting, URS uses a number of criteria to decide the order in which it directs the interactions to the best target. In general, URS uses the following hierarchy:

1. Interaction priority.
2. Best agent score.

> **Important**
>
> In scenarios where both scored and unscored interactions might have the same priority, scoring is disregarded for all the interactions and the selection is based on the next differentiating criterion, time.

3. Time in queue, which can be based on age of interaction or time in queue and can incorporate predicted wait time.
4. Interaction ID (URS selects the interaction with the lowest—oldest—ID). This is a rarely-used "tie-breaker" criterion.

Using gpmStatus and gpmSuitableAgentsCount to Monitor Your Routing

gpmStatus and gpmSuitableAgentsCount are KVP values written in the Genesys Info Mart database when an interaction is routed using the GPR subroutines. (You can also retrieve the values by using the GPR API to query the score log.)

- gpmStatus indicates whether there was an agent-surplus or an interaction-surplus condition when the interaction was routed.
- gpmSuitableAgentsCount indicates the number of agents who have scores returned from AICS greater than or equal to the initial threshold value when the scoring response is received. If gpmSuitableAgentsCount is `0`, then no agents have eligible scores compared with the threshold value, so the interaction must wait for a higher-scoring agent to become available or for the next threshold relaxation step

These KVP values, when analyzed for different interactions over a representative day or week period can help you understand your contact center traffic and GPR performance. The following table indicates certain scenarios and how to interpret them.

KVP Values	Inference
gpmStatus = caller-surplus gpmSuitableAgentsCount > 0	Your GPR Model is returning useful scores with relation to the configured routing threshold, but agent staffing is not adequate to produce satisfactory wait times.
gpmStatus = agent-surplus gpmSuitableAgentsCount > 0 or consistently a very small number	Analyze why the scores GPR returns are not meeting the configured threshold. You might need to retrain your Model, adjust the scoring expression, or reduce the threshold level.

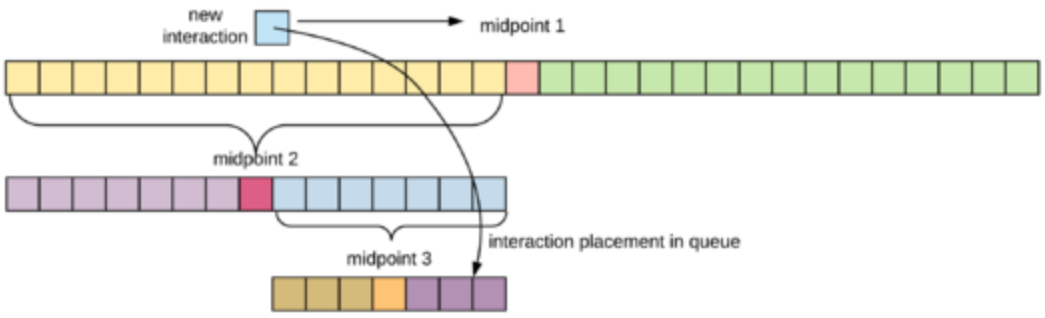
How Interactions are Sorted within the Queue

As each interaction comes in, it is scored, and then assessed relative to the interaction at the midpoint of the existing array of interactions. Should GPR route it before or after the mid-point interaction?

> **Important**
>
> The order in which interactions are prioritized is called an *array* here. This is not equivalent to a queue. These interactions might be from multiple queues, each of which is submitting interactions for URS sorting and routing.

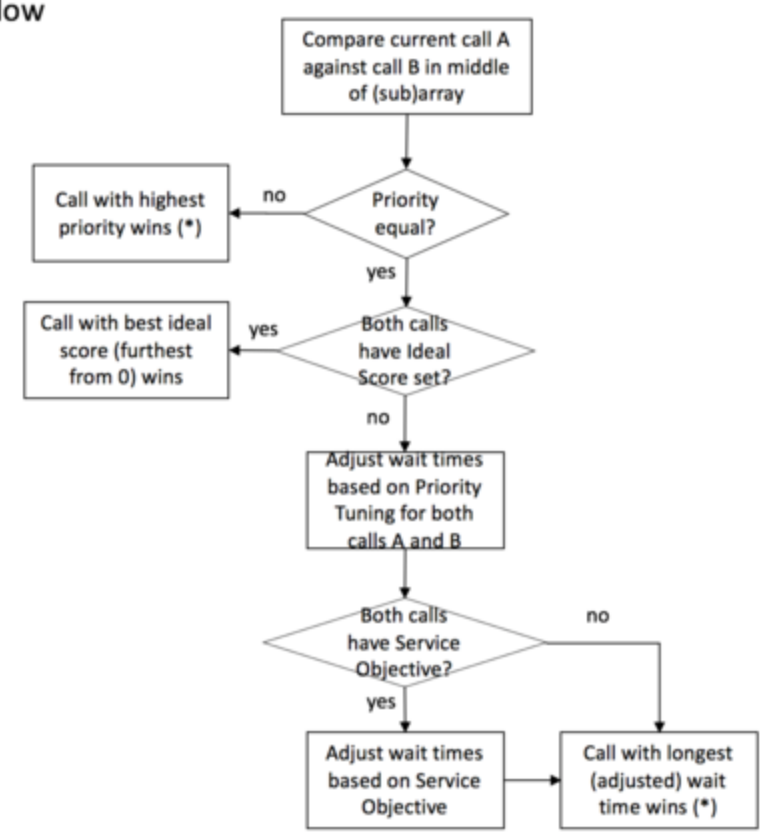
After this decision, URS compares the new interaction against the midpoint within the selected region. Each time URS evaluates the interaction, it is assigned to a smaller region with the total array, always relative to the midpoint of the previous region.

> **Important**
>
> The order in which interactions are prioritized is called an *array* here. This is not equivalent to a queue. These interactions might be from multiple queues, each of which is submitting interactions for URS sorting and routing.



The sorting decision tree:

flow



Example 1

Three calls arrive at a contact center:

- C1 - priority = 1, agent score = 0.3, timestamp = 0:00, URS ID = 1
- C2 - priority = 1, agent score = n/a, timestamp = 0:05, URS ID = 2
- C3 - priority = 1, agent score = 0.6, timestamp = 0:10, URS ID = 3

1. C1 arrives first and is placed into the empty array.
2. C2 arrives. URS compares it with the middle (in this case, only) call in the array, C1.
   The priority is equal and, because C2 has no agent score, URS moves to the next decision criterion.
3. C2 has a shorter wait time, so is put behind C1.
   With only two calls in the array, no further comparison is needed.
4. C3 arrives. URS compares it against the "middle" entry of the array, C1.
   The priority is equal. C3 has better score (further from 0), so URS puts it in front of C1.

Outcome: C3, C1, C2 (the example assumes that interactions are taken from the left end of the array)

Example 2

Five calls arrive at a contact center and are placed in either the Predictive Routing queue or a

conventional queue:

- C1 - priority = 1, agent score = n/a, timestamp = 0:00, URS ID = 1
- C2 - priority = 1, agent score = 0.5, timestamp = 0:05, URS ID = 2
- C3 - priority = 1, agent score = n/a, timestamp = 0:10, URS ID = 3
- C4 - priority = 1, agent score = 0.75, timestamp = 0:15, URS ID = 4
- C5 - priority = 1, agent score = 0.95, timestamp = 0:20, URS ID = 5

1. C1 arrives first. URS places it into the empty array.
2. C2 arrives. URS compares it with the middle (in this case, only) call in the array, C1.
   The priority is equal and, because C1 has no agent score, URS moves to the next decision criterion.
3. C2 has a shorter wait time, so is put behind C1. (In this example,
   Current order: C1 C2 (the example assumes that interactions are taken from the left end of the array)

   With only two calls in the array, no further comparison is needed.
4. C3 arrives. URS compares it against the "middle" entry of the array, C2.
   The priority is equal and, because C3 has no agent score, URS moves to the next decision criterion.
5. C3 has a shorter wait time, so is put behind C2.
   Current order: C1 C2 C3
6. C4 arrives. URS compares it against the middle entry of the array, C2.
   The priority is equal. C4 has a better score (further from 0), so URS places it before C2.
7. Now URS must determine whether C4 should be before or after C1, which is also before C2.
   The priority is equal and, because C3 has no agent score, URS moves to the next decision criterion.
8. C4 has a shorter wait time (a more recent timestamp), so URS places it behind C1.
   Current order: C1 C4 C2 C3
9. C5 arrives. URS compares it against the "middle" entry of the array, C2.
   The priority is equal. C5 has a better score (further from 0), so URS places it before C2.
10. Now URS must determine whether C5 should be before or after C4, the "middle" call in the section of the array before C2.
    The priority is equal. C5 has a better score, so URS places it before C4.
11. Now URS must determine whether C5 should be before or after C1.
12. C5 has a shorter wait time, so URS places it behind C1.
    Final order: C1 C5 C4 C2 C3

Using Agent Hold-Out

Agent hold-out enables you to have an interaction wait a specified time, even when an agent has become available, if the available agent is has a low score for the interaction and there is a chance a better-matched agent might become available within the configured time window. The interaction flow is as follows:

1. URS calls the IsAgentScoreGood subroutine, which determines whether any of the available agents meet the threshold for handling the interaction.

2. If available agents have low scores for this interaction and the interaction spent only a short time in the queue, URS waits for a better agent to become ready.
3. The minimum acceptable score required for an agent for the interaction is gradually reduced, so if no higher-scored agent becomes available, the lower-scored agent might finally be given the interaction.

After that determination occurs, the remainder of the flow is the same as that given in the agent-surplus flow above. Use the relevant Predictive_Route_DataCfg Transaction List Object configuration options to set up the priority increments.

Dynamic Interaction Priority Increments

To avoid having interactions lingering in a queue for an excessive amount of time, URS can trigger an escalation in interaction priority after a time delay that you set. To speed up interaction handling, you can incrementally relax the minimum skill level required for agents to handle the interaction or expand the pool of agents to consider.

Each time a routing strategy tries to route an interactions, it calls the ActivatePredictiveRouting subroutine. After each failed routing attempt, the strategy checks how long the interaction has been waiting in the queue and, if the time in queue is above a certain threshold, it routes the interaction to the next available agent, no matter their score for the interaction.

Use the relevant Predictive_Route_DataCfg Transaction List Object configuration options to set up the priority increments.

Using GPR with agent reservation

When your Genesys environment contains multiple URS nodes, they might compete to distribute interactions to the same pool of agents. URS uses *agent reservation* functionality to resolve which interaction should be routed to a particular agent. For details, see the **agent_reservation** option in the *Universal Routing 8.1 Reference Manual*.

Note: If you are using Service Objective, or Prediction/What-if routing, consult Genesys Customer Care for how to agent reservation works in these scenarios.

For agent reservation, a URS node sends an agent reservation request to a dedicated SIP Server/T-Server application with three extensions: priority, ar-priority-1, and ar-priority-2. The value for the extensions depends on whether this is a GPR interaction and what value you have set for the URS **automatic_ideal_agent** option.

For GPR interactions:

- **priority**: The current interaction priority.
- **ar-priority-1**: The score assigned by URS to the agent for this interaction calculated using the following formula: (100 - (*max-score*> -)). The value can be negative.
- **ar-priority-2**: The time the interaction spent in the strategy Target block, in milliseconds.

For non-GPR interactions when the URS **automatic_ideal_agent** option is set to a positive value (the recommended configuration):

- **priority**: The current interaction priority.
- **ar-priority-1**: The value for this extension is calculated as (100 - automatic_ideal_agent option>). The value can be negative.
- **ar-priority-2**: The time the interaction spent in the strategy Target block, in milliseconds.

For non-GPR interactions when the URS **automatic_ideal_agent** option is set to `false`:

- **priority**: The current interaction priority.
- **ar-priority-1**: The integer part of the time spent by the interaction in the strategy Target block, in seconds.
- **ar-priority-2**: The fractional part of a second of the time spent by the interaction in the strategy Target block, in milliseconds (value range is 0-999).

Example 1

Two URS nodes send concurrent agent reservation requests to SIP Server/T-Server targeting the same agent to handle GPR interactions. When SIP Server/T-Server resolves this race condition, it first compares the interaction priority values (the values of the **priority** Extension key). If they are equal, SIP Server/T-Server compares the values of the **ar-priority-1** keys, which were calculated based on agent scores for the interaction. If those values are also the same, SIP Server/T-Server assigns the interaction waiting longer in the Target block, based on the values of the **ar-priority-2** keys, to the agent. URS fails the routing attempt for the other interaction and the routing strategy handles rerouting it. GPR reports the failed routing attempt using the reporting key **gpmResult=14**.

Example 2a

Two URS nodes have the **automatic_ideal_agent** configuration option set to **45**. Predictive Routing has the ***max-score*** set to **100**. URS node 1 targets a GPR interaction to an agent with an **ar-priority-1** score of **60** for the interacrion. Concurrently, URS node 2 targets a non-GPR interaction to the same agent. The agent has an **ar-priority-1** score of **55** for that interaction. Based on these scores, the SIP Server/T-Server handling agent reservation assigns the GPR interaction from URS node 1 to the agent.

Example 2b

This example has a scenario similar to Example 2a, but the ***max-score*** and **automatic_ideal_agent** have different values.

Two URS nodes have the **automatic_ideal_agent** option set to **4500**. The Predictive Routing ***max-score*** option is set to **10000**. URS node 1 targets a GPR interaction to an agent for which the GPR Core Platform has assigned a score of **6000**. According to the formula for the calculation of ar-priority-1 (see the beginning of this section), the agent's adjusted value is **(100 - (10000 - 6000))=-3900**. Concurrently, URS node 2 targets a non-GPR interaction to the same agent, who has an ar-priority-1 value of **(100 - 4500)** = **-4400** for that non-GPR interaction. Based on these values, the SIP Server/T-Server handling agent reservation assigns the GPR interaction from URS node 1 to the agent.

Example 3

Two URS nodes have the **automatic_ideal_agent** option set to **false**. Predictive Routing has the ***max-score*** set to **100**. URS node 1 targets a GPR interaction, Interaction 1, to an agent for which the GPR Core Platform has assigned a score of **60**. Concurrently, URS node 2 targets a non-GPR

interaction, Interaction 2, to the to the same agent, who has the same ar-priority-1 value of **60** for that non-GPR interaction. Interaction 1 spent 67.3 seconds in the strategy Target block while Interaction 2 spent 180.5 seconds there. The SIP Server/T-Server handling agent reservation receives the requests: one from URS node 1 with **ar-priority-1=60**, **ar-priority-2=67300** and the other from URS node 2 with **ar-priority-1=180**, **ar-priority-2=500**. As a result, Interaction 2 is routed to the agent.

Configure A/B comparison test ratios

To test how GPR handles routing on your queues compared with skills-based routing, you can configure an A/B (comparison) test. GPR handles a specified percentage of interactions (GPR on) and the rest are routed by your original routing method (GPR off).

To start using comparison testing, set the ***prr-mode*** option to **ab-test-time-sliced**. This turns on comparison testing mode.

Start A/B testing

Genesys recommends the following sequence:

1. Start with a fourteen-day, 50/50, hour-on hour-off comparison test in your test environment.
2. Based on a satisfactory result, turn GPR on for all interactions in your production environment.
3. (Optional) - To check that GPR continues to generate improvement, run a non-50/50 comparison test in your production environment using a ratio such as 90/10 or 80/20. Such a split enables you to retain most of the benefit from using GPR while verifying GPR performance compared with traditional routing.

Instructions for configuring both 50/50 and non-50/50 time splits follow.

Notes on how A/B testing works

- When configuring A/B test periods, Genesys recommends that the time slices should be no shorter than 3600 seconds (one hour) and, if you use longer time slices, that they should be multiples of one hour.
- To ensure that the A and B test slices are comparable, GPR alternates which routing method is on during a specific time slice, both daily and weekly.
  - **Example 1** - With an hour-on, hour-off 50/50 test cycle (the default setting), Monday 8-9 uses A, Tuesday 8-9 uses B. The slices are also switched weekly, so that in week 1 Monday 8-9 is A and in week 2, Monday 8-9 is B. By the end of a two-week comparison period, each routing method has been used for each time-slice period.
  - **Example 2** - With an eight hours on, two hours off 80/20 test cycle (as explained in the "Configure a non-50/50 comparison test time split" section, below), Monday midnight - 8 am uses A, 8-10 uses B, and so on. In this pattern, the times of day routed using each method shift automatically. This shift produces a full cycle that uses each routing method for each time-slice period. Assuming your environment is active twenty-four hours, a single full cycle takes six days. You must then adjust the length of the comparison test depending on the nature of your environment and the KPI you specify.
- A comparison test should always run at least fourteen days. In most cases, you must run the test significantly longer to ensure that you produce consistent results. The length of a comparison test should take the following factors into account:

- **KPI type** - A KPI such as AHT, which returns an immediate result, requires a shorter test period than a KPI, such as first contact resolution, that takes time before the KPI outcome for an interaction is available.
- **Time ratio configured** - Non-50/50 time splits require longer comparison tests. The further from 50/50, the longer the comparison test should be to achieve solid results.
- **Improvement with GPR** - The stronger the percent improvement using GPR, the shorter the comparison test period can be.
- **Example** - In a test of AHT showing a 3% benefit using GPR, Genesys recommends that you run a test with an 80/20 split for 30 days, and a 90/10 split for 44 days.

Note: A/B comparison testing is based on time-on/time-off segments. The number of calls routed by each method might not exactly match the specified percentage because the number of calls coming into the queue might vary throughout the comparison test period.

Configure a 50/50 comparison test time split

For a 50/50 comparison test split, set the ***ab-test-time-slice*** option to a positive value. This is the time, in seconds, for each routing method to run (the time slice).

If you do not configure this option, A/B test periods are hourly by default. For more information, see the complete description of the ***ab-test-time-slice*** option.

- Genesys recommends that your time slice be at least 3600 seconds in a production environment.
- For robust results, run a 50/50 comparison test time split for at least 14 days with the times slices no shorter than one hour on, one hour off (this is the default setting).

Formula Used

GPR uses the following formula to determine the GPR Mode for a particular call when you configure a 50/50 test time split:

```
a = varStartTS mod (varABTestTimeSlice * 2)
if a = varABTestTimeSlice = GPR ON
```

where

- `varABTestTimeSlice` refers to the value set for the option, **ab-test-time-slice**.
- `varStartTS` refers to the date and time at which the interaction began as a Coordinated Universal Time (UTC) value. The UTC time is formatted as Unix time value.
  For example, if the call start time is *Thursday, 5 May 2022 5:30:00 AM GMT+05:30*, it is converted to UTC time as *Thursday, 5 May 2022 12:00:00 AM*. Then the corresponding Unix timestamp is identified, in this case, *1651708800* and will be used as the `varStartTS` value.

Using this start timestamp and when the **ab-test-time-slice** option is set to 86400 seconds (24 hours), the result for the calculation will enable GPR for all interactions coming on this day. For the next day, GPR will be turned off for the whole day.

Default Configuration

By default, the ***ab-test-time-slice*** option is set to zero. This will toggle GPR on and off every hour. The first hour, GPR will be enabled and in the second hour, it will be disabled and in the third hour, GPR will be re-enabled.

The formula used in the default configuration is: `varABTestTSUsePredictive = (dayOfyear % 2 + hourOfDay % 2 ) % 2` where `dayOfyear` indicates the number of days elapsed since January 1 of that year and the interaction date and `hourOfDay` indicates the hours elapsed since midnight.

Consider an example scenario where the **timestamp** value is *1653052118*, the **dayofyear** value is *140*, and the **hourofday** value is *13*, applying the formula gives the values:

- dayofyear%2 = 0
- hourofday%2 = 1
- varABTestTSUsePredictive = 1

In this scenario, GPR is enabled.

Similarly, consider an example scenario where the **timestamp** value is *1653049472*, the **dayofyear** value is *140*, and the **hourofday** value is *12*, applying the formula gives the values:

- dayofyear%2 = 0
- hourofday%2 = 0
- varABTestTSUsePredictive = 0

In this scenario, GPR is disabled.

Configure a non-50/50 comparison test time split

If you would like to move GPR into production but monitor the benefit at the same time, Genesys recommends using a non-50/50 split. To set the comparison test time periods to a non-50/50 split, specify values that produce the desired time split in the following two configuration options:

- ***ab-test-gpr-on-period***
- ***ab-test-gpr-off-period***

If you leave these options at the default settings, the comparison test defaults to a 50/50 split, with interactions routed half the time using GPR and half using skills-based routing. The on/off periods (time slices) are defined by the value set in the ***ab-test-time-slice*** option.

Example configuration for a 90/10 split

- **ab-test-gpr-on-period** - 60*60*9 (9 hours ON)
- **ab-test-gpr-off-period** - 60*60*1 (1 hour OFF)

Example configuration for an 80/20 split

- **ab-test-gpr-on-period** - 60*60*8 (8 hours ON)
- **ab-test-gpr-off-period** - 60*60*2 (2 hour OFF)

To get high-quality results from comparison testing, the test must run long enough to smooth out any unusual occurrences or statistical anomalies in the data. The further the split is set from a 50/50 ratio, the longer the comparison test period should be to achieve a high-quality test.

Formula Used

GPR uses the following formula to determine the GPR Mode for a particular call when you configure a non-50/50 test time split:

```
varStartTS mod (GPR_ON_PERIOD + GPR_OFF_PERIOD)
if a = GPR_OFF_PERIOD = GPR ON
```

where

- `GPR_ON_PERIOD` refers to the value set for the option, **ab-test-gpr-on-period**
- `GPR_OFF_PERIOD` refers to the value set for the option, **ab-test-gpr-off-period**
- `varStartTS` refers to the date and time at which the interaction began as a Coordinated Universal Time (UTC) value. The UTC time is formatted as Unix time value.
  For example, if the call start time is *Thursday, 5 May 2022 5:30:00 AM GMT+05:30*, it is converted to UTC time as *Thursday, 5 May 2022 12:00:00 AM*. Then the corresponding Unix timestamp is identified, in this case, *1651708800* and will be used as the `varStartTS` value.

Using this start timestamp and when the **ab-test-gpr-on-period** is set to *28800* seconds (8 hours) and **ab-test-gpr-off-period** to *7200* seconds (2 hours), applying the formula will result in disabling GPR for this interaction.